Railroad Dispatching Simulator

Joe Damico

CMPS 450: Senior Project

September 23, 2019

Part I: Introduction

For this project I decided to combine two of my interests; the first obviously being computer, and more specifically web applications. And the second being railroads which I've had a love for since I was a small child. Moreover, I always found dispatching (or controlling train movements) particularly interesting, because on the surface it seems like it would be an easy job. It actually takes a lot of mental power and planning ahead to make sure every train keeps moving. The mental challenge is what made me decide to create a simulator of a real dispatching desk. I modeled the simulator off of New Jersey Transits Main Line Desk, which controls the tracks between Laurel (located just west of the Secaucus Junction station) and CP Sparrow just west of Port Jervis NY. The tracks run through Mahwah so being here a Ramapo is seemed appropriate.

When it came time to decide how I was going to build this project, I gave myself a few criteria I wanted to meet. The first was I wanted to build it where the drawing the UI wouldn't be extremely difficult; the second was that ideally, I would like it to be cross platform, since there seems to be a ever growing divide between Windows and Mac. To check off both these boxes, I landed on a web application. Allowing it to work on any platform, and it will give me some experience with web development, which is a massive part of the industry. Once I decided on making the application as a web app, I had to choose a framework to create the app in. I first looked into Django which would be written in Python, but Django is more geared towards displaying data, which I great for most web applicants but since what I wanted to create is more like a game, it became clear the ReactJS would be a better choice, because it'll be easier to follow the "Model and View" idea for creating a game, similar to what is taught is the OPL class here.

ReactJS uses Javascript as the main language which was a I was somewhat familiar with after writing the extra credit project for OPL in Javascript for the scripting language project, although that game was created with just basic Javascript and HTML, instead of using a framework like ReactJS. I found that using one of these frameworks will likely help me not only in building the application, but in job interviews a well, since I noticed that all of the internship interviews, I went on asked me if I had any experience with any of these frameworks.

Part II: Installation Instructions

Option 1: There are two options for installing this simulator and running it. The first one is that I installed the application on an Amazon Web Service bucket, so you can just go to a link and it will ack like any other website game.

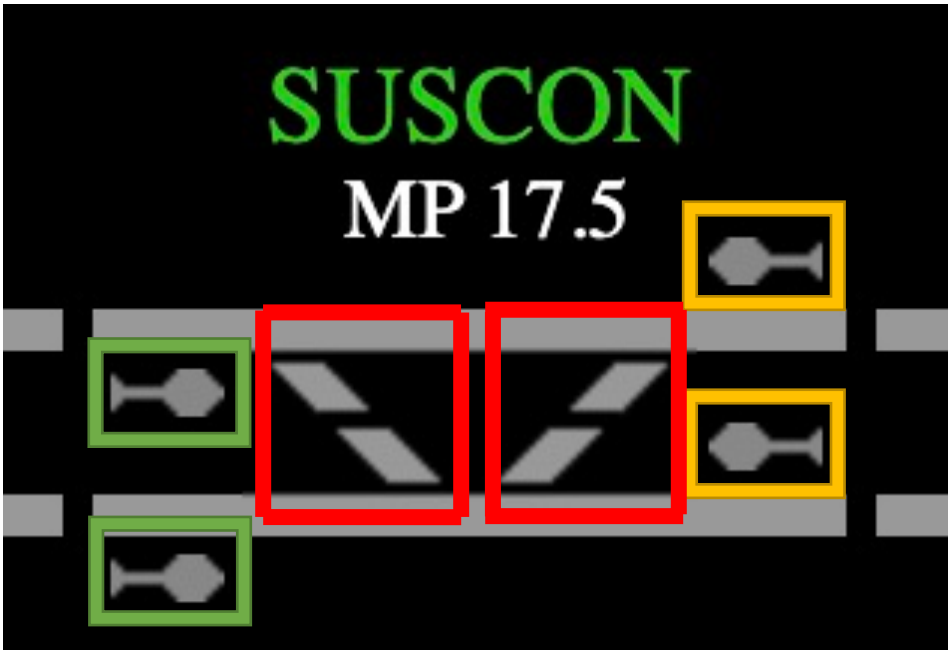
Link:

Option 2: You can install the ReactJS development server which will allow you to run the application in your browser at "Localhost:8080"

- Install the nodeJS package from their website (https://nodejs.org/en/) make sure to select the correct version for you system.
- Once installed open Terminal (or Command Prompt) and cd into the directory that holds the program that you copied off the USB stick provided.
- Once there run "npm start"
- Then everything should be running

Part III: User's Manual

This application works in the same way that program that is used by New Jersey Transit to actually control their railroad works, so if you can use this simulator then you'd be able to walk into the Operations Center in Kearny, NJ and dispatch the real railroad. The first thing you need to know is that going to the left is West and going to the right is East. All of the controls are in each interlocking, basically all you can control is which way the switches are facing, and which signal is lined, see the picture below for more information on how to dispatch.



What is in the Green and Yellow boxes are signals, they represent what is in the real world, that train crews used to know what there route is, think of them similar to traffic lights on the road, although these are not controlled by a timer, but my an actually person.

- The signals in the green boxes are the eastbound signals, these are used by trains that are heading east, so when you click one of these signals, it will line to the next interlocking to the east
- The signals in the yellow boxes are the westbound signals, work the same as the eastbound signals but instead for movement west.

What is in the red boxes are the switches in the interlocking, clicking on these will change the state of the switch, either if it is reversed or normal, the drawing will change on the screen depending on what state the switch is at.

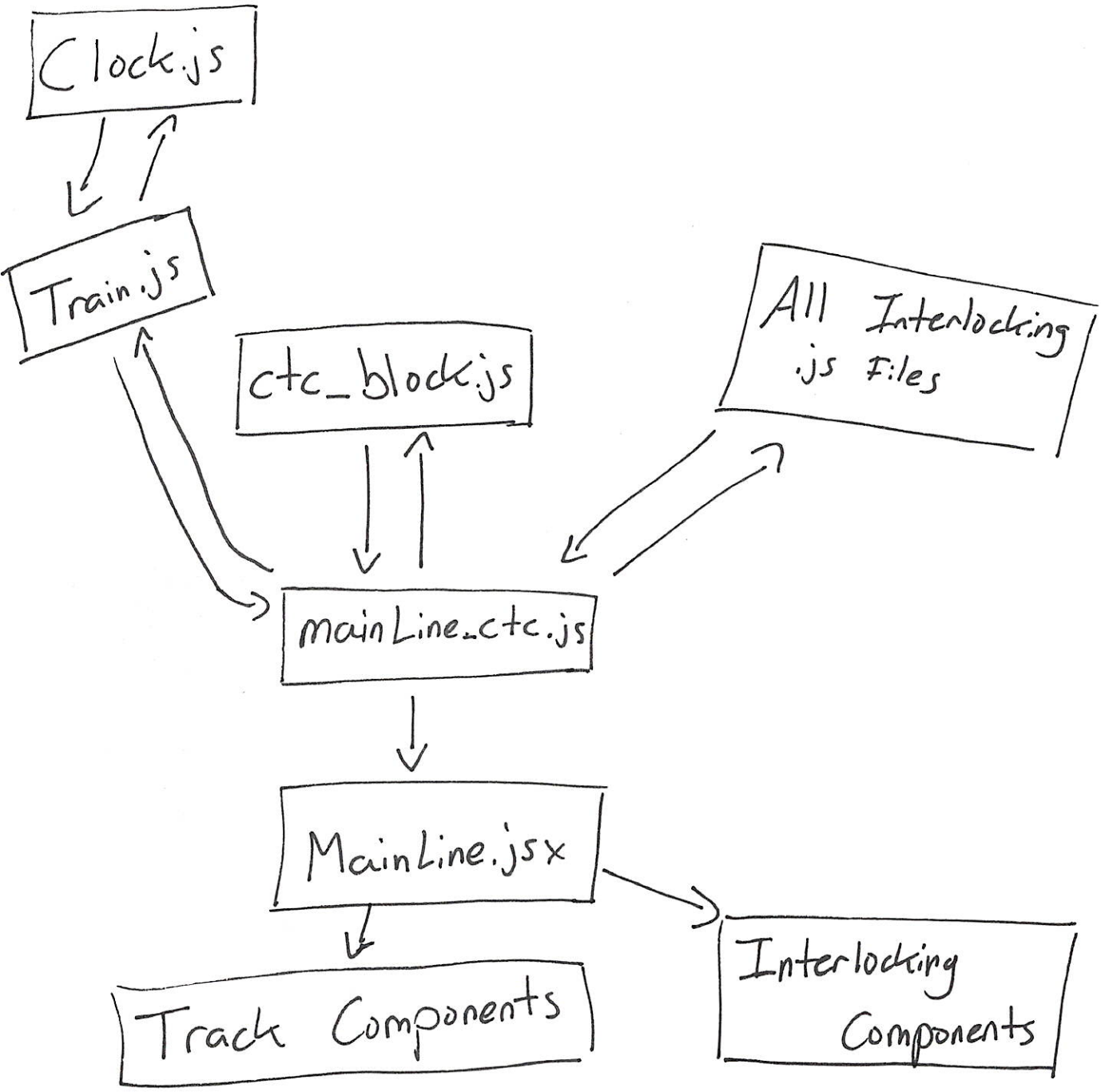
Above is what a reversed switch looks like, it is clear to see that it actually draws which way the train will move when it goes over this section of track.

When the signal is clicked, if there are conflicts with the route you have set up, you will get an alert message. If there are no conflicts then the track will turn green, to denote that there is a route setup between the two interlockings. Also if the track is Red that means that block or piece of track is currently occupied by a train, and the red text above the occupied track is the symbol of that train.

Part IV: Design

For this project, I followed the standard design which is called a “Model and View”, meaning that there are two major parts to the program. The Model holds all the information and is actually running is what the program is doing. And the View is what draws what is going on in the program for the user to see, the View literally lets you view the information. In this design standard, the view cannot change anything in the Model, it can only reference the data from the model. Do build the model, I created a Javascript class for each interlocking, and a main controller class. For the view part is a ReactJS component for each interlocking and tracks on the panel. Using the properties feacture of ReactJS components allows us to pass information to the component, but doesn’t allow it to pass data backwards, instead you have to pass a function to the component, and this function is from the controller class, so when the user clicks something, the corresponding function in the Model class is called.

DATA FLOW



The JSX component only recieve data they donot creat any data themselfs

Part V: File Structure

/Project
- /dist
- /doc
- /node_modules
- /public
  - /images
- /src
  - /components
    - /Panel
      - /Bergen_County_Line
      - /Main_Line
      - /Southern_Tier_Line
  - /css
    - /Bergen_County_Line
    - /Main_Line
    - /Southern_Tier_Line
  - /other
  - /scripts
    - /CTC
    - /Interlockings
      - /Bergen_Line
      - /Main_Line
      - /Southern_Tier_Line
    - /Trains

/Project – directory is what holds the entire project, all of the resource images, and code

/dist – Holds the main HTML file, which actually doesn't have any information in it other than reference to the ReactJS app bundle

/doc – Holds the documentation for the application, where you found this file

/node_modules – This is created by the ReactJS application which is NOT written by me

/public – Holds resource files in this case all it holds is a directory holding all the /images that might be used by the application

/src – Holds all the source code for the application

/src/components – Holds all the JSX files which are the ReactJS components that make up the panel on the screen, they are divided into the sections of the railroad

/src/css – The style sheets for the ReactJS components, these are also divided up by the sections of the railroad they're in

/src/other – Was used by me to hold the work log while coding this application

/src/scripts – Holds all the Javascript "model" files which control the train movements

/src/scripts/CTC – The ctc controller files to control the entire railroad

/src/scripts/Interlockings – All the model classes for each of the interlocking components

/src/scripts/Trains – the classes for the trains that move across the railroad.

Part VI: Class Descriptions

SEE NEXT PAGE

Class: MainLine_CTC

MainLine_CTC()

new MainLine_CTC()

The constructor for the Clock class

constructor()

Source: mainLine_ctc.js, line 84

Methods

add_train()

Takes in a new train and adds it to the train_list array

add_train()

Source: mainLine_ctc.js, line 600

get_bc()

Gets reference to the CP BC Interlocking

get_bc()

Source: mainLine_ctc.js, line 393

Returns:

Reference to the CP BC Interlocking

get_bergen_blocks_status()

Gets the status of all the blocks on the Southern Tier Section

get_bergen_blocks_status()

Source: mainLine_ctc.js, line 772

Returns:

An object with the status of each block

get_bergen_symbols()

Gets all the symbols for the blocks on the Bergen County Line Section

get_bergen_symbols()

Source: mainLine_ctc.js, line 850

Returns:

An obnject with all the block symbols on the Bergen Line

get_block_by_name(name,)

takes in the name of a block, and returns the reference to that specific block

get_block_by_name()

Parameters:

Name	Type	Description
name,		the name of the block to find

Source: mainLine_ctc.js, line 1191

Returns:

reference to the block

get_bt()

Gets reference to the BT Interlocking

get_bt()

Source: mainLine_ctc.js, line 569

Returns:

Reference to the BT Interlocking

get_hall()

Gets reference to the CP Hall Interlocking

get_hall()

Source: mainLine_ctc.js, line 426

Returns:

Reference to the CP Hall Interlocking

get_harriman()

Gets reference to the CP Harriman Interlocking

get_harriman()

Source: mainLine_ctc.js, line 459

Returns:

Reference to the CP Harriman Interlocking

get_hilburn()

Gets reference to the Hilburn Interlocking

get_hilburn()

Source: mainLine_ctc.js, line 481

Returns:

Reference to the Hilburn Interlocking

get_howells()

Gets reference to the CP Howells Interlocking

get_howells()

Source: mainLine_ctc.js, line 415

Returns:

Reference to the CP Howells Interlocking

get_hudson()

Gets reference to the CP Hudson Junction Interlocking

get_hudson()

Source: mainLine_ctc.js, line 437

Returns:

Reference to the CP Hudson Junction Interlocking

get_hx()

Gets reference to the HX Interlocking

get_hx()

Source: mainLine_ctc.js, line 591

Returns:

Reference to the HX Interlocking

get_interlocking_route(key,, direction,)

Takes where a train currently is and gets it's next route

get_interlocking_route()

Parameters:

Name	Type	Description
key,		Is ueed to find the trains curent interlocking
direction,		which way the train is traveling

Source: mainLine_ctc.js, line 952

get_laurel()

Gets reference to the Laurel Interlocking

get_laurel()

Source: mainLine_ctc.js, line 558

Returns:

Reference to the Laurel Interlocking

get_mainLine_blocks_status()

Gets the status of all the bloccks on the Southern Tier Section

get_mainLine_blocks_status()

Source: mainLine_ctc.js, line 725

Returns:

An object with the status of each block

get_mainLine_symbols()

Gets all the symbols for the blocks on the Main Line Section

get_mainLine_symbol()

> Source: mainLine_ctc.js, line 875

Returns:

An object with all the block symbols on the Main Line Section

get_mill()

Gets reference to the Mill Interlocking

get_mill()

> Source: mainLine_ctc.js, line 536

Returns:

Reference to the Mill Interlocking

get_ov()

Gets reference to the CP OV Interlocking

get_ov()

> Source: mainLine_ctc.js, line 404

Returns:

Reference to the CP OV Interlocking

get_pa()

Gets reference to the CP PA Interlocking

get_pa()

> Source: mainLine_ctc.js, line 371

Returns:

Reference to the CP PA Interlocking

get_pascack()

Gets reference to the Pascack Interlocking

get_pascack()

> Source: mainLine_ctc.js, line 580

Returns:

Reference to the Pascack Interlocking

get_port()

Gets reference to the CP Port Interlocking

get_port()

Source: mainLine_ctc.js, line 382

Returns:

Reference to the CP Port Interlocking

get_ridgewood()

Gets reference to the Ridgewood Junction Interlocking

get_ridgewood()

Source: mainLine_ctc.js, line 514

Returns:

Reference to the Ridgewood Junction Interlocking

get_sf()

Gets reference to the SF Interlocking

get_sf()

Source: mainLine_ctc.js, line 492

Returns:

Reference to the SF Interlocking

get_sparrow()

Gets reference to the CP Sparrow Interlocking

get_sparrow()

Source: mainLine_ctc.js, line 360

Returns:

Reference to the CP Sparrow Interlocking

get_sterling()

Gets reference to the CP Sterling Interlocking

get_sterling()

Source: mainLine_ctc.js, line 470

Returns:

Reference to the CP Sterling Interlocking

get_suscon()

Gets reference to the Suscon Interlocking

get_suscon()

Source: mainLine_ctc.js, line 525

Returns:

Reference to the Suscon Interlocking

get_tier_block_status()

get_tier_block_status()

Source: mainLine_ctc.js, line 801

Returns:

An object with the status of each block

get_tier_symbols()

Gets all the symbols for the blocks on the Southern Tier Section

get_tier_symbols()

Source: mainLine_ctc.js, line 914

Returns:

An object with all the block symbols on the Southern Tier Section

get_valley()

Gets reference to the CP Central Valley Interlocking

get_valley()

Source: mainLine_ctc.js, line 448

Returns:

Reference to the CP Central Valley Interlocking

get_wc()

Gets reference to the WC Interlocking

get_wc()

Source: mainLine_ctc.js, line 503

Returns:

Reference to the WC Interlocking

get_westSecaucus()

Gets reference to the West Secaucus Interlocking

get_westSecaucus()

Source: mainLine_ctc.js, line 547

Returns:

Reference to the West Secaucus Interlocking

occupy_blocks()

goes through all the trains and finds their current location and occupys the correct block

occupy_blocks()

Source: mainLine_ctc.js, line 610

reset_route_mainLine_blocks()

Resets all the blocks that are routed

reset_route_mainLine_blocks()

Source: mainLine_ctc.js, line 629

set_occupy_interlocking(track,, name,)

Takes in what interlocking and the track number, and set that the specific interlocking is occupied on the last track

set_occupy_interlocking

Parameters:

Name	Type	Description
track,		the track number in the interlocking to occupy, for some interlocking with only one route doesn't need the track
name,		the name of the interlocking to occupy

Source: mainLine_ctc.js, line 1048

update_interlockings()

Goes through to see if each interlocking can have a train clear if it's occupied

update_interlockings()

Source: mainLine_ctc.js, line 323

update_route_blocks()

Gets all the routes from each interlocking and sets the accoriding blocks

update_route_blocks()

Source: mainLine_ctc.js, line 210

update_trains()

Goes through all the trains in the list and updates their location if they're capable of doing so

updates_trains()

Source: mainLine_ctc.js, line 261

Class: CTC_Block

CTC_Block(p_name,, p_size,, p_status,)

new CTC_Block(p_name,, p_size,, p_status,)

The Constructor of the CTC_Block Class

Sets all the memeber variables to their initial values, when the application starts

Parameters:

Name	Type	Description
p_name,		The Name of the Block
p_size,		The Size of the Block
p_status,		Current Status. Only Used for debugging when build the applications

Source: ctc_block.js, line 36

Methods

get_block_status()

Getter for the block_status member variable

get_block_status()

Source: ctc_block.js, line 50

Returns:

The current status of the block

get_size()

Getter for the block_size member variable

get_size()

Source: ctc_block.js, line 61

Returns:

The size of the block

get_symbol()

Getter for the train_symbol memebr variable

get_symbol()

Source: ctc_block.js, line 72

Returns:

The symbol of the trail that is currently in the block

reset_block()

Resets the Block status to Empty

This is used to reset the block, when the CTC controller refreshes the train and route locations

Source: ctc_block.js, line 83

set_block_status(p_status,)

Sets the block current status based off of what tag is passed in

set_block_status()

Parameters:

Name	Type	Description
p_status,		A String which is the Kinda of status of what to set the block too

Source: ctc_block.js, line 108

set_symbol(n_symbol,)

Setter for the train_symbol member variable

set_symbol()

Parameters:

Name	Type	Description
n_symbol,		The new symbols to set the member variable too

Source: ctc_block.js, line 97

Class: Train

Train(p_symbol, p_location, p_direction, p_block_size)

CLASS Train

Constructor

new Train(p_symbol, p_location, p_direction, p_block_size)

constructor()

Parameters:

Name	Type	Description
p_symbol		> The Train's Symbol
p_location		> The Trains Inital Location
p_direction		> The Direction the train is traveling
p_block_size		> The size of the trains inital block

Source: train.js, line 33

Methods

can_update_location()

can_update_location()

Source: train.js, line 72

get_block_size()

get_block_size()

Source: train.js, line 99

get_direction()

get_direction()

Source: train.js, line 118

get_location()

get_location()

Source: train.js, line 90

get_route()

get_route()

Source: train.js, line 127

get_symbol()

get_symbol()

Source: train.js, line 53

Returns:

The train symbol

set_block_size(n_size,)

set_block_size()

Parameters:

Name	Type	Description
n_size,		the new size of the next block

Source: train.js, line 109

set_route(n_route,)

set_route()

Parameters:

Name	Type	Description
n_route,		the trains new route

Source: train.js, line 137

update_location()

update_location()

Source: train.js, line 62

Class: Clock

Home

Classes

Clock

Clock()

CLASS Clock

Constructor

new Clock()

constructor()

Source: clock.js, line 26

Members

getTimeFromStart

getTimeFromStart()

Source: clock.js, line 46

Methods

startClock()

startClock()

Source: clock.js, line 35

Class: CTC_Hilburn

CTC_Hilburn()

`new CTC_Hilburn()`

The constructor for the CTC_Hilburn class

This will initialize all the member variables when the program is started

Source: ctc_hilburn.js, line 40

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_hilburn.js, line 206

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hilburn.js, line 149

click_sig_2w_1(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_hilburn.js, line 88

click_sig_2w_2(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

> Source: ctc_hilburn.js, line 118

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

> Source: ctc_hilburn.js, line 265

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

> Source: ctc_hilburn.js, line 246

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

> Source: ctc_hilburn.js, line 64

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hilburn.js, line 188

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hilburn.js, line 230

Class: CTC_Laurel

CTC_Laurel()

new CTC_Laurel()

The constructor for the CTC_Laurel class

This will initialize all the member variables when the program is started

Source: ctc_laurel.js, line 66

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_laurel.js, line 785

click_sig_2w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_laurel.js, line 158

click_sig_4e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Name	Type	Description
next_block_4,		The next block on Track #4

Source: ctc_laurel.js, line 599

click_sig_4w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_laurel.js, line 226

click_sig_6e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_laurel.js, line 436

click_sig_8e(next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_4,		The next block on Track #4

Source: ctc_laurel.js, line 679

click_sig_8w(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

> Source: ctc_laurel.js, line 293

click_sig_10w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

> Source: ctc_laurel.js, line 372

click_sig_12e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

> Source: ctc_laurel.js, line 516

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_laurel.js, line 962

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_laurel.js, line 845

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_laurel.js, line 115

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 709

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 728

set_trk_3_occupied(n_state,)

Sets track #3 as occupied

set_trk_3_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 747

set_trk_4_occupied(n_state,)

Sets track #4 as occupied

set_trk_4_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 766

throw_sw_1()

Function to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 863

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 879

throw_sw_7()

Funtion to throw switch #7 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 895

throw_sw_9()

Funtion to throw switch #9 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 911

throw_sw_11()

Funtion to throw switch #11 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 927

throw_sw_13()

Funtion to throw switch #13 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 943

Documentation generated by JSDoc 3.6.3 on Wed Sep 25 2019 00:09:45 GMT-0400 (Eastern Daylight Time)

Class: CTC_Mill

CTC_Mill()

new CTC_Mill()

The constructor for the CTC_Mill class

This will initialize all the member variables when the program is started

Source: ctc_mill.js, line 49

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_mill.js, line 284

click_sig(sigNum,, next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
sigNum,		The number of the signal clicked
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_mill.js, line 85

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_mill.js, line 393

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_mill.js, line 323

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_mill.js, line 336

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_mill.js, line 246

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_mill.js, line 265

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_mill.js, line 360

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

> Source: ctc_mill.js, line 374

Class: CTC_Ridgewood

CTC_Ridgewood()

new CTC_Ridgewood()

The constructor for the CTC_Ridgewood class

This will initialize all the member variables when the program is started

Source: ctc_ridgewood.js, line 59

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_ridgewood.js, line 751

click_sig_2e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_ridgewood.js, line 421

click_sig_2w1(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Name	Type	Description
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 142

click_sig_2w2(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 211

click_sig_4e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_ridgewood.js, line 508

click_sig_4w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 282

click_sig_6e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_ridgewood.js, line 595

click_sig_6w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 351

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_ridgewood.js, line 884

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_ridgewood.js, line 676

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_ridgewood.js, line 102

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ridgewood.js, line 694

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ridgewood.js, line 713

set_trk_3_occupied(n_state,)

Sets track #3 as occupied

set_trk_3_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ridgewood.js, line 732

throw_sw_1()

Function to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 801

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 817

throw_sw_5()

Funtion to throw switch #5 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 833

throw_sw_7()

Funtion to throw switch #7 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 849

throw_sw_9()

Funtion to throw switch #9 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 865

Class: CTC_SF

CTC_SF()

new CTC_SF()

The constructor for the CTC_SF class

This will initialize all the member variables when the program is started

Source: ctc_sf.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_sf.js, line 393

click_sig_2e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sf.js, line 227

click_sig_2w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_sf.js, line 117

click_sig_4e_1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_sf.js, line 260

click_sig_4e_2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_sf.js, line 309

click_sig_4w(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_sf.js, line 179

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_sf.js, line 483

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_sf.js, line 464

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_sf.js, line 83

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sf.js, line 355

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sf.js, line 374

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sf.js, line 432

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sf.js, line 448

Class: CTC_Suscon

CTC_Suscon()

new CTC_Suscon()

The constructor for the CTC_Suscon class

This will initialize all the member variables when the program is started

Source: ctc_suscon.js, line 48

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_suscon.js, line 282

click_sig(sigNum,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
sigNum,		The signal number that was clicked
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_suscon.js, line 83

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_suscon.js, line 393

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_suscon.js, line 320

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_suscon.js, line 336

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_suscon.js, line 244

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_suscon.js, line 263

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_suscon.js, line 360

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_suscon.js, line 374

Class: CTC_WC

CTC_WC()

new CTC_WC()

The constructor for the CTC_WC class

This will initialize all the member variables when the program is started

Source: ctc_wc.js, line 57

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_wc.js, line 561

click_sig_2e_1(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 329

click_sig_2e_2(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 395

click_sig_2w_1(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 131

click_sig_2w_2(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 197

click_sig_4e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 461

click_sig_4w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

> Source: ctc_wc.js, line 263

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

> Source: ctc_wc.js, line 684

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

> Source: ctc_wc.js, line 664

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

> Source: ctc_wc.js, line 94

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_wc.js, line 523

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_wc.js, line 542

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 600

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 616

throw_sw_5()

Funtion to throw switch #5 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 632

throw_sw_7()

Funtion to throw switch #7 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 648

Class: CTC_WestSecaucus

CTC_WestSecaucus()

new CTC_WestSecaucus()

The constructor for the CTC_WestSecaucus class

This will initialize all the member variables when the program is started

Source: ctc_westSecaucus.js, line 43

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_westSecaucus.js, line 242

click_sig(sigNum,, next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
sigNum,		the id of the signal clicked
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_westSecaucus.js, line 74

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_westSecaucus.js, line 343

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_westSecaucus.js, line 266

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_westSecaucus.js, line 282

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_westSecaucus.js, line 224

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_westSecaucus.js, line 308

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_westSecaucus.js, line 324

Class: CTC_BT

CTC_BT()

new CTC_BT()

The constructor for the CTC_BT class

This will initialize all the member variables when the program is started

Source: ctc_bt.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_bt.js, line 442

click_sig_2e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_bt.js, line 268

click_sig_2w1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bt.js, line 117

click_sig_2w2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bt.js, line 167

click_sig_4e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_bt.js, line 335

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bt.js, line 217

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_bt.js, line 544

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_bt.js, line 525

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_bt.js, line 84

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_bt.js, line 398

set_trk_2_occupied(n_state,)

Sets track #1 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_bt.js, line 420

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_bt.js, line 481

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_bt.js, line 495

throw_sw_5()

Changes the current state of switch #5, used when user clicks the switch

throw_sw_5()

Source: ctc_bt.js, line 509

Documentation generated by JSDoc 3.6.3 on Wed Sep 25 2019 00:07:20 GMT-0400 (Eastern Daylight Time)

Class: CTC_HX

CTC_HX()

new CTC_HX()

The constructor for the CTC_BT class

constructor()

Source: ctc_hx.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

can_clear()

Source: ctc_hx.js, line 475

click_sig_2e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

click_sig_2e()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_hx.js, line 295

click_sig_2w1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_2w1()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 114

click_sig_2w2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_2w2()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 163

click_sig_2w3(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_2w3()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 212

click_sig_4e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

click_sig_4e()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_hx.js, line 362

click_sig_4w(next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_4w()

Parameters:

Name	Type	Description
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 260

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

get_interlocking_status()

> Source: ctc_hx.js, line 574

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

> Source: ctc_hx.js, line 555

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

> Source: ctc_hx.js, line 84

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

> Source: ctc_hx.js, line 437

set_trk_2_occupied(n_state,)

Sets track #1 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hx.js, line 456

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_hx.js, line 511

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_hx.js, line 525

throw_sw_5()

Changes the current state of switch #5, used when user clicks the switch

throw_sw_5()

Source: ctc_hx.js, line 539

Class: CTC_Pascack

CTC_Pascack()

new CTC_Pascack()

The constructor for the CTC_BT class

This will initialize all the member variables when the program is started

Source: ctc_pascack.js, line 47

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_pascack.js, line 339

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 206

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 108

click_sig_4e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 255

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 157

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_pascack.js, line 424

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_pascack.js, line 405

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_pascack.js, line 78

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pascack.js, line 301

set_trk_2_occupied(n_state,)

Sets track #1 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pascack.js, line 320

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_pascack.js, line 375

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_pascack.js, line 389

Class: CTC_BC

CTC_BC()

new CTC_BC()

The constructor for the CTC_BC class

This will initialize all the member variables when the program is started

Source: ctc_bc.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_bc.js, line 207

click_sig_2e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_bc.js, line 131

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bc.js, line 90

click_sig_4e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_bc.js, line 161

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_bc.js, line 267

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_bc.js, line 248

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_bc.js, line 65

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_bc.js, line 189

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_bc.js, line 232

Class: CTC_Hall

CTC_Hall()

new CTC_Hall()

The constructor for the CTC_Hall class

This will initialize all the member variables when the program is started

Source: ctc_hall.js, line 47

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_hall.js, line 299

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hall.js, line 186

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hall.js, line 107

click_sig_4e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_hall.js, line 231

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hall.js, line 140

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_hall.js, line 371

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_hall.js, line 352

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving

Name	Type	Description
track,		The Track number of the train

Source: ctc_hall.js, line 77

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hall.js, line 261

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hall.js, line 280

throw_sw_1()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hall.js, line 336

Class: CTC_Harriman

CTC_Harriman()

`new CTC_Harriman()`

The constructor for the CTC_Harriman class

This will initialize all the member variables when the program is started

Source: ctc_harriman.js, line 44

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_harriman.js, line 261

click_sig_1e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_harriman.js, line 155

click_sig_1w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_harriman.js, line 100

click_sig_2e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_harriman.js, line 185

click_sig_3e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_harriman.js, line 215

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_harriman.js, line 338

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_harriman.js, line 319

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving

Name	Type	Description
track,		The Track number of the train

Source: ctc_harriman.js, line 71

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_harriman.js, line 243

throw_sw_21()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_harriman.js, line 287

throw_sw_32()

Funtion to throw switch #32 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_harriman.js, line 303

Class: CTC_Howells

CTC_Howells()

`new CTC_Howells()`

The constructor for the CTC_Howells class

This will initialize all the member variables when the program is started

Source: ctc_howells.js, line 39

Methods

`can_clear()`

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_howells.js, line 204

`click_sig_2e(next_block_1,)`

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_howells.js, line 128

`click_sig_2es(next_block_1,)`

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_howells.js, line 158

`click_sig_2w(next_block_1,, next_block_2,)`

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so

what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_howells.js, line 87

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_howells.js, line 264

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_howells.js, line 245

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_howells.js, line 62

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_howells.js, line 186

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_howells.js, line 229

Class: CTC_Hudson

CTC_Hudson()

new CTC_Hudson()

The constructor for the CTC_Hudson class

This will initialize all the member variables when the program is started

Source: ctc_hudson.js, line 43

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_hudson.js, line 295

click_sig_2e(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_hudson.js, line 190

click_sig_2es(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_hudson.js, line 235

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hudson.js, line 100

click_sig_2ws(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hudson.js, line 145

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_hudson.js, line 383

Returns:

Object with the status of the interlocking

get_occupied()

get_occupied()

Source: ctc_hudson.js, line 321

Returns:

If the interlocking is occupied or not

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_hudson.js, line 364

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_hudson.js, line 70

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hudson.js, line 277

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hudson.js, line 332

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hudson.js, line 348

Class: CTC_OV

CTC_OV()

new CTC_OV()

The constructor for the CTC_OV class

This will initialize all the member variables when the program is started

Source: ctc_ov.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_ov.js, line 207

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_ov.js, line 150

click_sig_2w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_ov.js, line 89

click_sig_2ws(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_ov.js, line 119

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_ov.js, line 267

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_ov.js, line 248

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_ov.js, line 65

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ov.js, line 189

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_1()

Source: ctc_ov.js, line 232

Class: CTC_PA

CTC_PA()

new CTC_PA()

The constructor for the CTC_PA class

This will initialize all the member variables when the program is started

Source: ctc_pa.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_pa.js, line 398

click_sig_2e(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_pa.js, line 251

click_sig_2w_1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pa.js, line 120

click_sig_2w_2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pa.js, line 169

click_sig_4e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_pa.js, line 301

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pa.js, line 218

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_pa.js, line 505

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_pa.js, line 486

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_pa.js, line 81

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pa.js, line 360

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pa.js, line 379

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_1()

Source: ctc_pa.js, line 436

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_3()

Source: ctc_pa.js, line 453

throw_sw_5()

Funtion to throw switch #5 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_5()

Source: ctc_pa.js, line 470

Class: CTC_Port

CTC_Port()

new CTC_Port()

The constructor for the CTC_Port class

This will initialize all the member variables when the program is started

Source: ctc_port.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_port.js, line 207

click_sig_2e_1(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_port.js, line 131

click_sig_2e_2(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_port.js, line 161

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so

what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_port.js, line 90

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_port.js, line 266

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_port.js, line 247

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_port.js, line 65

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_port.js, line 189

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_port.js, line 231

Class: CTC_Sparrow

CTC_Sparrow()

new CTC_Sparrow()

The constructor for the CTC_Sparrow class

This will initialize all the member variables when the program is started

Source: ctc_sparrow.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_sparrow.js, line 258

click_sig_2e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_sparrow.js, line 187

click_sig_2w_1(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sparrow.js, line 95

click_sig_2w_2(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sparrow.js, line 125

click_sig_2w_3(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sparrow.js, line 155

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_sparrow.js, line 335

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_sparrow.js, line 316

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving

Name	Type	Description
track,		The Track number of the train

Source: ctc_sparrow.js, line 68

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sparrow.js, line 240

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sparrow.js, line 284

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sparrow.js, line 300

Class: CTC_Sterling

CTC_Sterling()

new CTC_Sterling()

The constructor for the CTC_Sterling class

This will initialize all the member variables when the program is started

Source: ctc_sterling.js, line 40

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_sterling.js, line 206

click_sig_1e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_sterling.js, line 149

click_sig_2w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sterling.js, line 88

click_sig_2ws(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sterling.js, line 118

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_sterling.js, line 265

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_sterling.js, line 246

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_sterling.js, line 64

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sterling.js, line 188

throw_sw_21()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sterling.js, line 230

Class: CTC_Valley

CTC_Valley()

new CTC_Valley()

The constructor for the CTC_Valley class

This will initialize all the member variables when the program is started

Source: ctc_valley.js, line 40

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_valley.js, line 206

click_sig_1e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_valley.js, line 149

click_sig_1w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_valley.js, line 88

click_sig_2w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_valley.js, line 118

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_valley.js, line 274

Returns:

Object with the status of the interlocking

get_occupied()

Getter for the int_occupied

get_occupied()

Source: ctc_valley.js, line 228

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_valley.js, line 255

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_valley.js, line 64

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_valley.js, line 188

throw_sw_21()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_valley.js, line 239

Class: MainLine

MainLine(props,)

The React JSX Component Class for the entire Maine Line Dispatcher Panel This class is a JSX React Component for the Maine Line Dispatch Panel, this will control all the other components that make up the pannel. This also controls the functions that allow each component to change their respected back end functions.

Constructor

new MainLine(props,)

The Constructor for the MainLine JSX class. All this does is set that state for every thing getting the information fro the CTC controller, the state here is used to send to the child components so they can render the correct information

constructor()

Parameters:

Name	Type	Description
props,		Required as park of ReactJS, but is not used here

Source: MainLine.jsx, line 90

Members

bc_click_sig_2e

The event handler for Signal #2e

bc_click_sig_2e()

Source: MainLine.jsx, line 1148

bc_click_sig_2w

The event handler for Signal #2w

bc_click_sig_2w()

Source: MainLine.jsx, line 1132

bc_click_sig_4e

The event handler for Signal #4e

bc_click_sig_4e()

Source: MainLine.jsx, line 1163

bc_throw_sw_1

The event handler for switch #1

bc_throw_sw_1()

> Source: MainLine.jsx, line 1178

bt_click_sig_2e

Event handler for the signal #2e

bt_click_sig_2e()

> Source: MainLine.jsx, line 795

bt_click_sig_2w1

Event handler for the signal #2w1

bt_click_sig_2w1()

> Source: MainLine.jsx, line 747

bt_click_sig_2w2

Event handler for the signal #2w2

bt_click_sig_2w2()

> Source: MainLine.jsx, line 763

bt_click_sig_4e

Event handler for the signal #4e

bt_click_sig_4e()

> Source: MainLine.jsx, line 812

bt_click_sig_4w

Event handler for the signal #4

bt_click_sig_4w()

> Source: MainLine.jsx, line 779

bt_throw_sw_1

The event handler for switch #1

bt_throw_sw_1()

> Source: MainLine.jsx, line 829

bt_throw_sw_3

The event handler for switch #3

bt_throw_sw_3()

> Source: MainLine.jsx, line 841

bt_throw_sw_5

The event handler for switch #5

bt_throw_sw_5()

> Source: MainLine.jsx, line 853

hall_click_sig_2e

The event handler for Signal #2e

hall_click_sig_2e()

> Source: MainLine.jsx, line 1346

hall_click_sig_2w

The event handler for Signal #2w

hall_click_sig_2w()

> Source: MainLine.jsx, line 1315

hall_click_sig_4e

The event handler for Signal #4e

hall_click_sig_4e()

> Source: MainLine.jsx, line 1362

hall_click_sig_4w

The event handler for Signal #4w

hall_click_sig_4w()

> Source: MainLine.jsx, line 1330

hall_throw_sw_1

The event handler for switch #1

hall_throw_sw_1()

> Source: MainLine.jsx, line 1377

harriman_click_sig_1e

The event handler for Signal #1e

harriman_click_sig_1e()

> Source: MainLine.jsx, line 1561

harriman_click_sig_1w

The event handler for Signal #1w

harriman_click_sig_1w()

> Source: MainLine.jsx, line 1544

harriman_click_sig_2e

The event handler for Signal #2e

harriman_click_sig_2e()

> Source: MainLine.jsx, line 1576

harriman_click_sig_3e

The event handler for Signal #3e

harriman_click_sig_3e()

> Source: MainLine.jsx, line 1591

harriman_throw_sw_21

The event handler for switch #21

harriman_throw_sw_21()

> Source: MainLine.jsx, line 1606

harriman_throw_sw_32

The event handler for switch #32

harriman_throw_sw_32()

> Source: MainLine.jsx, line 1618

hilburn_click_sig_2e

The event handler for Signal #2e

hilburn_click_sig_2e()

> Source: MainLine.jsx, line 1728

hilburn_click_sig_2w_1

The event handler for Signal #2w_1

hilburn_click_sig_2w_1()

> Source: MainLine.jsx, line 1698

hilburn_click_sig_2w_2

The event handler for Signal #2w_2

hilburn_click_sig_2w_2()

> Source: MainLine.jsx, line 1713

hilburn_throw_sw_1

The event handler for switch #1

hilburn_throw_sw_1()

> Source: MainLine.jsx, line 1744

howells_click_sig_2e

The event handler for Signal #2e

howells_click_sig_2e()

> Source: MainLine.jsx, line 1270

howells_click_sig_2es

The event handler for Signal #2es

howells_click_sig_2es()

> Source: MainLine.jsx, line 1285

howells_click_sig_2w

The event handler for Signal #2w

howells_click_sig_2w()

> Source: MainLine.jsx, line 1254

howells_throw_sw_3

The event handler for switch #3

howells_throw_sw_3()

> Source: MainLine.jsx, line 1300

hudson_click_sig_2e

The event handler for Signal #2e

hudson_click_sig_2e()

> Source: MainLine.jsx, line 1424

hudson_click_sig_2es

The event handler for Signal #2es

hudson_click_sig_2es()

> Source: MainLine.jsx, line 1440

hudson_click_sig_2w

The event handler for Signal #2w

hudson_click_sig_2w()

> Source: MainLine.jsx, line 1392

hudson_click_sig_2ws

The event handler for Signal #2ws

hudson_click_sig_2ws()

> Source: MainLine.jsx, line 1408

hudson_throw_sw_1

The event handler for switch #1

hudson_throw_sw_1()

> Source: MainLine.jsx, line 1456

hudson_throw_sw_3

The event handler for switch #3

hudson_throw_sw_3()

> Source: MainLine.jsx, line 1468

hx_click_sig_2e

The event handler for the Signal 2e

hx_click_sig_2e()

> Source: MainLine.jsx, line 582

hx_click_sig_2w1

The event handler for Signal #2w-1

hx_click_sig_2w1()

> Source: MainLine.jsx, line 519

hx_click_sig_2w2

The event handler for the Signal #2w2

hx_click_sig_2w2()

> Source: MainLine.jsx, line 535

hx_click_sig_2w3

The event handler for the Signal #2w3

hx_click_sig_2w3()

> Source: MainLine.jsx, line 551

hx_click_sig_4e

The event handler for the Signal 4e

hx_click_sig_4e()

> Source: MainLine.jsx, line 599

hx_click_sig_4w

The event handler for the Signal #4w

hx_click_sig_4w()

> Source: MainLine.jsx, line 567

hx_throw_sw_1

The event handler for switch #1

hx_throw_sw_1()

> Source: MainLine.jsx, line 617

hx_throw_sw_3

The event handler for switch #3

hx_throw_sw_3()

> Source: MainLine.jsx, line 629

hx_throw_sw_5

The event handler for switch #5

hx_throw_sw_5()

> Source: MainLine.jsx, line 641

laurel_click_sig_2w

The event handler for Signal #2w

laurel_click_sig_2w()

> Source: MainLine.jsx, line 2489

laurel_click_sig_4e

The event handler for Signal #4e

laurel_click_sig_4e()

> Source: MainLine.jsx, line 2594

laurel_click_sig_4w

The event handler for Signal #4w

laurel_click_sig_4w()

> Source: MainLine.jsx, line 2506

laurel_click_sig_6e

The event handler for Signal #6e

laurel_click_sig_6e()

> Source: MainLine.jsx, line 2558

laurel_click_sig_8e

The event handler for Signal #8e

laurel_click_sig_8e()

> Source: MainLine.jsx, line 2612

laurel_click_sig_8w

The event handler for Signal #8w

laurel_click_sig_8w()

> Source: MainLine.jsx, line 2523

laurel_click_sig_10w

The event handler for Signal #10w

laurel_click_sig_10w()

> Source: MainLine.jsx, line 2541

laurel_click_sig_12e

The event handler for Signal #12e

laurel_click_sig_12e()

> Source: MainLine.jsx, line 2576

laurel_throw_sw_1

The event handler for switch #1

laurel_throw_sw_1()

> Source: MainLine.jsx, line 2627

laurel_throw_sw_3

The event handler for switch #3

laurel_throw_sw_3()

> Source: MainLine.jsx, line 2639

laurel_throw_sw_7

The event handler for switch #7

laurel_throw_sw_7()

> Source: MainLine.jsx, line 2651

laurel_throw_sw_11

The event handler for switch #11

laurel_throw_sw_11()

> Source: MainLine.jsx, line 2663

laurel_throw_sw_13

The event handler for switch #13

laurel_throw_sw_13()

> Source: MainLine.jsx, line 2675

mill_click_sig_2e

The event handler for Signal #2e

mill_click_sig_2e()

> Source: MainLine.jsx, line 2316

mill_click_sig_2w

The event handler for Signal #2w

mill_click_sig_2w()

> Source: MainLine.jsx, line 2299

mill_click_sig_4e

The event handler for Signal #4e

mill_click_sig_4e()

> Source: MainLine.jsx, line 2350

mill_click_sig_4w

The event handler for Signal #4w

mill_click_sig_4w()

> Source: MainLine.jsx, line 2333

mill_throw_sw_1

The event handler for switch #1

mill_throw_sw_1()

> Source: MainLine.jsx, line 2367

mill_throw_sw_3

The event handler for switch #3

mill_throw_sw_3()

> Source: MainLine.jsx, line 2379

ov_click_sig_2e

The event handler for Signal #2e

ov_click_sig_2e()

> Source: MainLine.jsx, line 1223

ov_click_sig_2w

The event handler for Signal #2w

ov_click_sig_2w()

> Source: MainLine.jsx, line 1193

ov_click_sig_2ws

The event handler for Signal #2ws

ov_click_sig_2ws()

> Source: MainLine.jsx, line 1208

ov_throw_sw_1

The event handler for switch #1

ov_throw_sw_1()

> Source: MainLine.jsx, line 1239

pa_click_sig_2e

The event handler for Signal #2e

pa_click_sig_2e()

> Source: MainLine.jsx, line 1011

pa_click_sig_2w_1

The event handler for Signal #2w_1

pa_click_sig_2w_1()

> Source: MainLine.jsx, line 961

pa_click_sig_2w_2

The event handler for Signal #2w_2

pa_click_sig_2w_2()

> Source: MainLine.jsx, line 978

pa_click_sig_4e

The event handler for Signal #4e

pa_click_sig_4e()

> Source: MainLine.jsx, line 1027

pa_click_sig_4w

The event handler for Signal #4w

pa_click_sig_4w()

> Source: MainLine.jsx, line 995

pa_throw_sw_1

The event handler for switch #1

pa_throw_sw_1()

> Source: MainLine.jsx, line 1044

pa_throw_sw_3

The event handler for switch #3

pa_throw_sw_3()

> Source: MainLine.jsx, line 1056

pascack_click_sig_2e

Event handler for the signal #2e

pascack_click_sig_2e()

> Source: MainLine.jsx, line 688

pascack_click_sig_2w

Event handler for the signal #2w

pascack_click_sig_2w()

> Source: MainLine.jsx, line 656

pascack_click_sig_4e

Event handler for the signal #4e

pascack_click_sig_4e()

> Source: MainLine.jsx, line 704

pascack_click_sig_4w

Event handler for the signal #4w

pascack_click_sig_4w()

> Source: MainLine.jsx, line 672

pascack_throw_sw_1

The event handler for switch #1

pascack_throw_sw_1()

> Source: MainLine.jsx, line 720

pascack_throw_sw_3

The event handler for switch #3

pascack_throw_sw_3()

> Source: MainLine.jsx, line 732

port_click_sig_2e_1

The event handler for Signal #2e_1

pa_click_sig_2e_1()

> Source: MainLine.jsx, line 1087

port_click_sig_2e_2

The event handler for Signal #2e_2

pa_click_sig_2e_2()

> Source: MainLine.jsx, line 1102

port_click_sig_2w

The event handler for Signal #2w

pa_click_sig_2w()

> Source: MainLine.jsx, line 1071

port_throw_sw_1

The event handler for switch #1

port_throw_sw_1()

> Source: MainLine.jsx, line 1117

ridgewood_click_sig_2e

The event handler for Signal #2e

ridgewood_click_sig_2e()

> Source: MainLine.jsx, line 2087

ridgewood_click_sig_2w_1

The event handler for Signal #2w_1

ridgewood_click_sig_2w_1()

> Source: MainLine.jsx, line 2019

ridgewood_click_sig_2w_2

The event handler for Signal #2w_2

ridgewood_click_sig_2w_2()

> Source: MainLine.jsx, line 2036

ridgewood_click_sig_4e

The event handler for Signal #4e

ridgewood_click_sig_4e()

> Source: MainLine.jsx, line 2105

ridgewood_click_sig_4w

The event handler for Signal #4w

ridgewood_click_sig_4w()

> Source: MainLine.jsx, line 2053

ridgewood_click_sig_6e

The event handler for Signal #6e

ridgewood_click_sig_6e()

> Source: MainLine.jsx, line 2123

ridgewood_click_sig_6w

The event handler for Signal #6w

ridgewood_click_sig_6w()

> Source: MainLine.jsx, line 2070

ridgewood_throw_sw_1

The event handler for switch #1

ridgewood_throw_sw_1()

> Source: MainLine.jsx, line 2141

ridgewood_throw_sw_3

The event handler for switch #3

ridgewood_throw_sw_3()

> Source: MainLine.jsx, line 2153

ridgewood_throw_sw_5

The event handler for switch #5

ridgewood_throw_sw_5()

> Source: MainLine.jsx, line 2165

ridgewood_throw_sw_7

The event handler for switch #7

ridgewood_throw_sw_7()

> Source: MainLine.jsx, line 2177

ridgewood_throw_sw_9

The event handler for switch #9

ridgewood_throw_sw_9()

> Source: MainLine.jsx, line 2189

sf_click_sig_2e

The event handler for Signal #2e

sf_click_sig_2e()

> Source: MainLine.jsx, line 1792

sf_click_sig_2w

The event handler for Signal #2w

sf_click_sig_2w()

> Source: MainLine.jsx, line 1759

sf_click_sig_4e_1

The event handler for Signal #4e_1

sf_click_sig_4e_1()

Source: MainLine.jsx, line 1807

sf_click_sig_4e_2

The event handler for Signal #4e_2

sf_click_sig_4e_2()

Source: MainLine.jsx, line 1823

sf_click_sig_4w

The event handler for Signal #4w

sf_click_sig_4w()

Source: MainLine.jsx, line 1776

sf_throw_sw_1

The event handler for switch #1

sf_throw_sw_1()

Source: MainLine.jsx, line 1839

sf_throw_sw_3

The event handler for switch #3

sf_throw_sw_3()

Source: MainLine.jsx, line 1851

sparrow_click_sig_2e

The event handler for Signal #2e

sparrow_click_sig_2e()

Source: MainLine.jsx, line 917

sparrow_click_sig_2w_1

The event handler for Signal #2w_1

sparrow_click_sig_2w_1()

Source: MainLine.jsx, line 872

sparrow_click_sig_2w_2

The event handler for Signal #2w_2

sparrow_click_sig_2w_2()

Source: MainLine.jsx, line 887

sparrow_click_sig_2w_3

The event handler for Signal #2w_3

sparrow_click_sig_2w_3()

> Source: MainLine.jsx, line 902

sparrow_throw_sw_1

The event handler for switch #1

sparrow_throw_sw_1()

> Source: MainLine.jsx, line 934

sparrow_throw_sw_3

The event handler for switch #3

sparrow_throw_sw_3()

> Source: MainLine.jsx, line 946

state

Object that holds the state or status information for the component This object holds all the information for everything on the pannel that is required to display the routes correctly

State

> Source: MainLine.jsx, line 99

sterling_click_sig_1e

The event handler for Signal #1e

sterling_click_sig_1e()

> Source: MainLine.jsx, line 1663

sterling_click_sig_2w

The event handler for Signal #2w

sterling_click_sig_2w()

> Source: MainLine.jsx, line 1633

sterling_click_sig_2ws

The event handler for Signal #2ws

sterling_click_sig_2ws()

> Source: MainLine.jsx, line 1648

sterling_throw_sw_21

The event handler for switch #21

sterling_throw_sw_21()

> Source: MainLine.jsx, line 1679

suscon_click_sig_2e

The event handler for Signal #2e

suscon_click_sig_2e()

> Source: MainLine.jsx, line 2221

suscon_click_sig_2w

The event handler for Signal #2w

suscon_click_sig_2w()

> Source: MainLine.jsx, line 2204

suscon_click_sig_4e

The event handler for Signal #4e

suscon_click_sig_4e()

> Source: MainLine.jsx, line 2255

suscon_click_sig_4w

The event handler for Signal #4w

suscon_click_sig_4w()

> Source: MainLine.jsx, line 2238

suscon_throw_sw_1

The event handler for switch #1

suscon_throw_sw_1()

> Source: MainLine.jsx, line 2272

suscon_throw_sw_3

The event handler for switch #3

suscon_throw_sw_3()

> Source: MainLine.jsx, line 2284

update_blocks

This function is called every 0.5 Seconds and updates all the tracks blocks

When this function is called it call 2 functions in the CTC controler class. The first one will check find all the routes at each interlocking and set the correct next block to routed, so the route can be displayed on the pannel The second will get all the trains current locations and make those blocks as occupied, to show the correct location of each train on the pannel

> Source: MainLine.jsx, line 150

update_trains

This function is called every 2 Seconds and updates all the Trains locations

When this function is called it will call 2 functions in the CTC controler The first function updates the trains allowing them to move to the next location if the correct time has be spend in their current block The second function updates the interlockings showing if they are occupied or cleared if the correct time has passed

> Source: MainLine.jsx, line 180

valley_click_sig_1e

The event handler for Signal #1e

valley_click_sig_1e()

> Source: MainLine.jsx, line 1513

valley_click_sig_1w

The event handler for Signal #1w

valley_click_sig_1w()

> Source: MainLine.jsx, line 1483

valley_click_sig_2w

The event handler for Signal #2w

valley_click_sig_2w()

> Source: MainLine.jsx, line 1498

valley_throw_sw_21

The event handler for switch #21

valley_throw_sw_21()

> Source: MainLine.jsx, line 1529

wc_click_sig_2e_1

The event handler for Signal #2e_1

wc_click_sig_2e_1()

> Source: MainLine.jsx, line 1917

wc_click_sig_2e_2

The event handler for Signal #2e_2

wc_click_sig_2e_2()

> Source: MainLine.jsx, line 1934

wc_click_sig_2w_1

The event handler for Signal #2w_1

wc_click_sig_2w_1()

> Source: MainLine.jsx, line 1866

wc_click_sig_2w_2

The event handler for Signal #2w_2

wc_click_sig_2w_2()

> Source: MainLine.jsx, line 1883

wc_click_sig_4e

The event handler for Signal #4e

wc_click_sig_4e()

> Source: MainLine.jsx, line 1951

wc_click_sig_4w

The event handler for Signal #4w

wc_click_sig_4w()

> Source: MainLine.jsx, line 1900

wc_throw_sw_1

The event handler for switch #1

wc_throw_sw_1()

> Source: MainLine.jsx, line 1968

wc_throw_sw_3

The event handler for switch #3

wc_throw_sw_3()

> Source: MainLine.jsx, line 1980

wc_throw_sw_5

The event handler for switch #5

wc_throw_sw_5()

Source: MainLine.jsx, line 1992

wc_throw_sw_7

The event handler for switch #7

wc_throw_sw_7()

Source: MainLine.jsx, line 2004

westSecaucus_click_sig_2e

The event handler for Signal #2e

westSecaucus_click_sig_2e()

Source: MainLine.jsx, line 2411

westSecaucus_click_sig_2w

The event handler for Signal #2w

westSecaucus_click_sig_2w()

Source: MainLine.jsx, line 2394

westSecaucus_click_sig_4e

The event handler for Signal #4e

westSecaucus_click_sig_4e()

Source: MainLine.jsx, line 2445

westSecaucus_click_sig_4w

The event handler for Signal #4w

westSecaucus_click_sig_4w()

Source: MainLine.jsx, line 2428

westSecaucus_throw_sw_1

The event handler for switch #1

westSecaucus_throw_sw_1()

Source: MainLine.jsx, line 2462

westSecaucus_throw_sw_3

The event handler for switch #3

westSecaucus_throw_sw_3()

Source: MainLine.jsx, line 2474

Methods

componentDidMount()

ReactJS function that allows you do set the intervals for when certin functions are called

This function sets the intervals for each function that is called repeadely after a amount of time Will call the update_blocks() function every 0.5 Seconds Will call the update_trains() function every 2 Seconds

Source: MainLine.jsx, line 235

componentWillUnmount()

ReactJS function that removes the intervals, this is never called in this program

This function deletes the intervals that are used to update the blocks & trains This is never called in this program

Source: MainLine.jsx, line 250

render()

standard React function that draws all the other interlockings and track components to the screen

This will draw all the components to the screen to assemble the pannel, it also passes all the function and information to each components through their properties or (props)

Source: MainLine.jsx, line 263

Class: Hilburn

Hilburn()

The React JSX Component Class for the Hilburn Interlocking This class is a JSX React Component for the Hilburn Interlocking, this will control all the UI for the component, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Hilburn()

Source: Hilburn.jsx, line 46

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Hilburn.jsx, line 225

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: Hilburn.jsx, line 54

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component

The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Hilburn.jsx, line 80

render()

standard React function that draws the interlocking to the screen

render()

Source: Hilburn.jsx, line 93

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Hilburn.jsx, line 245

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Hilburn.jsx, line 129

Class: Laurel

Laurel()

The React JSX Component Class for the Laurel Interlocking This class is a JSX React Component for the Laurel Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Laurel()

Source: Laurel.jsx, line 68

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Laurel.jsx, line 2007

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Laurel.jsx, line 78

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Laurel.jsx, line 131

render()

standard React function that draws the interlocking to the screen

render()

Source: Laurel.jsx, line 157

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

reset_drawings()

Source: Laurel.jsx, line 2079

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not. There are a lot of possible drawings for this interlocking, which is why the function is so long, I'm not sure if there is a quicker or faster way to accomplish what this function does

set_route_drawings()

Source: Laurel.jsx, line 221

Class: MainLineTracks

MainLineTracks()

The React JSX Component Class for the Tracks in the Main Line portion This class is a JSX React Component for the Main Line Tracks, this will control all the UI for the comonent, showing what blocks are occupied by a train

Constructor

new MainLineTracks()

Source: MainLineTracks.jsx, line 24

Members

state

State

Source: MainLineTracks.jsx, line 34

Methods

componentWillReceiveProps(nextProps,)

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: MainLineTracks.jsx, line 106

render()

render()

Source: MainLineTracks.jsx, line 177

Class: Mill

Mill()

The React JSX Component Class for the Mill Interlocking This class is a JSX React Component for the Mill Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Mill()

Source: Mill.jsx, line 68

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Mill.jsx, line 443

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Mill.jsx, line 78

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Mill.jsx, line 111

render()

standard React function that draws the interlocking to the screen

render()

Source: Mill.jsx, line 129

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

reset_drawings()

Source: Mill.jsx, line 475

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not.

set_route_drawings()

Source: Mill.jsx, line 169

Class: RidgewoodJunction

RidgewoodJunction()

The React JSX Component Class for the Ridgewood Junction Interlocking This class is a JSX React Component for the Ridgewood Junction Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new RidgewoodJunction()

Source: RidgewoodJunction.jsx, line 73

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: RidgewoodJunction.jsx, line 1260

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: RidgewoodJunction.jsx, line 83

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: RidgewoodJunction.jsx, line 132

render()

standard React function that draws the interlocking to the screen

render()

Source: RidgewoodJunction.jsx, line 155

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: RidgewoodJunction.jsx, line 1320

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: RidgewoodJunction.jsx, line 207

Class: SF

SF()

The React JSX Component Class for the SF Interlocking This class is a JSX React Component for the SF Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new SF()

Source: SF.jsx, line 60

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: SF.jsx, line 500

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: SF.jsx, line 70

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: SF.jsx, line 105

render()

standard React function that draws the interlocking to the screen

render()

Source: SF.jsx, line 122

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: SF.jsx, line 530

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: SF.jsx, line 163

Class: Suscon

Suscon()

The React JSX Component Class for the Suscon Interlocking This class is a JSX React Component for the Suscon Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Suscon()

Source: Suscon.jsx, line 67

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Suscon.jsx, line 439

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Suscon.jsx, line 77

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Suscon.jsx, line 108

render()

standard React function that draws the interlocking to the screen

render()

Source: Suscon.jsx, line 125

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

reset_drawings()

Source: Suscon.jsx, line 471

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not.

set_route_drawings()

Source: Suscon.jsx, line 169

Class: WC

WC()

The React JSX Component Class for the WC Interlocking This class is a JSX React Component for the WC Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new WC()

Source: WC.jsx, line 80

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: WC.jsx, line 771

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: WC.jsx, line 90

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: WC.jsx, line 132

render()

standard React function that draws the interlocking to the screen

render()

Source: WC.jsx, line 151

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: WC.jsx, line 821

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: WC.jsx, line 198

Class: WestSecaucus

WestSecaucus()

The React JSX Component Class for the West Secaucus Interlocking This class is a JSX React Component for the West Secaucus Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new WestSecaucus()

Source: WestSecaucus.jsx, line 54

Members

set_route_drawing

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: WestSecaucus.jsx, line 155

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: WestSecaucus.jsx, line 353

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: WestSecaucus.jsx, line 64

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: WestSecaucus.jsx, line 94

render()

standard React function that draws the interlocking to the screen

render()

Source: WestSecaucus.jsx, line 117

Class: BergenTracks

BergenTracks()

The React JSX Component Class for the Tracks in the Bergen County Line portion his class is a JSX React Component for the Bergen County Line Tracks, this will control all the UI for the comonent, showing what blocks are occupied by a train

Constructor

new BergenTracks()

Source: BergenTracks.jsx, line 22

Members

state

Object that holds the state or status information for the component This object holds all the information for the tracks that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: BergenTracks.jsx, line 29

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: BergenTracks.jsx, line 68

render()

standard React function that draws the interlocking to the screen

render()

Source: BergenTracks.jsx, line 107

Class: BT

BT()

The React JSX Component Class for the BT Interlocking This class is a JSX React Component for the BT Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new BT()

Source: BT.jsx, line 73

Members

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: BT.jsx, line 80

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: BT.jsx, line 115

render()

standard React function that draws the interlocking to the screen

render()

Source: BT.jsx, line 134

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: BT.jsx, line 636

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: BT.jsx, line 176

set_switch_images()

Changes image sources for the switches, depending on switch status

This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: BT.jsx, line 596

Class: HX

HX()

The React JSX Component Class for the HX Interlocking This class is a JSX React Component for the HX Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new HX()

Source: HX.jsx, line 58

Members

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: HX.jsx, line 66

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component

The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: HX.jsx, line 105

render()

standard React function that draws the interlocking to the screen

render()

Source: HX.jsx, line 124

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: HX.jsx, line 712

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: HX.jsx, line 168

set_switch_images()

Changes image sources for the switches, depending on switch status

This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: HX.jsx, line 672

Class: PascackJct

PascackJct()

The React JSX Component Class for the Pascack Junction Interlocking This class is a JSX React Component for the Pascack Junction Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new PascackJct()

Source: PascackJct.jsx, line 65

Members

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: PascackJct.jsx, line 73

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component

The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: PascackJct.jsx, line 106

render()

standard React function that draws the interlocking to the screen

render()

Source: PascackJct.jsx, line 124

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: PascackJct.jsx, line 445

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: PascackJct.jsx, line 163

set_switch_images()

Changes image sources for the switches, depending on switch status

This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: PascackJct.jsx, line 415

Class: BC

BC()

The React JSX Component Class for the BC Interlocking This class is a JSX React Component for the BC Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new BC()

Source: BC.jsx, line 47

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: BC.jsx, line 57

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: BC.jsx, line 83

render()

standard React function that draws the interlocking to the screen

render()

Source: BC.jsx, line 96

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: BC.jsx, line 246

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: BC.jsx, line 132

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: BC.jsx, line 226

Class: CentralValley

CentralValley()

The React JSX Component Class for the Central Valley Interlocking This class is a JSX React Component for the Central Valley Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new CentralValley()

Source: CentralValley.jsx, line 46

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: CentralValley.jsx, line 56

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: CentralValley.jsx, line 81

render()

standard React function that draws the interlocking to the screen

Source: CentralValley.jsx, line 93

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: CentralValley.jsx, line 248

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: CentralValley.jsx, line 135

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: CentralValley.jsx, line 228

Class: Hall

Hall()

The React JSX Component Class for the Hall Interlocking This class is a JSX React Component for the Hall Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Hall()

Source: Hall.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Hall.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Hall.jsx, line 94

render()

standard React function that draws the interlocking to the screen

render()

Source: Hall.jsx, line 110

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Hall.jsx, line 365

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Hall.jsx, line 148

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Hall.jsx, line 348

Class: Harriman

Harriman()

The React JSX Component Class for the Harriman Interlocking This class is a JSX React Component for the Harriman Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Harriman()

Source: Harriman.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Harriman.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Harriman.jsx, line 93

render()

standard React function that draws the interlocking to the screen

render()

Source: Harriman.jsx, line 107

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Harriman.jsx, line 316

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Harriman.jsx, line 146

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Harriman.jsx, line 286

Class: HudsonJunction

HudsonJunction()

The React JSX Component Class for the Hudson Junction Interlocking This class is a JSX React Component for the Hudson Junction Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new HudsonJunction()

Source: HudsonJunction.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: HudsonJunction.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: HudsonJunction.jsx, line 93

render()

standard React function that draws the interlocking to the screen

render()

Source: HudsonJunction.jsx, line 107

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: HudsonJunction.jsx, line 360

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: HudsonJunction.jsx, line 146

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: HudsonJunction.jsx, line 330

Class: OV

OV()

The React JSX Component Class for the OV Interlocking This class is a JSX React Component for the OV Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new OV()

Source: OV.jsx, line 47

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: OV.jsx, line 57

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: OV.jsx, line 83

render()

standard React function that draws the interlocking to the screen

render()

Source: OV.jsx, line 96

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: OV.jsx, line 246

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: OV.jsx, line 132

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: OV.jsx, line 226

Class: PA

PA()

The React JSX Component Class for the PA Interlocking This class is a JSX React Component for the PA Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new PA()

Source: PA.jsx, line 60

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: PA.jsx, line 70

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: PA.jsx, line 105

render()

standard React function that draws the interlocking to the screen

render()

Source: PA.jsx, line 122

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: PA.jsx, line 535

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: PA.jsx, line 163

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: PA.jsx, line 505

Class: Port

Port()

The React JSX Component Class for the PA Interlocking This class is a JSX React Component for the PA Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Port()

Source: Port.jsx, line 46

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Port.jsx, line 56

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Port.jsx, line 82

render()

standard React function that draws the interlocking to the screen

render()

Source: Port.jsx, line 95

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Port.jsx, line 245

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Port.jsx, line 131

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Port.jsx, line 225

Class: SouthernTierTracks

SouthernTierTracks()

The React JSX Component Class for the Tracks in the Southern Tier portion This class is a JSX React Component for the Southern Tier Tracks, this will control all the UI for the comonent, showing what blocks are occupied by a train

Constructor

new SouthernTierTracks()

Source: SouthernTierTracks.jsx, line 24

Members

state

Object that holds the state or status information for the component This object holds all the information for the tracks that is required to display the routes correctly

State

Source: SouthernTierTracks.jsx, line 32

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: SouthernTierTracks.jsx, line 103

render()

standard React function that draws the interlocking to the screen

render()

Source: SouthernTierTracks.jsx, line 173

Class: Sparrow

Sparrow()

The React JSX Component Class for the Sparrow Interlocking This class is a JSX React Component for the Sparrow Interlocking, this will control all the UI for the component, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Sparrow()

Source: Sparrow.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Sparrow.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Sparrow.jsx, line 93

render()

standard React function that draws the interlocking to the screen

render()

Source: Sparrow.jsx, line 106

reset_drawing()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Sparrow.jsx, line 317

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Sparrow.jsx, line 145

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Sparrow.jsx, line 287

Class: Sterling

Sterling()

The React JSX Component Class for the Hilburn Interlocking This class is a JSX React Component for the Hilburn Interlocking, this will control all the UI for the component, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Sterling()

Source: Sterling.jsx, line 45

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Sterling.jsx, line 55

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Sterling.jsx, line 81

render()

standard React function that draws the interlocking to the screen

render()

Source: Sterling.jsx, line 94

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Sterling.jsx, line 244

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Sterling.jsx, line 130

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Sterling.jsx, line 224

Part VII: Code

SEE NEXT PAGE

```
/**
 * @file index.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary The main extry point for the program
 */
import React from "react";
import ReactDOM from "react-dom";

import MainLine from './components/Panel/MainLine.jsx';

ReactDOM.render(<div> <MainLine /> </div>,
document.getElementById('app'));
```

```
/**
 * @file mainLine_ctc.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller that uses all the other CTC classes and
controlls basically the entire game
 */


// Import my custom clock class
import Clock from '../Trains/clock.js';
// Import the block class, that is a piece of track
import CTC_Block from '../CTC/ctc_block.js';

// Southern Tier Interlockings
import CTC_Sparrow from '../Interlockings/Southern_Tier/
ctc_sparrow.js';
import CTC_PA from '../Interlockings/Southern_Tier/ctc_pa.js';
import CTC_Port from '../Interlockings/Southern_Tier/ctc_port.js';
import CTC_BC from '../Interlockings/Southern_Tier/ctc_bc.js';
import CTC_OV from '../Interlockings/Southern_Tier/ctc_ov.js';
import CTC_Howells from '../Interlockings/Southern_Tier/
ctc_howells.js';
import CTC_Hall from '../Interlockings/Southern_Tier/ctc_hall.js';
import CTC_Hudson from '../Interlockings/Southern_Tier/ctc_hudson.js';
import CTC_Valley from '../Interlockings/Southern_Tier/ctc_valley.js';
import CTC_Harriman from '../Interlockings/Southern_Tier/
ctc_harriman.js';
import CTC_Sterling from '../Interlockings/Southern_Tier/
ctc_sterling.js';

// Main Line Interlockings
import CTC_Hilburn from '../Interlockings/Main_Line/ctc_hilburn.js';
import CTC_SF from '../Interlockings/Main_Line/ctc_sf.js';
import CTC_WC from '../Interlockings/Main_Line/ctc_wc';
import CTC_Ridgewood from '../Interlockings/Main_Line/
ctc_ridgewood.js';
import CTC_Suscon from '../Interlockings/Main_Line/ctc_suscon.js';
import CTC_Mill from '../Interlockings/Main_Line/ctc_mill.js';
import CTC_WestSecaucus from '../Interlockings/Main_Line/
ctc_westSecaucus.js';
import CTC_Laurel from '../Interlockings/Main_Line/ctc_laurel.js';

// Bergen County Interlockings
import CTC_BT from '../Interlockings/Bergen_Line/ctc_bt.js';
import CTC_Pascack from '../Interlockings/Bergen_Line/ctc_pascack.js';
import CTC_HX from '../Interlockings/Bergen_Line/ctc_hx.js';


/**
```

```
 * Class that runs the entire railroad, and the routes and train
movements. Controlls updating all the blocks and trains, and routes.
 * It really is the engine behind everything in the simulation.
 *
 * @member game_clock -> Clock class to keep track of time in the
simulation
 *
 * @member train_list -> An array that holds all the trains that are
on the railroad
 *
 * @member interlocking_sparrow -> The CTC class for CP Sparrow
 * @member interlocking_pa -> The CTC class for CP PA
 * @member interlocking_port -> The CTC class for CP Port
 * @member interlocking_bc -> The CTC class for CP BC
 * @member interlocking_ov -> The CTC class for CP OV
 * @member interlocking_howells -> The CTC class for CP Howells
 * @member interlocking_hall -> The CTC class for CP Hall
 * @member interlocking_hudson -> The CTC class for CP Hudson Junction
 * @member interlocking_valley -> The CTC class for CP Central Valley
 * @member interlocking_harriman -> The CTC class for CP Harriman
 * @member interlocking_sterling -> The CTC class for CP Sterling
 *
 * @member interlocking_hilburn -> The CTC class for the Hilburn
Interlocking
 * @member interlocking_sf -> The CTC class for the SF Interlocking
 * @member interlocking_wc -> The CTC class for the WC Interlocking
 * @member interlocking_ridgewood -> The CTC class for the Ridgewood
Junction Interlocking
 * @member interlocking_suscon -> The CTC class for the Suscon
Interlocking
 * @member interlocking_mill -> The CTC class for the Mill
Interlocking
 * @member interlocking_westSecacus -> The CTC class for the West
Secacus Interlocking
 *
 * @member interlocking_bt -> The CTC class for the BT Interlocking
 * @member interlocking_pascack -> THE CTC class for Pascack Junction
Interlocking
 * @member interlocking_hx -> The CTC class for HX Interlocking
 *
 * @member blocks_mainLine -> An object that holds all the Block
classes for the railroad
 */
class MainLine_CTC {
    /**
     * constructor()
     * @summary The constructor for the Clock class
     *
     * @details This will initialize all the member variables when the
program is started
```

```
     */
    constructor() {
        this.game_clock = new Clock();
        this.game_clock.startClock();

        this.train_list = [];

        this.interlocking_sparrow = new CTC_Sparrow();
        this.interlocking_pa = new CTC_PA();
        this.interlocking_port = new CTC_Port();
        this.interlocking_bc = new CTC_BC();
        this.interlocking_ov = new CTC_OV();
        this.interlocking_howells = new CTC_Howells();
        this.interlocking_hall = new CTC_Hall();
        this.interlocking_hudson = new CTC_Hudson();
        this.interlocking_valley = new CTC_Valley();
        this.interlocking_harriman = new CTC_Harriman();
        this.interlocking_sterling = new CTC_Sterling();

        this.interlocking_hilburn = new CTC_Hilburn();
        this.interlocking_sf = new CTC_SF();
        this.interlocking_wc = new CTC_WC();
        this.interlocking_ridgewood = new CTC_Ridgewood();
        this.interlocking_suscon = new CTC_Suscon();
        this.interlocking_mill = new CTC_Mill();
        this.interlocking_westSecaucus = new CTC_WestSecaucus();
        this.interlocking_laurel = new CTC_Laurel();

        this.interlocking_bt = new CTC_BT();
        this.interlocking_pascack = new CTC_Pascack();
        this.interlocking_hx = new CTC_HX();

        this.blocks_mainLine = {
            // Southern Tier Blocks
            block_harriman_sterling_1: new
CTC_Block("1_harriman_sterling", 105),

            block_valley_harriman_1: new
CTC_Block("1_valley_harriman", 28),
            block_valley_harriman_2: new
CTC_Block("2_valley_harriman", 28),
            block_harriman_industrial: new
CTC_Block("1_harriman_industrial", 8),

            block_hudson_valley_1: new CTC_Block("1_hudson_valley",
156),
            block_hudson_nysw: new CTC_Block("2_hudson_nysw", 8),

            block_hall_hudson_1: new CTC_Block("1_hall_hudson", 13),
            block_hall_hudson_2: new CTC_Block("2_hall_hudson", 13),
```

```
            block_hall_yard: new CTC_Block("2_hall_yard", 8),

            block_howells_hall_1: new CTC_Block("1_howells_hall",
132),

            block_ov_howells_1: new CTC_Block("1_ov_howells", 59),
            block_ov_howells_2: new CTC_Block("2_ov_howells", 59),

            block_bc_ov_1: new CTC_Block("1_bc_ov", 117),

            block_port_bc_1: new CTC_Block("1_port_bc", 8),
            block_pa_port_1: new CTC_Block("1_pa_port", 8),
            block_pa_bc_2: new CTC_Block("2_pa_bc", 16),
            block_port_yard_west: new CTC_Block("3_port_yard_west",
8),
            block_port_yard_east: new CTC_Block("3_port_yard_east",
8),

            block_buckleys_west: new CTC_Block("4_buckleys_west", 12),
            block_buckleys_east: new CTC_Block("4_buckleys_east", 12),

            block_sparrow_pa_1: new CTC_Block("1_sparrow_pa", 16),
            block_sparrow_pa_2: new CTC_Block("2_sparrow_pa", 16),
            block_sparrow_cripple: new CTC_Block("3_sparrow_cripple",
8),

            block_bingo_sparrow: new CTC_Block("1_bingo_sparrow", 25),

            // Main Line Blocks
            block_westEnd_laurel_1: new CTC_Block("1_westEnd_laurel",
8),
            block_westEnd_laurel_2: new CTC_Block("2_westEnd_laurel",
8),
            block_westEnd_laurel_3: new CTC_Block("3_westEnd_laurel",
8),
            block_westEnd_laurel_4: new CTC_Block("4_westEnd_laurel",
8),

            block_westSecaucus_laurel_1: new
CTC_Block("1_laurel_westSecaucus", 8),
            block_westSecaucus_laurel_2: new
CTC_Block("2_laurel_westSecaucus", 8),

            block_mill_westSecaucus_1: new
CTC_Block("1_mill_westSecaucus", 61),
            block_mill_westSecaucus_2: new
CTC_Block("2_mill_westSecaucus", 61),

            block_suscon_mill_1: new CTC_Block("1_suscon_mill", 64),
            block_suscon_mill_2: new CTC_Block("2_suscon_mill", 64),
```

```
            block_ridgewood_suscon_1: new
CTC_Block("1_ridgewood_suscon", 28),
            block_ridgewood_suscon_2: new
CTC_Block("2_ridgewood_suscon", 28),

            block_wc_ridgewood_3: new CTC_Block("3_wc_ridgewood", 33),
            block_wc_ridgewood_1: new CTC_Block("1_wc_ridgewood", 33),
            block_wc_ridgewood_2: new CTC_Block("2_wc_ridgewood", 33),

            block_sf_wc_1: new CTC_Block("1_sf_wc", 69),
            block_sf_wc_2: new CTC_Block("2_sf_wc", 69),

            block_hilburn_sf: new CTC_Block("2_hilburn_sf", 20),

            block_sterling_sf: new CTC_Block("1_sterling_sf", 40),
            block_sterling_hilburn: new
CTC_Block("2_sterling_hilburn", 20),

            block_hilburn_yard_west: new
CTC_Block("1_hilburn_yard_west", 8),
            block_hilburn_yard_east: new
CTC_Block("1_hilburn_yard_east", 8),
            block_wc_yard: new CTC_Block("1_wc_yard", 8),

            // Bergen County Blocks
            block_hx_laurel_1: new CTC_Block("1_hx_laurel", 11),
            block_hx_laurel_2: new CTC_Block("2_hx_laurel", 11),

            block_pascack_hx_1: new CTC_Block("1_pascack_hx", 22),
            block_pascack_hx_2: new CTC_Block("2_pascack_hx", 22),

            block_bt_pascack_1: new CTC_Block("1_bt_pascack", 66),
            block_bt_pascack_2: new CTC_Block("2_bt_pascack", 66),

            block_ridgewood_bt_1: new CTC_Block("1_ridgewood_bt", 61),
            block_ridgewood_bt_2: new CTC_Block("2_ridgewood_bt", 61),

            block_bt_nysw: new CTC_Block("3_bt_nysw", 8),
            block_hx_croxton_1: new CTC_Block("1_hx_croxton", 8),
            block_hx_croxton_2: new CTC_Block("2_hx_croxton", 8)
        };
    }
    // ---- END constructor() ----

    /**
     *  update_route_blocks()
     * @summary Gets all the routes from each interlocking and sets
the accoriding blocks
     */
```

```
    update_route_blocks() {
        // Reset All The Blocks
        this.reset_route_mainLine_blocks();

        let routes = [];

        // Add Main Line Routes
        routes = routes.concat(this.interlocking_laurel.get_routes());
        routes =
routes.concat(this.interlocking_westSecaucus.get_routes());
        routes = routes.concat(this.interlocking_mill.get_routes());
        routes = routes.concat(this.interlocking_suscon.get_routes());
        routes =
routes.concat(this.interlocking_ridgewood.get_routes());
        routes = routes.concat(this.interlocking_wc.get_routes());
        routes = routes.concat(this.interlocking_sf.get_routes());
        routes =
routes.concat(this.interlocking_hilburn.get_routes());

        // Add Bergen County Routes
        routes = routes.concat(this.interlocking_hx.get_routes());
        routes =
routes.concat(this.interlocking_pascack.get_routes());
        routes = routes.concat(this.interlocking_bt.get_routes());

        // Add Southern Tier Routes
        routes =
routes.concat(this.interlocking_sterling.get_routes());
        routes =
routes.concat(this.interlocking_harriman.get_routes());
        routes = routes.concat(this.interlocking_valley.get_routes());
        routes = routes.concat(this.interlocking_hudson.get_routes());
        routes = routes.concat(this.interlocking_hall.get_routes());
        routes =
routes.concat(this.interlocking_howells.get_routes());
        routes = routes.concat(this.interlocking_ov.get_routes());
        routes = routes.concat(this.interlocking_bc.get_routes());
        routes = routes.concat(this.interlocking_port.get_routes());
        routes = routes.concat(this.interlocking_pa.get_routes());
        routes =
routes.concat(this.interlocking_sparrow.get_routes());

        // Update all the blocks that have routes
        for (var i = 0; i < routes.length; i++) {
            if (routes[i] === null) {
                // Do Nothing
            }
            else {
                let name = routes[i].substr(routes[i].indexOf("|") +
3, routes[i].size);
```

```
this.get_block_by_name(name).set_block_status("Route");
            }
        }
    }
    // ---- END update_route_blocks() ----

    /**
     * updates_trains()
     * @summary Goes through all the trains in the list and updates
their location if they're capable of doing so
     */
    update_trains() {
        // Loop through all the trains
        for (let i = 0; i < this.train_list.length; i++) {
            if (this.train_list[i].can_update_location()) {
                let new_route =
this.get_interlocking_route(this.train_list[i].get_location(),
this.train_list[i].get_direction());
                if (new_route === null) {
                    // Do Nothing
                    // Train Cannot Move
                }
                else if (new_route === undefined) {
                    // Clear Previous Block

this.get_block_by_name(this.train_list[i].get_location()).set_symbol('
');

this.get_block_by_name(this.train_list[i].get_location()).set_block_st
atus("Empty");

                    this.train_list.splice(i, 1);
                    break;
                }
                else {
                    // Clear Previous Block

this.get_block_by_name(this.train_list[i].get_location()).set_symbol('
');

this.get_block_by_name(this.train_list[i].get_location()).set_block_st
atus("Empty");

                    // Get the last location
                    let location = this.train_list[i].get_location();

                    // Occupy the Interlockings
                    if (this.train_list[i].get_direction() === "WEST")
{
```

```
                        let cp_trk = location.substr(0,
location.indexOf("_"));
                        let cp = this.train_list[i].get_location();
                        cp = cp.substr(cp.indexOf("_") + 1,
cp.lastIndexOf("_") - 2);
                        //console.log(cp_trk, cp);
                        this.set_occupy_interlocking(cp_trk, cp);

                        // Occupy the Next Block
                        let block = new_route.substr(10,
new_route.size);

this.train_list[i].set_block_size(this.get_block_by_name(block).get_si
ze());
                        //this.train_list[i].set_block_size(8);
                        this.train_list[i].update_location(block);
                    }
                    else {
                        let cp_trk = location.substr(0,
location.indexOf("_"));
                        let cp = this.train_list[i].get_location();
                        cp = cp.substr(cp.lastIndexOf("_") + 1,
cp.size);
                        console.log(cp_trk, cp);
                        this.set_occupy_interlocking(cp_trk, cp);

                        // Occupy the Next Block
                        let block = new_route.substr(10,
new_route.size);

this.train_list[i].set_block_size(this.get_block_by_name(block).get_si
ze());
                        //this.train_list[i].set_block_size(8);
                        this.train_list[i].update_location(block);
                    }
                }
            }
        }
    }
    // ---- END update_trains() ----

    /**
     * update_interlockings()
     * @summary Goes through to see if each interlocking can have a
train clear if it's occupied
     */
    update_interlockings() {
        // Bergen County Line
        this.interlocking_hx.can_clear();
        this.interlocking_pascack.can_clear();
```

```
    this.interlocking_bt.can_clear();

    // Main Line
    this.interlocking_laurel.can_clear();
    this.interlocking_westSecaucus.can_clear();
    this.interlocking_mill.can_clear();
    this.interlocking_suscon.can_clear();
    this.interlocking_ridgewood.can_clear();
    this.interlocking_wc.can_clear();
    this.interlocking_sf.can_clear();
    this.interlocking_hilburn.can_clear();

    // Southern Tier Line
    this.interlocking_sterling.can_clear();
    this.interlocking_harriman.can_clear();
    this.interlocking_valley.can_clear();
    this.interlocking_hudson.can_clear();
    this.interlocking_hall.can_clear();
    this.interlocking_howells.can_clear();
    this.interlocking_ov.can_clear();
    this.interlocking_bc.can_clear();
    this.interlocking_port.can_clear();
    this.interlocking_pa.can_clear();
    this.interlocking_sparrow.can_clear();
}
// ---- END update_interlockings() ----

/**
 * get_sparrow()
 * @summary Gets reference to the CP Sparrow Interlocking
 *
 * @returns Reference to the CP Sparrow Interlocking
 */
get_sparrow() {
    return this.interlocking_sparrow;
}
// ---- END get_sparrow() ----

/**
 * get_pa()
 * @summary Gets reference to the CP PA Interlocking
 *
 * @returns Reference to the CP PA Interlocking
 */
get_pa() {
    return this.interlocking_pa;
}
// ---- END get_pa() ----

/**
```

```
 * get_port()
 * @summary Gets reference to the CP Port Interlocking
 *
 * @returns Reference to the CP Port Interlocking
 */
get_port() {
    return this.interlocking_port;
}
// ---- END get_port() ----

/**
 * get_bc()
 * @summary Gets reference to the CP BC Interlocking
 *
 * @returns Reference to the CP BC Interlocking
 */
get_bc() {
    return this.interlocking_bc;
}
// ---- END get_bc() ----

/**
 * get_ov()
 * @summary Gets reference to the CP OV Interlocking
 *
 * @returns Reference to the CP OV Interlocking
 */
get_ov() {
    return this.interlocking_ov;
}
// ---- END get_ov() ----

/**
 * get_howells()
 * @summary Gets reference to the CP Howells Interlocking
 *
 * @returns Reference to the CP Howells Interlocking
 */
get_howells() {
    return this.interlocking_howells;
}
// ---- END get_howells() ----

/**
 * get_hall()
 * @summary Gets reference to the CP Hall Interlocking
 *
 * @returns Reference to the CP Hall Interlocking
 */
get_hall() {
```

```
    return this.interlocking_hall;
}
// ---- END get_hall() ----

/**
 * get_hudson()
 * @summary Gets reference to the CP Hudson Junction Interlocking
 *
 * @returns Reference to the CP Hudson Junction Interlocking
 */
get_hudson() {
    return this.interlocking_hudson;
}
// ---- END get_hudson() ----

/**
 * get_valley()
 * @summary Gets reference to the CP Central Valley Interlocking
 *
 * @returns Reference to the CP Central Valley Interlocking
 */
get_valley() {
    return this.interlocking_valley;
}
// ---- END get_valley() ----

/**
 * get_harriman()
 * @summary Gets reference to the CP Harriman Interlocking
 *
 * @returns Reference to the CP Harriman Interlocking
 */
get_harriman() {
    return this.interlocking_harriman;
}
// ---- END get_harriman() ----

/**
 * get_sterling()
 * @summary Gets reference to the CP Sterling Interlocking
 *
 * @returns Reference to the CP Sterling Interlocking
 */
get_sterling() {
    return this.interlocking_sterling;
}
// ---- END get_sterling() ----

/**
 * get_hilburn()
```

```
 * @summary Gets reference to the Hilburn Interlocking
 *
 * @returns Reference to the Hilburn Interlocking
 */
get_hilburn() {
    return this.interlocking_hilburn;
}
// ---- END get_hilburn() ----

/**
 * get_sf()
 * @summary Gets reference to the SF Interlocking
 *
 * @returns Reference to the SF Interlocking
 */
get_sf() {
    return this.interlocking_sf;
}
// ---- END get_sf() ----

/**
 * get_wc()
 * @summary Gets reference to the WC Interlocking
 *
 * @returns Reference to the WC Interlocking
 */
get_wc() {
    return this.interlocking_wc;
}
// ---- END get_wc() ----

/**
 * get_ridgewood()
 * @summary Gets reference to the Ridgewood Junction Interlocking
 *
 * @returns Reference to the Ridgewood Junction Interlocking
 */
get_ridgewood() {
    return this.interlocking_ridgewood;
}
// ---- END get_ridgewood() ----

/**
 * get_suscon()
 * @summary Gets reference to the Suscon Interlocking
 *
 * @returns Reference to the Suscon Interlocking
 */
get_suscon() {
    return this.interlocking_suscon;
```

```
}
// ---- END get_suscon() ----

/**
 * get_mill()
 * @summary Gets reference to the Mill Interlocking
 *
 * @returns Reference to the Mill Interlocking
 */
get_mill() {
    return this.interlocking_mill;
}
// ---- END get_mill() ----

/**
 * get_westSecaucus()
 * @summary Gets reference to the West Secaucus Interlocking
 *
 * @returns Reference to the West Secaucus Interlocking
 */
get_westSecaucus() {
    return this.interlocking_westSecaucus;
}
// ---- END get_westSecaucus() ----

/**
 * get_laurel()
 * @summary Gets reference to the Laurel Interlocking
 *
 * @returns Reference to the Laurel Interlocking
 */
get_laurel() {
    return this.interlocking_laurel;
}
// ---- END get_laurel() ----

/**
 * get_bt()
 * @summary Gets reference to the BT Interlocking
 *
 * @returns Reference to the BT Interlocking
 */
get_bt() {
    return this.interlocking_bt;
}
// ---- END get_bt() ----

/**
 * get_pascack()
 * @summary Gets reference to the Pascack Interlocking
```

```
     *
     * @returns Reference to the Pascack Interlocking
     */
    get_pascack() {
        return this.interlocking_pascack;
    }
    // ---- END get_pascack() ----

    /**
     * get_hx()
     * @summary Gets reference to the HX Interlocking
     *
     * @returns Reference to the HX Interlocking
     */
    get_hx() {
        return this.interlocking_hx;
    }
    // ---- END get_hx() ----

    /**
     * add_train()
     * @summary Takes in a new train and adds it to the train_list
array
     */
    add_train(new_train) {
        this.train_list.push(new_train);
    }
    // ---- END add_train() ----


    /**
     * occupy_blocks()
     * @summary goes through all the trains and finds their current
location and occupys the correct block
     */
    occupy_blocks() {
        for (let i = 0; i < this.train_list.length; i++) {
            let block =
this.get_block_by_name(this.train_list[i].get_location());

            if (block === false) {

            }
            else {
                block.set_block_status("Occupied");
                block.set_symbol(this.train_list[i].get_symbol());
            }
        }
    }
    // ---- END occupy_blocks() ----
```

```
    /**
     * reset_route_mainLine_blocks()
     * @summary Resets all the blocks that are routed
     */
    reset_route_mainLine_blocks() {
        this.blocks_mainLine.block_westEnd_laurel_1.reset_block();
        this.blocks_mainLine.block_westEnd_laurel_2.reset_block();
        this.blocks_mainLine.block_westEnd_laurel_3.reset_block();
        this.blocks_mainLine.block_westEnd_laurel_4.reset_block();


this.blocks_mainLine.block_westSecaucus_laurel_1.reset_block();

this.blocks_mainLine.block_westSecaucus_laurel_2.reset_block();

        this.blocks_mainLine.block_mill_westSecaucus_1.reset_block();
        this.blocks_mainLine.block_mill_westSecaucus_2.reset_block();

        this.blocks_mainLine.block_suscon_mill_1.reset_block();
        this.blocks_mainLine.block_suscon_mill_2.reset_block();

        this.blocks_mainLine.block_ridgewood_suscon_1.reset_block();
        this.blocks_mainLine.block_ridgewood_suscon_2.reset_block();

        this.blocks_mainLine.block_wc_ridgewood_3.reset_block();
        this.blocks_mainLine.block_wc_ridgewood_1.reset_block();
        this.blocks_mainLine.block_wc_ridgewood_2.reset_block();

        this.blocks_mainLine.block_sf_wc_1.reset_block();
        this.blocks_mainLine.block_sf_wc_2.reset_block();

        this.blocks_mainLine.block_hilburn_sf.reset_block();

        this.blocks_mainLine.block_sterling_sf.reset_block();
        this.blocks_mainLine.block_sterling_hilburn.reset_block();

        this.blocks_mainLine.block_hilburn_yard_west.reset_block();
        this.blocks_mainLine.block_hilburn_yard_east.reset_block();

        this.blocks_mainLine.block_wc_yard.reset_block();

        // Bergen County Line
        this.blocks_mainLine.block_hx_laurel_1.reset_block();
        this.blocks_mainLine.block_hx_laurel_2.reset_block();

        this.blocks_mainLine.block_pascack_hx_1.reset_block();
        this.blocks_mainLine.block_pascack_hx_2.reset_block();

        this.blocks_mainLine.block_bt_pascack_1.reset_block();
```

```
    this.blocks_mainLine.block_bt_pascack_2.reset_block();

    this.blocks_mainLine.block_ridgewood_bt_1.reset_block();
    this.blocks_mainLine.block_ridgewood_bt_2.reset_block();

    this.blocks_mainLine.block_bt_nysw.reset_block();
    this.blocks_mainLine.block_hx_croxton_1.reset_block();
    this.blocks_mainLine.block_hx_croxton_2.reset_block();

    // Southern Tier Line
    this.blocks_mainLine.block_harriman_sterling_1.reset_block();

    this.blocks_mainLine.block_valley_harriman_1.reset_block();
    this.blocks_mainLine.block_valley_harriman_2.reset_block();
    this.blocks_mainLine.block_harriman_industrial.reset_block();

    this.blocks_mainLine.block_hudson_valley_1.reset_block();
    this.blocks_mainLine.block_hudson_nysw.reset_block();

    this.blocks_mainLine.block_hall_hudson_1.reset_block();
    this.blocks_mainLine.block_hall_hudson_2.reset_block();
    this.blocks_mainLine.block_hall_yard.reset_block();

    this.blocks_mainLine.block_howells_hall_1.reset_block();

    this.blocks_mainLine.block_ov_howells_1.reset_block();
    this.blocks_mainLine.block_ov_howells_2.reset_block();

    this.blocks_mainLine.block_bc_ov_1.reset_block();

    this.blocks_mainLine.block_port_bc_1.reset_block();
    this.blocks_mainLine.block_pa_port_1.reset_block();
    this.blocks_mainLine.block_pa_bc_2.reset_block();
    this.blocks_mainLine.block_port_yard_west.reset_block();
    this.blocks_mainLine.block_port_yard_east.reset_block();

    this.blocks_mainLine.block_buckleys_west.reset_block();
    this.blocks_mainLine.block_buckleys_east.reset_block();

    this.blocks_mainLine.block_sparrow_pa_1.reset_block();
    this.blocks_mainLine.block_sparrow_pa_2.reset_block();
    this.blocks_mainLine.block_sparrow_cripple.reset_block();

    this.blocks_mainLine.block_bingo_sparrow.reset_block();
}
// ---- END reset_route_mainLine_blocks() ----

/**
 * get_mainLine_blocks_status()
 * @summary Gets the status of all the bloccks on the Southern
```

```
Tier Section
     *
     * @returns An object with the status of each block
     */
    get_mainLine_blocks_status() {
        var status = {
            block_westEnd_laurel_1:
this.blocks_mainLine.block_westEnd_laurel_1.get_block_status(),
            block_westEnd_laurel_2:
this.blocks_mainLine.block_westEnd_laurel_2.get_block_status(),
            block_westEnd_laurel_3:
this.blocks_mainLine.block_westEnd_laurel_3.get_block_status(),
            block_westEnd_laurel_4:
this.blocks_mainLine.block_westEnd_laurel_4.get_block_status(),

            block_laurel_westSecaucus_1:
this.blocks_mainLine.block_westSecaucus_laurel_1.get_block_status(),
            block_laurel_westSecaucus_2:
this.blocks_mainLine.block_westSecaucus_laurel_2.get_block_status(),

            block_mill_westSecaucus_1:
this.blocks_mainLine.block_mill_westSecaucus_1.get_block_status(),
            block_mill_westSecaucus_2:
this.blocks_mainLine.block_mill_westSecaucus_2.get_block_status(),

            block_suscon_mill_1:
this.blocks_mainLine.block_suscon_mill_1.get_block_status(),
            block_suscon_mill_2:
this.blocks_mainLine.block_suscon_mill_2.get_block_status(),

            block_ridgewood_suscon_1:
this.blocks_mainLine.block_ridgewood_suscon_1.get_block_status(),
            block_ridgewood_suscon_2:
this.blocks_mainLine.block_ridgewood_suscon_2.get_block_status(),

            block_wc_ridgewood_3:
this.blocks_mainLine.block_wc_ridgewood_3.get_block_status(),
            block_wc_ridgewood_1:
this.blocks_mainLine.block_wc_ridgewood_1.get_block_status(),
            block_wc_ridgewood_2:
this.blocks_mainLine.block_wc_ridgewood_2.get_block_status(),

            block_sf_wc_1:
this.blocks_mainLine.block_sf_wc_1.get_block_status(),
            block_sf_wc_2:
this.blocks_mainLine.block_sf_wc_2.get_block_status(),

            block_hilburn_sf:
this.blocks_mainLine.block_hilburn_sf.get_block_status(),
```

```
            block_sterling_sf:
this.blocks_mainLine.block_sterling_sf.get_block_status(),
            block_sterling_hilburn:
this.blocks_mainLine.block_sterling_hilburn.get_block_status(),

            block_hilburn_yard_west:
this.blocks_mainLine.block_hilburn_yard_west.get_block_status(),
            block_hilburn_yard_east:
this.blocks_mainLine.block_hilburn_yard_east.get_block_status(),

            block_wc_yard:
this.blocks_mainLine.block_wc_yard.get_block_status()
        };

        return status;
    }
    // ---- END get_mainLine_blocks_status() ----

    /**
     * get_bergen_blocks_status()
     * @summary Gets the status of all the blocks on the Southern Tier
Section
     *
     * @returns An object with the status of each block
     */
    get_bergen_blocks_status() {
        let status = {
            block_hx_laurel_1:
this.blocks_mainLine.block_hx_laurel_1.get_block_status(),
            block_hx_laurel_2:
this.blocks_mainLine.block_hx_laurel_2.get_block_status(),

            block_pascack_hx_1:
this.blocks_mainLine.block_pascack_hx_1.get_block_status(),
            block_pascack_hx_2:
this.blocks_mainLine.block_pascack_hx_2.get_block_status(),

            block_bt_pascack_1:
this.blocks_mainLine.block_bt_pascack_1.get_block_status(),
            block_bt_pascack_2:
this.blocks_mainLine.block_bt_pascack_2.get_block_status(),

            block_ridgewood_bt_1:
this.blocks_mainLine.block_ridgewood_bt_1.get_block_status(),
            block_ridgewood_bt_2:
this.blocks_mainLine.block_ridgewood_bt_2.get_block_status(),

            block_bt_nysw:
this.blocks_mainLine.block_bt_nysw.get_block_status(),
            block_hx_croxton_1:
```

```
this.blocks_mainLine.block_hx_croxton_1.get_block_status(),
            block_hx_croxton_2:
this.blocks_mainLine.block_hx_croxton_2.get_block_status()
        };

        return status;
    }
    // ---- END get_bergen_block_status() ----

    /**
     * get_tier_block_status()
     * @breif Gets the status of all the blocks on the Southern Tier
Section
     *
     * @returns An object with the status of each block
     */
    get_tier_block_status() {
        let status = {
            // Block Status
            block_harriman_sterling_1:
this.blocks_mainLine.block_harriman_sterling_1.get_block_status(),

            block_valley_harriman_1:
this.blocks_mainLine.block_valley_harriman_1.get_block_status(),
            block_valley_harriman_2:
this.blocks_mainLine.block_valley_harriman_2.get_block_status(),
            block_harriman_industrial:
this.blocks_mainLine.block_harriman_industrial.get_block_status(),

            block_hudson_valley_1:
this.blocks_mainLine.block_hudson_valley_1.get_block_status(),
            block_hudson_nysw:
this.blocks_mainLine.block_hudson_nysw.get_block_status(),

            block_hall_hudson_1:
this.blocks_mainLine.block_hall_hudson_1.get_block_status(),
            block_hall_hudson_2:
this.blocks_mainLine.block_hall_hudson_2.get_block_status(),
            block_hall_yard:
this.blocks_mainLine.block_hall_yard.get_block_status(),

            block_howells_hall_1:
this.blocks_mainLine.block_howells_hall_1.get_block_status(),

            block_ov_howells_1:
this.blocks_mainLine.block_ov_howells_1.get_block_status(),
            block_ov_howells_2:
this.blocks_mainLine.block_ov_howells_2.get_block_status(),

            block_bc_ov_1:
```

```
this.blocks_mainLine.block_bc_ov_1.get_block_status(),

            block_port_bc_1:
this.blocks_mainLine.block_port_bc_1.get_block_status(),
            block_pa_port_1:
this.blocks_mainLine.block_pa_port_1.get_block_status(),
            block_pa_bc_2:
this.blocks_mainLine.block_pa_bc_2.get_block_status(),
            block_port_yard_west:
this.blocks_mainLine.block_port_yard_west.get_block_status(),
            block_port_yard_east:
this.blocks_mainLine.block_port_yard_east.get_block_status(),

            block_buckleys_west:
this.blocks_mainLine.block_buckleys_west.get_block_status(),
            block_buckleys_east:
this.blocks_mainLine.block_buckleys_east.get_block_status(),

            block_sparrow_pa_1:
this.blocks_mainLine.block_sparrow_pa_1.get_block_status(),
            block_sparrow_pa_2:
this.blocks_mainLine.block_sparrow_pa_2.get_block_status(),
            block_sparrow_cripple:
this.blocks_mainLine.block_sparrow_cripple.get_block_status(),

            block_bingo_sparrow:
this.blocks_mainLine.block_bingo_sparrow.get_block_status()
        };

        return status;
    }
    // ---- END get_tier_block_status() ----

    /**
     * get_bergen_symbols()
     * @summary Gets all the symbols for the blocks on the Bergen
County Line Section
     *
     * @returns An obnject with all the block symbols on the Bergen
Line
     */
    get_bergen_symbols() {
        let symbols = {
            symbol_ridgewood_bt_1:
this.blocks_mainLine.block_ridgewood_bt_1.get_symbol(),
            symbol_ridgewood_bt_2:
this.blocks_mainLine.block_ridgewood_bt_2.get_symbol(),
            symbol_bt_pascack_1:
this.blocks_mainLine.block_bt_pascack_1.get_symbol(),
            symbol_bt_pascack_2:
```

```
this.blocks_mainLine.block_bt_pascack_2.get_symbol(),
            symbol_bt_nysw:
this.blocks_mainLine.block_bt_nysw.get_symbol(),
            symbol_pascack_hx_1:
this.blocks_mainLine.block_pascack_hx_1.get_symbol(),
            symbol_pascack_hx_2:
this.blocks_mainLine.block_pascack_hx_2.get_symbol(),
            symbol_hx_laurel_1:
this.blocks_mainLine.block_hx_laurel_1.get_symbol(),
            symbol_hx_laurel_2:
this.blocks_mainLine.block_hx_laurel_2.get_symbol(),
            symbol_hx_croxton_1:
this.blocks_mainLine.block_hx_croxton_1.get_symbol(),
            symbol_hx_croxton_2:
this.blocks_mainLine.block_hx_croxton_2.get_symbol(),
        };

        return symbols;
    }
    // ---- END get_bergen_symbols() ----

    /**
     * get_mainLine_symbol()
     * @summary Gets all the symbols for the blocks on the Main Line
Section
     *
     * @returns An object with all the block symbols on the Main Line
Section
     */
    get_mainLine_symbols() {
        let symbols = {
            // First Row
            symbol_sterling_sf_1:
this.blocks_mainLine.block_sterling_sf.get_symbol(),
            symbol_sterling_hilburn_2:
this.blocks_mainLine.block_sterling_hilburn.get_symbol(),
            symbol_hilburn_sf_2:
this.blocks_mainLine.block_hilburn_sf.get_symbol(),
            symbol_hilburn_yardWest:
this.blocks_mainLine.block_hilburn_yard_west.get_symbol(),
            symbol_hilburn_yardEast:
this.blocks_mainLine.block_hilburn_yard_east.get_symbol(),
            symbol_sf_wc_1:
this.blocks_mainLine.block_sf_wc_1.get_symbol(),
            symbol_sf_wc_2:
this.blocks_mainLine.block_sf_wc_2.get_symbol(),
            symbol_wc_yard:
this.blocks_mainLine.block_wc_yard.get_symbol(),
            symbol_wc_ridgewood_1:
this.blocks_mainLine.block_wc_ridgewood_1.get_symbol(),
```

```
            symbol_wc_ridgewood_2:
this.blocks_mainLine.block_wc_ridgewood_2.get_symbol(),
            symbol_wc_ridgewood_3:
this.blocks_mainLine.block_wc_ridgewood_3.get_symbol(),
            // Second Row
            symbol_ridgewood_suscon_1:
this.blocks_mainLine.block_ridgewood_suscon_1.get_symbol(),
            symbol_ridgewood_suscon_2:
this.blocks_mainLine.block_ridgewood_suscon_2.get_symbol(),
            symbol_suscon_mill_1:
this.blocks_mainLine.block_suscon_mill_1.get_symbol(),
            symbol_suscon_mill_2:
this.blocks_mainLine.block_suscon_mill_2.get_symbol(),
            symbol_mill_westSecaucus_1:
this.blocks_mainLine.block_mill_westSecaucus_1.get_symbol(),
            symbol_mill_westSecaucus_2:
this.blocks_mainLine.block_mill_westSecaucus_2.get_symbol(),
            symbol_westSecaucus_laurel_1:
this.blocks_mainLine.block_westSecaucus_laurel_1.get_symbol(),
            symbol_westSecaucus_laurel_2:
this.blocks_mainLine.block_westSecaucus_laurel_2.get_symbol(),
            symbol_laurel_westEnd_1:
this.blocks_mainLine.block_westEnd_laurel_1.get_symbol(),
            symbol_laurel_westEnd_2:
this.blocks_mainLine.block_westEnd_laurel_2.get_symbol(),
            symbol_laurel_westEnd_3:
this.blocks_mainLine.block_westEnd_laurel_3.get_symbol(),
            symbol_laurel_westEnd_4:
this.blocks_mainLine.block_westEnd_laurel_4.get_symbol(),
        };

        return symbols;
    }
    // ---- END get_mainLine_symbols() ----

    /**
     * get_tier_symbols()
     * @summary Gets all the symbols for the blocks on the Southern
Tier Section
     *
     * @returns An object with all the block symbols on the Southern
Tier Section
     */
    get_tier_symbols() {
        let symbols = {
            // First Row
            symbol_bingo_sparrow:
this.blocks_mainLine.block_bingo_sparrow.get_symbol(),
            symbol_sparrow_pa_1:
this.blocks_mainLine.block_sparrow_pa_1.get_symbol(),
```

```
            symbol_sparrow_pa_2:
this.blocks_mainLine.block_sparrow_pa_2.get_symbol(),
            symbol_pa_port_1:
this.blocks_mainLine.block_pa_port_1.get_symbol(),
            symbol_port_bc_1:
this.blocks_mainLine.block_port_bc_1.get_symbol(),
            symbol_pa_bc_2:
this.blocks_mainLine.block_pa_bc_2.get_symbol(),
            symbol_port_yardEast:
this.blocks_mainLine.block_port_yard_east.get_symbol(),
            symbol_bc_ov:
this.blocks_mainLine.block_bc_ov_1.get_symbol(),
            symbol_ov_howells_1:
this.blocks_mainLine.block_ov_howells_1.get_symbol(),
            symbol_ov_howells_2:
this.blocks_mainLine.block_ov_howells_2.get_symbol(),
            // Second Row
            symbol_howells_hall:
this.blocks_mainLine.block_howells_hall_1.get_symbol(),
            symbol_hall_yard:
this.blocks_mainLine.block_hall_yard.get_symbol(),
            symbol_hall_hudson_1:
this.blocks_mainLine.block_hall_hudson_1.get_symbol(),
            symbol_hall_hudson_2:
this.blocks_mainLine.block_hall_hudson_2.get_symbol(),
            symbol_hudson_valley:
this.blocks_mainLine.block_hudson_valley_1.get_symbol(),
            symbol_hudson_nysw:
this.blocks_mainLine.block_hudson_nysw.get_symbol(),
            symbol_valley_harriman_1:
this.blocks_mainLine.block_valley_harriman_1.get_symbol(),
            symbol_valley_harriman_2:
this.blocks_mainLine.block_valley_harriman_2.get_symbol(),
            // Third Row
            symbol_harriman_sterling:
this.blocks_mainLine.block_harriman_sterling_1.get_symbol(),
            symbol_harriman_industrial:
this.blocks_mainLine.block_harriman_industrial.get_symbol(),
        };

        return symbols;
    }
    // ---- END get_tier_symbols() ----

    /**
     * get_interlocking_route()
     * @summary Takes where a train currently is and gets it's next
route
     *
     * @param key, Is ueed to find the trains curent interlocking
```

```
     * @param direction, which way the train is traveling
     */
    get_interlocking_route(key, direction) {
        let first_index = key.indexOf("_");
        let second_index = key.lastIndexOf("_");
        let track;
        let interlocking;

        if (direction === "WEST") {
            track = key.substr(0, first_index);
            interlocking = key.substr(first_index + 1, second_index -
2);
        }
        else {
            track = key.substr(0, first_index);
            interlocking = key.substr(second_index + 1, key.size);
        }

        // Southern Tier Line
        if (interlocking === "sparrow") {
            return this.get_sparrow().get_train_route(direction,
track);
        }
        if (interlocking === "pa") {
            return this.get_pa().get_train_route(direction, track);
        }
        if (interlocking === "port") {
            return this.get_port().get_train_route(direction, track);
        }
        if (interlocking === "bc") {
            return this.get_bc().get_train_route(direction, track);
        }
        if (interlocking === "ov") {
            return this.get_ov().get_train_route(direction, track);
        }
        if (interlocking === "howells") {
            return this.get_howells().get_train_route(direction,
track);
        }
        if (interlocking === "hall") {
            return this.get_hall().get_train_route(direction, track);
        }
        if (interlocking === "hudson") {
            return this.get_hudson().get_train_route(direction,
track);
        }
        if (interlocking === "valley") {
            return this.get_valley().get_train_route(direction,
track);
        }
```

```
        if (interlocking === "harriman") {
            return this.get_harriman().get_train_route(direction,
track);
        }
        if (interlocking === "sterling") {
            return this.get_sterling().get_train_route(direction,
track);
        }

        // Main Line
        if (interlocking === "hilburn") {
            return this.get_hilburn().get_train_route(direction,
track);
        }
        if (interlocking === "sf") {
            return this.get_sf().get_train_route(direction, track);
        }
        if (interlocking === "wc") {
            return this.get_wc().get_train_route(direction, track);
        }
        if (interlocking === "ridgewood") {
            return this.get_ridgewood().get_train_route(direction,
track);
        }
        if (interlocking === "suscon") {
            return this.get_suscon().get_train_route(direction,
track);
        }
        if (interlocking === "mill") {
            return this.get_mill().get_train_route(direction, track);
        }
        if (interlocking === "westSecaucus") {
            return this.get_westSecaucus().get_train_route(direction,
track);
        }
        if (interlocking === "laurel") {
            return this.get_laurel().get_train_route(direction,
track);
        }

        // Bergen County Line
        if (interlocking === "bt") {
            return this.get_bt().get_train_route(direction, track);
        }
        if (interlocking === "pascack") {
            return this.get_pascack().get_train_route(direction,
track);
        }
        if (interlocking === "hx") {
            return this.get_hx().get_train_route(direction, track);
```

```
        }
    }
    // ---- END get_interlocking_route() ----

    /**
     * set_occupy_interlocking
     * @summary Takes in what interlocking and the track number, and
set that the specific interlocking is occupied on the last track
     *
     * @param track, the track number in the interlocking to occupy,
for some interlocking with only one route doesn't need the track
     * @param name, the name of the interlocking to occupy
     */
    set_occupy_interlocking(track, name) {
        if (name === "hx") {
            if (track === "2") {
                this.get_hx().set_trk_2_occupied(true);
            }
            else {
                this.get_hx().set_trk_1_occupied(true);
            }
        }
        if (name === "pascack") {
            if (track === "1") {
                this.get_pascack().set_trk_1_occupied(true);
            }
            else {
                this.get_pascack().set_trk_2_occupied(true);
            }
        }
        if (name === "bt") {
            if (track === "2") {
                this.get_bt().set_trk_2_occupied(true);
            }
            else {
                this.get_bt().set_trk_1_occupied(true);
            }
        }
        if (name === "laurel") {
            if (track === "1") {
                this.get_laurel().set_trk_1_occupied(true);
            }
            else if (track === "2") {
                this.get_laurel().set_trk_2_occupied(true);
            }
            else if (track === "3") {
                this.get_laurel().set_trk_3_occupied(true);
            }
            else {
                this.get_laurel().set_trk_4_occupied(true);
```

```
    }
}
if (name === "westSecaucus") {
    this.get_westSecaucus().set_occupied(true);
}
if (name === "mill") {
    if (track === "1") {
        this.get_mill().set_trk_1_occupied(true);
    }
    else {
        this.get_mill().set_trk_2_occupied(true);
    }
}
if (name === "suscon") {
    if (track === "1") {
        this.get_suscon().set_trk_1_occupied(true);
    }
    else {
        this.get_suscon().set_trk_2_occupied(true);
    }
}
if (name === "ridgewood") {
    if (track === "1" || track === "4") {
        this.get_ridgewood().set_trk_1_occupied(true);
    }
    else if (track === "2") {
        this.get_ridgewood().set_trk_2_occupied(true);
    }
    else {
        this.get_ridgewood().set_trk_3_occupied(true);
    }
}
if (name === "wc") {
    if (track === "2") {
        this.get_wc().set_trk_2_occupied(true);
    }
    else {
        this.get_wc().set_trk_1_occupied(true);
    }
}
if (name === "sf") {
    if (track === "1") {
        this.get_sf().set_trk_1_occupied(true);
    }
    else {
        this.get_sf().set_trk_2_occupied(true);
    }
}
if (name === "hilburn") {
    this.get_hilburn().set_occupied(true);
```

```
    }

    if (name === "sterling") {
        this.get_sterling().set_occupied(true);
    }
    if (name === "harriman") {
        this.get_harriman().set_occupied(true);
    }
    if (name === "valley") {
        this.get_valley().set_occupied(true);
    }
    if (name === "hudson") {
        this.get_hudson().set_occupied(true);
    }
    if (name === "hall") {
        if (track === "1") {
            this.get_hall().set_trk_1_occupied(true);
        }
        else {
            this.get_hall().set_trk_2_occupied(true);
        }
    }
    if (name === "howells") {
        this.get_howells().set_occupied(true);
    }
    if (name === "ov") {
        this.get_ov().set_occupied(true);
    }
    if (name === "bc") {
        this.get_bc().set_occupied(true);
    }
    if (name === "port") {
        this.get_port().set_occupied(true);
    }
    if (name === "pa") {
        if (track === "1") {
            this.get_pa().set_trk_1_occupied(true);
        }
        else {
            this.get_pa().set_trk_2_occupied(true);
        }
    }
    if (name === "sparrow") {
        this.get_sparrow().set_occupied(true);
    }
}
// ---- END set_occupy_interlocking() ----

/**
 * get_block_by_name()
```

```
     * @summary takes in the name of a block, and returns the
reference to that specific block
     *
     * @param name, the name of the block to find
     *
     * @return reference to the block
     */
    get_block_by_name(name) {
        var block = name.substring(2, name.size);
        var track = name.substring(0, 1);

        if (block === "harriman_sterling") {
            return this.blocks_mainLine.block_harriman_sterling_1;
        }
        else if (block === "valley_harriman") {
            if (track === "1") {
                return this.blocks_mainLine.block_valley_harriman_1;
            }
            else {
                return this.blocks_mainLine.block_valley_harriman_2;
            }
        }
        else if (block === "industrial_harriman") {
            return this.blocks_mainLine.block_harriman_industrial;
        }
        else if (block === "hudson_valley") {
            return this.blocks_mainLine.block_hudson_valley_1;
        }
        else if (block === "hudson_nysw") {
            return this.blocks_mainLine.block_hudson_nysw;
        }
        else if (block === "hall_hudson"){
            if (track === "1") {
                return this.blocks_mainLine.block_hall_hudson_1;
            }
            else {
                return this.blocks_mainLine.block_hall_hudson_2;
            }
        }
        else if (block === "yard_hall") {
            return this.blocks_mainLine.block_hall_yard;
        }
        else if (block === "howells_hall") {
            return this.blocks_mainLine.block_howells_hall_1;
        }
        else if (block === "ov_howells") {
            if (track === "1") {
                return this.blocks_mainLine.block_ov_howells_1;
            }
            else {
```

```
        return this.blocks_mainLine.block_ov_howells_2;
    }
}
else if (block === "bc_ov") {
    return this.blocks_mainLine.block_bc_ov_1;
}
else if (block === "port_bc") {
    return this.blocks_mainLine.block_port_bc_1;
}
else if (block === "pa_port") {
    return this.blocks_mainLine.block_pa_port_1;
}
else if (block === "pa_bc") {
    return this.blocks_mainLine.block_pa_bc_2;
}
else if (block === "port_yardWest") {
    return this.blocks_mainLine.block_port_yard_west;
}
else if (block === "yardEast_port") {
    return this.blocks_mainLine.block_port_yard_east;
}
else if (block === "sparrow_pa") {
    if (track === "1") {
        return this.blocks_mainLine.block_sparrow_pa_1;
    }
    else {
        return this.blocks_mainLine.block_sparrow_pa_2;
    }
}
else if (block === "sparrow_cripple") {
    return this.blocks_mainLine.block_sparrow_cripple;
}
else if (block === "bingo_sparrow") {
    return this.blocks_mainLine.block_bingo_sparrow;
}
else if (block === "laurel_westEnd") {
    if (track === "1") {
        return this.blocks_mainLine.block_westEnd_laurel_1;
    }
    else if (track === "2") {
        return this.blocks_mainLine.block_westEnd_laurel_2;
    }
    else if (track === "3") {
        return this.blocks_mainLine.block_westEnd_laurel_3;
    }
    else if (track === "4") {
        return this.blocks_mainLine.block_westEnd_laurel_4;
    }
}
else if (block === "westSecaucus_laurel") {
```

```
            if (track === "2") {
                return
this.blocks_mainLine.block_westSecaucus_laurel_1;
            }
            else if (track === "4") {
                return
this.blocks_mainLine.block_westSecaucus_laurel_2;
            }
        }
        else if (block === "mill_westSecaucus") {
            if (track === "1") {
                return this.blocks_mainLine.block_mill_westSecaucus_1;
            }
            else if (track === "2") {
                return this.blocks_mainLine.block_mill_westSecaucus_2;
            }
        }
        else if (block === "suscon_mill") {
            if (track === "1") {
                return this.blocks_mainLine.block_suscon_mill_1;
            }
            else if (track === "2") {
                return this.blocks_mainLine.block_suscon_mill_2;
            }
        }
        else if (block === "ridgewood_suscon") {
            if (track === "1") {
                return this.blocks_mainLine.block_ridgewood_suscon_1;
            }
            else if (track === "2") {
                return this.blocks_mainLine.block_ridgewood_suscon_2;
            }
        }
        else if (block === "wc_ridgewood") {
            if (track === "1") {
                return this.blocks_mainLine.block_wc_ridgewood_1;
            }
            else if (track === "2") {
                return this.blocks_mainLine.block_wc_ridgewood_2;
            }
            else if (track === "3") {
                return this.blocks_mainLine.block_wc_ridgewood_3;
            }
        }
        else if (block === "sf_wc") {
            if (track === "1") {
                return this.blocks_mainLine.block_sf_wc_1;
            }
            else if (track === "2") {
                return this.blocks_mainLine.block_sf_wc_2;
```

```
    }
}
else if (block === "sterling_sf") {
    return this.blocks_mainLine.block_sterling_sf;
}
else if (block === "hilburn_sf") {
    return this.blocks_mainLine.block_hilburn_sf;
}
else if (block === "sterling_hilburn") {
    return this.blocks_mainLine.block_sterling_hilburn;
}
else if (block === "hilburn_yardWest") {
    return this.blocks_mainLine.block_hilburn_yard_west;
}
else if (block === "yardHilburn_sf") {
    return this.blocks_mainLine.block_hilburn_yard_east;
}
else if (block === "yard_wc") {
    return this.blocks_mainLine.block_wc_yard;
}
else if (block === "hx_laurel") {
    if (track === "3") {
        return this.blocks_mainLine.block_hx_laurel_1;
    }
    else {
        return this.blocks_mainLine.block_hx_laurel_2;
    }
}
else if (block === "pascack_hx") {
    if (track === "1") {
        return this.blocks_mainLine.block_pascack_hx_1;
    }
    else {
        return this.blocks_mainLine.block_pascack_hx_2;
    }
}
else if (block === "bt_pascack") {
    if (track === "1") {
        return this.blocks_mainLine.block_bt_pascack_1;
    }
    else {
        return this.blocks_mainLine.block_bt_pascack_2;
    }
}
else if (block === "ridgewood_bt") {
    if (track === "1" || track === "3") {
        return this.blocks_mainLine.block_ridgewood_bt_1;
    }
    else {
        return this.blocks_mainLine.block_ridgewood_bt_2;
```

```
            }
        }
        else if (block === "bt_nysw") {
            return this.blocks_mainLine.block_bt_nysw;
        }
        else if (block === "hx_croxton") {
            if (track === "1" || track === "4") {
                return this.blocks_mainLine.block_hx_croxton_1;
            }
            else {
                return this.blocks_mainLine.block_hx_croxton_2;
            }
        }
        else {
            return false;
        }
    }
    // ---- END get_block_by_name() ----
}

export default MainLine_CTC;
```

```
/**
 * @file ctc_block.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary Class that is a "block" or track, that makes up the
railroad
 *
 * @description This class is a section of track, between two
interlockings, this classes make up the railroad
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Route = '#75fa4c';
const Occupied = '#eb3323';

/**
 * Class that is a "block" or track, that makes up part of the
railroad. This class is a section of track,
 * between two interlockings, this classes make up the railroad. The
block class variables that are basically
 * characteristics of a real piece of track
 *
 * @member block_name -> The name of the piece of track, usually the
two location it bridges
 * @member block_size -> The size of the track, (i.e. how long it
takes for a train to travel it)
 * @member block_status -> Wheter the block is Empty, Routed (A train
is coming), or Occupied (A train is there)
 * @member train_symbol -> The symbol or the train that occupys that
block
 */
class CTC_Block {
    /**
     * constructor(),
     * @summary The Constructor of the CTC_Block Class
     *
     * @description Sets all the memeber variables to their initial
values, when the application starts
     *
     * @param p_name, The Name of the Block
     * @param p_size, The Size of the Block
     * @param p_status, Current Status. Only Used for debugging when
build the applications
     */
    constructor(p_name, p_size, p_status) {
        this.block_name = p_name;
        this.block_size = p_size;
        this.block_status = p_status;
        this.train_symbol = null;
```

```
    }
    // ---- END constructor() ----

    /**
     * get_block_status()
     * @summary Getter for the block_status member variable
     *
     * @returns The current status of the block
     */
    get_block_status() {
        return this.block_status;
    }
    // ---- END get_block_status() ----

    /**
     * get_size()
     * @summary Getter for the block_size member variable
     *
     * @return The size of the block
     */
    get_size() {
        return this.block_size;
    }
    // ---- END get_size() ----

    /**
     * get_symbol()
     * @summary Getter for the train_symbol memebr variable
     *
     * @returns The symbol of the trail that is currently in the block
     */
    get_symbol() {
        return this.train_symbol;
    }
    // ---- END get_symbol() ----

    /**
     * reset_block()
     * @summary Resets the Block status to Empty
     *
     * @description This is used to reset the block, when the CTC
controller refreshes the train and route locations
     */
    reset_block() {
        // Check if the Block Is Routed
        if (this.block_status === Route) {
            this.block_status = Empty;
        }
    }
    // ---- END reset_block() ----
```

```
    /**
     * set_symbol()
     * @summary Setter for the train_symbol member variable
     *
     * @param n_symbol, The new symbols to set the member variable too
     */
    set_symbol(n_symbol) {
        this.train_symbol = n_symbol;
    }
    // ---- END set_symbol() ----

    /**
     * set_block_status()
     * @summary Sets the block current status based off of what tag is
passed in
     *
     * @param p_status, A String which is the Kinda of status of what
to set the block too
     */
    set_block_status(p_status) {
        if (p_status === 'Empty') {
            this.block_status = Empty;
        }
        else if (p_status === 'Route') {
            this.block_status = Route;
        }
        else if (p_status === 'Occupied') {
            this.block_status = Occupied;
        }
        else {
            console.log("ERROR!! - CTC_Block " + this.block_name +
" [set_block_status()]");
        }
    }
}

// This is required when using ReactJS
export default CTC_Block;
```

```
/**
 * @file ctc_clock.js
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CTC Controller Class for a Clock for the trains
 */


/**
 * CLASS Clock
 * @brief Class that keeps track of the time since the start of the
application
 *
 * @details This class is used to keep track and calculate how much
time has passed since the launch
 * of the program, it is used to keep the trains moving at the correct
times
 *
 * MEMBER VARIABLES
 * start_time -> The the games was started
 */
class Clock {
    /**
     * constructor()
     * @brief The constructor for the Clock class
     *
     * @details This will initialize all the member variables when the
program is started
     */
    constructor() {
        this.start_time;
    }
    // ---- END constructor() ----

    /**
     * startClock()
     * @brief Intialize the start time variable
     */
    startClock() {
        this.start_time = new Date().getTime() / 1000;
    }
    // ---- END startClock() ----

    /**
     * getTimeFromStart()
     * @brief Calculated how long it's been since the start of the
program in seconds
     *
     * @returns The number of seconds since the program was started
     */
```

```
    getTimeFromStart = () => {
        var current_time = new Date().getTime() / 1000;
        var diff = current_time - this.start_time;

        return diff;
    }
    // ---- END getTimeFromStart() ----
}

// This is required when using ReactJS
export default Clock;
```

```
/**
 * @file ctc_train.js
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CTC Controller Class for a Clock for the trains
 */

// Import the Custom Clock Class
import Clock from '../Trains/clock.js';

/**
 * CLASS Train
 * @brief Class that keeps track of the time since the start of the
application
 *
 * @details This class is used to keep track and calculate how much
time has passed since the launch
 * of the program, it is used to keep the trains moving at the correct
times
 *
 * MEMBER VARIABLES
 * start_time -> The the games was started
 */
class Train {
    /**
     * constructor()
     * @brief The constructor for the Train class
     *
     * @details This will initialize all the member variables when the
program is started
     *
     * @param p_symbol -> The Train's Symbol
     * @param p_location -> The Trains Inital Location
     * @param p_direction -> The Direction the train is traveling
     * @param p_block_size -> The size of the trains inital block
     */
    constructor(p_symbol, p_location, p_direction, p_block_size) {
        this.clock = new Clock();
        this.clock.startClock();

        this.symbol = p_symbol;
        this.current_location = p_location;
        this.direction = p_direction;
        this.block_size = p_block_size;
        this.block_start = this.clock.getTimeFromStart();

        this.route = true;
    }
    // ---- END constructor() ----
```

```
    /**
     * get_symbol()
     * @brief Getter for the trains symbol
     *
     * @returns The train symbol
     */
    get_symbol() {
        return this.symbol;
    }
    // ---- END get_symbol() ----

    /**
     * update_location()
     * @brief Take in a new location and sets it for the train
     */
    update_location(new_next_location) {
        this.current_location = new_next_location;
        this.block_start = this.clock.getTimeFromStart();
    }
    // ---- END update_location() ----

    /**
     * can_update_location()
     * @brief Determines if the train can move to the next location
     */
    can_update_location() {
        // If The train has a route
        if (this.route) {
            // Check if the train has spent enough time in the curent
block
            if (this.clock.getTimeFromStart() - this.block_start >
this.block_size) {
                return true;
            }
            else {
                return false;
            }
        }
    }
    // ---- END can_update_location() ----

    /**
     * get_location()
     * @brief Getter for the current_location variable
     */
    get_location() {
        return this.current_location;
    }
    // ---- END get_location() ----
```

```
    /**
     * get_block_size()
     * @brief Getter for the block_size variable
     */
    get_block_size() {
        return this.block_size;
    }
    // ---- END get_block_size() ----

    /**
     * set_block_size()
     * @brief Takes in the new block size, and sets the member
variable
     * @param n_size, the new size of the next block
     */
    set_block_size(n_size) {
        this.block_size = n_size;
    }
    // ---- END set_block_size() ----

    /**
     * get_direction()
     * @brief Getter for the direction member variable
     */
    get_direction() {
        return this.direction;
    }
    // ---- END get_direction() ----

    /**
     * get_route()
     * @brief Getter for the route member variable
     */
    get_route() {
        return this.route;
    }
    // ---- END get_route() ----

    /**
     * set_route()
     * @brief Takes in the next route and sets the member variable
     * @param n_route, the trains new route
     */
    set_route(n_route) {
        this.route = n_route;
    }
    // ---- END set_route() ----
}

// Export the panel to be drawn on the screen
```

```
export default Train;
```

```
/**
 * @file ctc_bt.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the BT Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the BT Interlocking This class is what
controlls the BT Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 * @member sw_5 -> Bool if Switch #5 is Reveresed or Not
 *
 * @member sig_2w1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_BT {
    /**
     * constructor()
```

```
     * @summary The constructor for the CTC_BT class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        this.sw_5 = false;
        // Bools for the signals
        this.sig_2w1 = false;
        this.sig_2w2 = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_w_trk_3 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
        this.trk_1_occupied = false;
        this.trk_2_occupied = false;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else if (track === "2") {
                return this.route_w_trk_2;
            }
            else {
                return this.route_w_trk_3;
            }
```

```
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w1(next_block_1, next_block_2) {
        // Checks if Any Switches are Against the signal
        if (this.sw_5 || this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2w1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__3_ridgewood_bt";
                this.routed_trk_1 = true;
                this.sig_2w1 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_2w1) {
```

```
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w1 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__4_ridgewood_bt";
                this.routed_trk_1 = true;
                this.sig_2w1 = true;
            }
        }
    }
    // ---- END click_sig_2w1() ----

    /**
     * click_sig_2w2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w2(next_block_1, next_block_2) {
        // Checks if Any Switches are Against the signal
        if (!this.sw_5 || this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2w2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
```

```
                this.route_w_trk_3 = "W_3_1__|__3_ridgewood_bt";
                this.routed_trk_1 = true;
                this.sig_2w2 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_2w2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w2 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_2__|__4_ridgewood_bt";
                this.routed_trk_1 = true;
                this.sig_2w2 = true;
            }
        }
    }
    // ---- END click_sig_2w2() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4w(next_block_1, next_block_2) {
        // Checks if Any Switches are Against the signal
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__4_ridgewood_bt";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (this.sw_1) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__3_ridgewood_bt";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2e(next_block_1, next_block_2, next_block_3) {
        // Checks if Any Switches are Against the signal
        if (this.sw_3) {
```

```
            return;
        }
        else if (!this.sw_1 && !this.sw_5) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_bt_pascack";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else if (!this.sw_1 && this.sw_5) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__3_bt_nysw";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else if (this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
```

```
                }
                this.route_e_trk_1 = "E_1_2__|__2_bt_pascack";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4e(next_block_1, next_block_2, next_block_3) {
        // Checks if Any Switches are Against the signal
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_bt_pascack";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_3 && !this.sw_5) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
```

```
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_bt_pascack";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_3 && this.sw_5) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__3_bt_nysw";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            // Set the track #1 as Occupied
            this.trk_1_occupied = n_state;
            // Remove the route from track #1
            this.routed_trk_1 = false;
            // Set the time track #1 was occupied
```

```
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            // Set the track #2 as Occupied
            this.trk_2_occupied = n_state;
            // Remove the route from track #2
            this.routed_trk_2 = false;
            // Set the time track #2 was occupied
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;

        // Checking Track 1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            // Clear Track 1
            this.sig_2w1 = false;
            this.sig_2w2 = false;
```

```
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.route_w_trk_3 = null;
            this.routed_trk_1 = false;

            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
        // Checking Track 2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            // Clear Track 2
            this.sig_4w = false;
            this.sig_4e = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * throw_sw_1()
     * @summary Changes the current state of switch #1, used when user
clicks the switch
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * throw_sw_3()
     * @summary Changes the current state of switch #3, used when user
clicks the switch
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
```

```
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * throw_sw_5()
     * @summary Changes the current state of switch #5, used when user
clicks the switch
     */
    throw_sw_5() {
        if (this.sw_5 === false) {
            this.sw_5 = true;
        }
        else {
            this.sw_5 = false;
        }
    }
    // ---- END throw_sw_5() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
this.route_w_trk_3,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
```

```
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        var status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_5: this.sw_5,

            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied,
            routed_1: this.routed_trk_1,
            routed_2: this.routed_trk_2,
            routes: this.get_routes()
        };

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_BT;
```

```
/**
 * @file ctc_hx.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the HX Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the HX Interlocking This class is what
controlls the HX Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 * @member sw_5 -> Bool if Switch #5 is Reveresed or Not
 *
 * @member sig_2w1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_2w3 -> Bool if Signal #2w-3 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_HX {
    /**
     * constructor()
```

```
     * @summary The constructor for the CTC_BT class
     *
     * @discription This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        this.sw_5 = false;
        // Bools for the signals
        this.sig_2w1 = false;
        this.sig_2w2 = false;
        this.sig_2w3 = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
        this.trk_1_occupied = false;
        this.trk_2_occupied = false;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "2") {
                return this.route_w_trk_2;
            }
            else {
                return this.route_w_trk_1;
            }
        }
        else {
            if (track === "1") {
```

```
                return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @discription When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w1(next_block_1, next_block_2) {
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2w1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2w1 = true;
            }
        }
        else {
            if (this.sig_2w1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w1 = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2w1 = true;
            }
        }
    }
    // ---- END click_sig_2w1() ----

    /**
     * click_sig_2w2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @discription When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w2(next_block_1, next_block_2) {
        if (!this.sw_3 || this.sw_5) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2w2) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_3_1__|__1_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2w2 = true;
            }
        }
```

```
        else {
            if (this.sig_2w2) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w2 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_3_2__|__2_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2w2 = true;
            }
        }
    }
    // ---- END click_sig_2w2() ----

    /**
     * click_sig_2w3()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @discription When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w3(next_block_1, next_block_2) {
        if (!this.sw_3 || !this.sw_5) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2w3) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w3 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
```

```
                }
                this.route_w_trk_1 = "W_4_1__|__1_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2w3 = true;
            }
        }
        else {
            if (this.sig_2w3) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w3 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_4_2__|__2_pascack_hx";
                this.routed_trk_1 =
SVGComponentTransferFunctionElement;
                this.sig_2w3 = true;
            }
        }
    }
    // ---- END click_sig_2w3() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @discription When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4w(next_block_2) {
        if (this.sw_1) {
            return;
        }
        else {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_pascack_hx";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @discription When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_2e(next_block_1, next_block_3, next_block_4) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__3_hx_laurel";
                this.routed_trk_1 = true;
                this.sig_2e = true;
```

```
            }
        }
        else if (this.sw_3 && !this.sw_5) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__3_hx_croxton";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else if (this.sw_3 && this.sw_5) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_4__|__4_hx_croxton";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @discription When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
```

```
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_4e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (!this.sw_1) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__1_hx_laurel";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_1 && !this.sw_3) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__3_hx_laurel";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_1 && this.sw_3 && !this.sw_5) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
```

```
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__3_hx_croxton";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_1 && this.sw_3 && this.sw_5) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_4__|__4_hx_croxton";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
```

```
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.trk_2_occupied = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @discription Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;

        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w1 = false;
            this.sig_2w2 = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
```

```
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * throw_sw_1()
     * @summary Changes the current state of switch #1, used when user
clicks the switch
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * throw_sw_3()
     * @summary Changes the current state of switch #3, used when user
clicks the switch
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * throw_sw_5()
     * @summary Changes the current state of switch #5, used when user
clicks the switch
```

```
     */
    throw_sw_5() {
        if (this.sw_5 === false) {
            this.sw_5 = true;
        }
        else {
            this.sw_5 = false;
        }
    }
    // ---- END throw_sw_5() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @discription All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        var status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_5: this.sw_5,

            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied,
            routed_1: this.routed_trk_1,
            routed_2: this.routed_trk_2,
```

```
            routes: this.get_routes()
        };

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_HX;
```

```
/**
 * @file ctc_pascack.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the Pascack Junction Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the Pascack Junction Interlocking This
class is what controlls the Pascack Junction Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible, and the ReactJS Component
class
 * gets information from this class to display the correct status of
the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_Pascack {
    /**
     * constructor()
     * @summary The constructor for the CTC_BT class
     *
```

```
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
        this.trk_1_occupied = false;
        this.trk_2_occupied = false;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else {
                return this.route_w_trk_2;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
```

```
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w(next_block_1, next_block_2) {
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_bt_pascack";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
        else {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
```

```
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_bt_pascack";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4w(next_block_1, next_block_2) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_bt_pascack";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
```

```
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_bt_pascack";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2e(next_block_1, next_block_2) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
```

```
        }
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_pascack_hx";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4e(next_block_1, next_block_2) {
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
```

```
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_pascack_hx";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_pascack_hx";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
parameмter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
```

```
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.trk_2_occupied = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;

        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w1 = false;
            this.sig_2w2 = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e = false;
```

```
            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * throw_sw_1()
     * @summary Changes the current state of switch #1, used when user
clicks the switch
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * throw_sw_3()
     * @summary Changes the current state of switch #3, used when user
clicks the switch
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
```

```
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        var status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,

            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied,
            routed_1: this.routed_trk_1,
            routed_2: this.routed_trk_2,
            routes: this.get_routes()
        };

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Pascack;
```

```
/**
 * @file ctc_hilburn.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the Hilburn Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the Hilburn Interlocking This class is
what controlls the Hilburn Interlocking, it is sort of like a backen,
but is
 * the controller, this is what makes all the train movements
possible, and the ReactJS Component class
 * gets information from this class to display the correct status of
the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w_1 -> Bool if Signal #2w_1 is Lined or Not
 * @member sig_2w_2 -> Bool if Signal #2w_2 is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Hilburn {
    /**
     * constructor()
     * @summary The constructor for the CTC_Hilburn class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        // Bools for the signals
        this.sig_2w_1 = false;
```

```
        this.sig_2w_2 = false;
        this.sig_2e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "2") {
                return this.route_w_trk_1;
            }
            else {
                return this.route_w_trk_2;
            }
        }
        else {
            return this.route_e_trk_1;
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w_1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w_1(next_block_1) {
        if (this.sw_1) {
            return;
        }
```

```
        else {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__2_sterling_hilburn";
                this.sig_2w_1 = true;
            }
        }
    }
    // ---- END click_sig_2w_1() ----

    /**
     * click_sig_2w_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w_2(next_block_1) {
        if (!this.sw_1) {
            return;
        }
        else {
            if (this.sig_2w_2) {
                this.route_w_trk_2 = null;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__2_sterling_hilburn";
                this.sig_2w_2 = true;
            }
```

```
        }
    }
    // ---- END click_sig_2w_2() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2e(next_block_1, next_block_2) {
        if (!this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__2_hilburn_sf";
                this.sig_2e = true;
            }
        }
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__0_hilburn_yardWest";
                this.sig_2e = true;
            }
        }
```

```
    }
    // ---- END click_sig_2e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true || n_state === false) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w_1 = false;
            this.sig_2w_2 = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_w_trk_2 = null;
            this.route_e_trk_1 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----
```

```
    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            occupied: this.int_occupied,
```

```
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Hilburn;
```

```
/**
 * @file ctc_laurel.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the Laurel Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the Laurel Interlocking This class is what
controlls the Laurel Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 * @member sw_7 -> Bool if Switch #7 is Reveresed or Not
 * @member sw_9 -> Bool if Switch #9 is Reveresed or Not
 * @member sw_11 -> Bool if Switch #11 is Reveresed or Not
 * @member sw_13 -> Bool if Switch #13 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_8w -> Bool if Signal #8w is Lined or Not
 * @member sig_10w -> Bool if Signal #10w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 * @member sig_8e -> Bool if Signal #8e is Lined or Not
 * @member sig_12e -> Bool if Signal #12e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_w_trk_4 = The west bound route for track #4
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 * @member route_e_trk_3 = The east bound route for track #3
 * @member route_e_trk_4 = The east bound route for track #4
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
```

```
 * @member routed_trk_3 = Bool if track #3 is routed or not
 * @member routed_trk_4 = Bool if track #4 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_3_time = The time track #3 was occupied, used to know
when to clear the route
 * @member trk_4_time = The time track #4 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 * @member trk_3_occupied = Bool if track #3 is occupied or not
 * @member trk_4_occupied = Bool if track #4 is occupied or not
 */
class CTC_Laurel {
    /**
     * constructor()
     * @summary The constructor for the CTC_Laurel class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        this.sw_7 = false;
        this.sw_9 = false;
        this.sw_11 = false;
        this.sw_13 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_4w = false;
        this.sig_8w = false;
        this.sig_10w = false;
        this.sig_6e = false;
        this.sig_12e = false;
        this.sig_8e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_3 = null;
        this.route_w_trk_4 = null;
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_3 = null;
        this.route_e_trk_4 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
```

```
    this.routed_trk_1 = false;
    this.routed_trk_2 = false;
    this.routed_trk_3 = false;
    this.routed_trk_4 = false;
    this.occupied_trk_1 = false;
    this.occupied_trk_2 = false;
    this.occupied_trk_3 = false;
    this.occupied_trk_4 = false;
    this.trk_1_time = null;
    this.trk_2_time = null;
    this.trk_3_time = null;
    this.trk_4_time = null;
}
// ---- END constructor() ----

/**
 * get_train_route()
 * @summary Returns the route for the train at a given track
 *
 * @param direction, The direction the train is moving
 * @param track, The Track number of the train
 */
get_train_route(direction, track) {
    if (direction === "WEST") {
        if (track === "1") {
            return this.route_w_trk_1;
        }
        else if (track === "2") {
            return this.route_w_trk_2;
        }
        else if (track === "3") {
            return this.route_w_trk_3;
        }
        else {
            return this.route_w_trk_4;
        }
    }
    else {
        if (track === "1") {
            return this.route_e_trk_1;
        }
        else if (track === "2") {
            return this.route_e_trk_2;
        }
        else if (track === "3") {
            return this.route_e_trk_3;
        }
        else {
            return this.route_e_trk_4;
        }
```

```
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2w(next_block_1, next_block_2, next_block_3) {
        if (this.sw_11 || this.sw_1) {
            return;
        }
        else if (!this.sw_7 && !this.sw_3) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__2_hx_laurel";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
        else if (!this.sw_7 && this.sw_3) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
```

```
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_3__|__3_hx_laurel";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
        else if (this.sw_7) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
                return;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|
__2_westSecaucus_laurel";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4w(next_block_1, next_block_2, next_block_3) {
        if (this.sw_13 || this.sw_7) {
            return;
        }
```

```
        else if (!this.sw_1) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|
__2_westSecaucus_laurel";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (this.sw_1 && !this.sw_3) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__2_hx_laurel";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (this.sw_1 && this.sw_3) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
```

```
                this.route_w_trk_2 = "W_2_3__|__3_hx_laurel";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_8w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_8w(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (!this.sw_13) {
            if (this.sig_8w) {
                this.route_w_trk_4 = null;
                this.routed_trk_4 = false;
                this.sig_8w = false;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_4 = "W_4_4__|
__4_westSecaucus_laurel";
                this.routed_trk_4 = true;
                this.sig_8w = true;
            }
        }
        else if (this.sw_13 && !this.sw_7 && !this.sw_1) {
            if (this.sig_8w) {
                this.route_w_trk_4 = null;
                this.routed_trk_4 = false;
                this.sig_8w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_4 = "W_4_2__|
__2_westSecaucus_laurel";
                this.routed_trk_4 = true;
                this.sig_8w = true;
            }
        }
        else if (this.sw_13 && !this.sw_7 && this.sw_1 && !this.sw_3)
{
            if (this.sig_8w) {
                this.route_w_trk_4 = null;
                this.routed_trk_4 = false;
                this.sig_8w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_4 = "W_4_1__|__2_hx_laurel";
                this.routed_trk_4 = true;
                this.sig_8w = true;
            }
        }
        else if (this.sw_13 && !this.sw_7 && this.sw_1 && this.sw_3) {
            if (this.sig_8w) {
                this.route_w_trk_4 = null;
                this.routed_trk_4 = false;
                this.sig_8w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_4 = "W_4_3__|__3_hx_laurel";
                this.routed_trk_4 = true;
                this.sig_8w = true;
            }
        }
```

```
    }
    // ---- END click_sig_8w() ----

    /**
     * click_sig_10w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_10w(next_block_1, next_block_2, next_block_3) {
        if (!this.sw_11 && !this.sw_3) {
            if (this.sig_10w) {
                this.route_w_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_10w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_3__|__3_hx_laurel";
                this.routed_trk_3 = true;
                this.sig_10w = true;
            }
        }
        else if (this.sw_11 && !this.sw_7 && !this.sw_3 && !this.sw_1)
{
            if (this.sig_10w) {
                this.route_w_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_10w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
```

```
                this.route_w_trk_3 = "W_3_1__|__1_hx_laurel";
                this.routed_trk_3 = true;
                this.sig_10w = true;
            }
        }
        else if (this.sw_11 && this.sw_7 && !this.sw_1) {
            if (this.sig_10w) {
                this.route_w_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_10w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_2__|
__2_westSecaucus_laurel";
                this.routed_trk_3 = true;
                this.sig_10w = true;
            }
        }
    }
    // ---- END click_sig_10w() ----

    /**
     * click_sig_6e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_6e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (!this.sw_3 && !this.sw_11) {
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
            }
```

```
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_3__|__3_laurel_westEnd";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
        else if (this.sw_3 && !this.sw_1 && !this.sw_7) {
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_1__|__1_laurel_westEnd";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
        else if (this.sw_3 && this.sw_1 && !this.sw_7 && !this.sw_13)
{
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_2__|__2_laurel_westEnd";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
        else if (this.sw_3 && this.sw_1 && !this.sw_7 && this.sw_13) {
```

```
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_4__|__4_laurel_westEnd";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
    }
    // ---- END click_sig_6e() ----

    /**
     * click_sig_12e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_12e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (this.sw_3 || this.sw_7) {
            return;
        }
        else if (!this.sw_1 && !this.sw_11) {
            if (this.sig_12e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_12e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
```

```
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_laurel_westEnd";
                this.routed_trk_1 = true;
                this.sig_12e = true;
            }
        }
        else if (!this.sw_1 && this.sw_11) {
            if (this.sig_12e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_12e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__3_laurel_westEnd";
                this.routed_trk_1 = true;
                this.sig_12e = true;
            }
        }
        else if (this.sw_1 && !this.sw_13) {
            if (this.sig_12e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_12e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_laurel_westEnd";
                this.routed_trk_1 = true;
                this.sig_12e = true;
            }
        }
        else if (this.sw_1 && this.sw_13) {
            if (this.sig_12e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_12e = false;
            }
```

```
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_4__|__4_laurel_westEnd";
                this.routed_trk_1 = true;
                this.sig_12e = true;
            }
        }
    }
    // ---- END click_sig_12e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_4e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_7 && !this.sw_13) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_laurel_westEnd";
                this.routed_trk_2 = true;
```

```
                this.sig_4e = true;
            }
        }
        else if (!this.sw_7 && this.sw_13) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_4__|__4_laurel_westEnd";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_7 && !this.sw_11) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_laurel_westEnd";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_7 && this.sw_11) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
```

```
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__3_laurel_westEnd";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * click_sig_8e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_4, The next block on Track #4
     */
    click_sig_8e(next_block_4) {
        if (this.sw_13) {
            return;
        }
        else {
            if (this.sig_8e) {
                this.route_e_trk_4 = null;
                this.routed_trk_4 = false;
                this.sig_8e = false;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_4 = "E_4_4__|__4_laurel_westEnd";
                this.routed_trk_4 = true;
                this.sig_8e = true;
            }
        }
    }
    // ---- END click_sig_8e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
```

```
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_1 = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_2 = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * set_trk_3_occupied()
     * @summary Sets track #3 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_3_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_3 = n_state;
            this.routed_trk_3 = false;
            this.trk_3_time = new Date().getTime() / 1000;
```

```
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_3_occupied() ----

    /**
     * set_trk_4_occupied()
     * @summary Sets track #4 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_4_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_4 = n_state;
            this.routed_trk_4 = false;
            this.trk_4_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_4_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;
        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
            this.sig_12e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.occupied_trk_1 = false;
```

```
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.occupied_trk_2 = false;
            this.trk_2_time = null;
        }
        // Track #3
        if (current_time - this.trk_3_time > 4 && current_time -
this.trk_3_time< 100000) {
            this.sig_10w = false;
            this.sig_6e = false;

            this.route_w_trk_3 = null;
            this.route_e_trk_3 = null;
            this.routed_trk_3 = false;

            this.occupied_trk_3 = false;
            this.trk_3_time = null;
        }
        // Track #4
        if (current_time - this.trk_4_time > 4 && current_time -
this.trk_4_time< 100000) {
            this.sig_8w = false;
            this.sig_8e = false;

            this.route_w_trk_4 = null;
            this.route_e_trk_4 = null;
            this.routed_trk_4 = false;

            this.occupied_trk_4 = false;
            this.trk_4_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
```

```
     */
    get_routes() {
        let routes = [
            this.route_e_trk_4, this.route_e_trk_3,
            this.route_e_trk_1, this.route_e_trk_2,
            this.route_w_trk_4, this.route_w_trk_3,
            this.route_w_trk_2, this.route_w_trk_1,
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * @summary Function to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * @summary Funtion to throw switch #7 in the interlocking
     *
```

```
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_7() {
        if (this.sw_7 === false) {
            this.sw_7 = true;
        }
        else {
            this.sw_7 = false;
        }
    }
    // ---- END throw_sw_7() ----

    /**
     * @summary Funtion to throw switch #9 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_9() {
        if (this.sw_9 === false) {
            this.sw_9 = true;
        }
        else {
            this.sw_9 = false;
        }
    }
    // ---- END throw_sw_9() ----

    /**
     * @summary Funtion to throw switch #11 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_11() {
        if (this.sw_11 === false) {
            this.sw_11 = true;
        }
        else {
            this.sw_11 = false;
        }
    }
    // ---- END throw_sw_11() ----

    /**
     * @summary Funtion to throw switch #13 in the interlocking
```

```
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_13() {
        if (this.sw_13 === false) {
            this.sw_13 = true;
        }
        else {
            this.sw_13 = false;
        }
    }
    // ---- END throw_sw_13() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_7: this.sw_7,
            sw_9: this.sw_9,
            sw_11: this.sw_11,
            sw_13: this.sw_13,
            routed_1: this.routed_trk_1,
            routed_2: this.routed_trk_2,
            routed_3: this.routed_trk_3,
            routed_4: this.routed_trk_4,
            occupied_1: this.occupied_trk_1,
            occupied_2: this.occupied_trk_2,
            occupied_3: this.occupied_trk_3,
            occupied_4: this.occupied_trk_4,
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}
```

```
// This is required when using ReactJS
export default CTC_Laurel;
```

```
/**
 * @file ctc_mill.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the Mill Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the Mill Interlocking This class is what
controlls the Mill Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_Mill {
    /**
     * constructor()
     * @summary The constructor for the CTC_Mill class
```

```
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        this.cross_over = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_2e = false;
        this.sig_4w = false;
        this.sig_4e = false;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.occupied_trk_1 = false;
        this.occupied_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
    }
    // ---- END constructor() ----

    /**
     * click_sig()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param sigNum, The number of the signal clicked
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig(sigNum, next_block_1, next_block_2) {
        if (sigNum === "2W") {
            if (this.sw_3) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_2w) {
```

```
                    this.route_w_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2w = false;
                    return;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_1 = "W_1_1__|__1_suscon_mill";
                    this.routed_trk_1 = true;
                    this.sig_2w = true;
                }
            }
            else if (this.sw_1 && !this.sw_3){
                if (this.sig_2w) {
                    this.route_w_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2w = false;
                    return;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_1 = "W_1_2__|__2_suscon_mill";
                    this.routed_trk_1 = true;
                    this.sig_2w = true;
                }
            }
        }
        else if (sigNum === "4W") {
            if (this.sw_1) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_4w) {
                    this.route_w_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4w = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
```

```
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_2 = "W_2_2__|__2_suscon_mill";
                    this.routed_trk_2 = true;
                    this.sig_4w = true;
                }
            }
            else if (!this.sw_1 && this.sw_3) {
                if (this.sig_4w) {
                    this.route_w_trk_2 = null;
                    this.routed_trk_2 = true;
                    this.sig_4w = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_2 = "W_2_1__|__1_suscon_mill";
                    this.routed_trk_2 = true;
                    this.sig_4w = true;
                }
            }
        }
        else if (sigNum === "2E") {
            if (this.sw_1) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_2e) {
                    this.route_e_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2e = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_1 = "E_1_1__|
__1_mill_westSecaucus";
                    this.routed_trk_1 = true;
                    this.sig_2e = true;
                }
```

```
            }
            else if (!this.sw_1 && this.sw_3) {
                if (this.sig_2e) {
                    this.route_e_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2e = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_1 = "E_1_2__|
__2_mill_westSecaucus";
                    this.routed_trk_1 = true;
                    this.sig_2e = true;
                }
            }
        }
        else if (sigNum === "4E") {
            if (this.sw_3) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_4e) {
                    this.route_e_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4e = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_2 = "E_2_2__|
__2_mill_westSecaucus";
                    this.routed_trk_2 = true;
                    this.sig_4e = true;
                }
            }
            else if (this.sw_1 && !this.sw_3) {
                if (this.sig_4e) {
                    this.route_e_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4e = false;
                }
```

```
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_2 = "E_2_1__|
__1_mill_westSecaucus";
                    this.routed_trk_2 = true;
                    this.sig_4e = true;
                }
            }
        }
    }
    // ---- END click_sig() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_1 = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_2 = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
```

```
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;

        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.occupied_trk_1 = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e_1 = false;
            this.sig_4e_2 = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.route_e_trk_3 = null;
            this.routed_trk_2 = false;

            this.occupied_trk_2 = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----
```

```
    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [this.route_w_trk_1, this.route_w_trk_2,
this.route_e_trk_1, this.route_e_trk_2];
        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else {
                return this.route_w_trk_2;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * throw_sw_1()
     * @summary Changes the current state of switch #1, used when user
clicks the switch
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
```

```
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * throw_sw_3()
     * @summary Changes the current state of switch #3, used when user
clicks the switch
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        var status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            occupied_trk_1: this.occupied_trk_1,
            occupied_trk_2: this.occupied_trk_2,
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            routes: this.get_routes()
        };

        return status;
    }
}

// This is required when using ReactJS
```

```
export default CTC_Mill;
```

```
/**
 * @file ctc_ridgewood.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the Ridgewood Junction
Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the Ridgewood Junction Interlocking This
class is what controlls the Ridgewood Junction Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible, and the ReactJS Component
 * class gets information from this class to display the correct
status of the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 * @member sw_5 -> Bool if Switch #5 is Reveresed or Not
 * @member sw_7 -> Bool if Switch #7 is Reveresed or Not
 * @member sw_9 -> Bool if Switch #9 is Reveresed or Not
 *
 * @member sig_2w_1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w_2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_6w -> Bool if Signal #6w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 * @member sig_6e -> Bool if Signal #6e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_w_trk_4 = The west bound route for track #4
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 * @member route_e_trk_3 = The east bound route for track #3
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member routed_trk_3 = Bool if track #3 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
```

```
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_3_time = The time track #3 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 * @member trk_3_occupied = Bool if track #3 is occupied or not
 */
class CTC_Ridgewood {
    /**
     * constructor()
     * @summary The constructor for the CTC_Ridgewood class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        this.sw_5 = false;
        this.sw_7 = false;
        this.sw_9 = false;
        // Bools for the signals
        this.sig_2w_1 = false;
        this.sig_2w_2 = false;
        this.sig_4w = false;
        this.sig_6w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        this.sig_6e = false;
        // Track routes
        this.route_w_trk_3 = null;
        this.route_w_trk_4 = null;
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_3 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.routed_trk_3 = false;
        this.occupied_trk_1 = false;
        this.occupied_trk_2 = false;
        this.occupied_trk_3 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
        this.trk_3_time = null;
    }
```

```
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else if (track === "2") {
                return this.route_w_trk_2;
            }
            else if (track === "3") {
                return this.route_w_trk_3;
            }
            else {
                return this.route_w_trk_4;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else if (track === "2") {
                return this.route_e_trk_2;
            }
            else {
                return this.route_e_trk_3;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w_1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
```

```
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2w1(next_block_1, next_block_2, next_block_3) {
        if (this.sw_3 || this.sw_7 || this.sw_9) {
            return;
        }
        else if (!this.sw_1 && !this.sw_5) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
        else if (this.sw_1 && !this.sw_5) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
                return;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_3__|__3_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
        else if (!this.sw_1 && this.sw_5) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
                return;
```

```
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
    }
    // ---- END click_sig_2w_1() ----

    /**
     * click_sig_2w_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2w2(next_block_1, next_block_2, next_block_3) {
        if (this.sw_3 || this.sw_7) {
            return;
        }
        if (this.sw_9) {
            if (!this.sw_1 && !this.sw_5) {
                if (this.sig_2w_2) {
                    this.route_w_trk_4 = null;
                    this.routed_trk_1 = false;
                    this.sig_2w_2 = false;
                    return;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_4 = "W_4_1__|__1_wc_ridgewood";
```

```
                    this.routed_trk_1 = true;
                    this.sig_2w_2 = true;
                }
            }
            else if (this.sw_1 && !this.sw_5) {
                if (this.sig_2w_2) {
                    this.route_w_trk_4 = null;
                    this.routed_trk_1 = false;
                    this.sig_2w_2 = false;
                    return;
                }
                else {
                    if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_4 = "W_4_3__|__3_wc_ridgewood";
                    this.routed_trk_1 = true;
                    this.sig_2w_2 = true;
                }
            }
            else if (!this.sw_1 && this.sw_5) {
                if (this.sig_2w_2) {
                    this.route_w_trk_4 = null;
                    this.routed_trk_1 = false;
                    this.sig_2w_2 = false;
                    return;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_4 = "W_4_2__|__2_wc_ridgewood";
                    this.routed_trk_1 = true;
                    this.sig_2w_2 = true;
                }
            }
        }
    }
    // ---- END click_sig_2w_2() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
```

```
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4w(next_block_1, next_block_2, next_block_3) {
        if (this.sw_5) {
            return;
        }
        if (!this.sw_3) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
                return;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_wc_ridgewood";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (!this.sw_1 && this.sw_3) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_wc_ridgewood";
                this.routed_trk_2 = true;
                this.sig_4w = true;
```

```
            }
        }
        else if (this.sw_1 && this.sw_3) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
                return;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_3__|__3_wc_ridgewood";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_6w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_6w(next_block_1, next_block_2, next_block_3) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_7) {
            if (this.sig_6w) {
                this.route_w_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6w = false;
                return;
            }
            else {
```

```
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_3__|__3_wc_ridgewood";
                this.routed_trk_3 = true;
                this.sig_6w = true;
            }
        }
        else if (this.sw_7 && !this.sw_5 && !this.sw_3) {
            if (this.sig_6w) {
                this.route_w_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6w = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_1__|__1_wc_ridgewood";
                this.routed_trk_3 = true;
                this.sig_6w = true;
            }
        }
        else if (this.sw_7 && this.sw_5 && !this.sw_3) {
            if (this.sig_6w) {
                this.route_w_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6w = false;
                return;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_2__|__2_wc_ridgewood";
                this.routed_trk_3 = true;
                this.sig_6w = true;
            }
        }
    }
```

```
    // ---- END click_sig_6w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_2e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (this.sw_1 || this.sw_5) {
            return;
        }
        else if (!this.sw_3 && !this.sw_7 && !this.sw_9) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_ridgewood_suscon";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else if (this.sw_3 && !this.sw_7 && !this.sw_9) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
                return;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
```

```
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_ridgewood_suscon";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else if (!this.sw_3 && this.sw_7 && !this.sw_9) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
                return;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__1_ridgewood_bt";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else if (!this.sw_3 && !this.sw_7 && this.sw_9) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
                return;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_4__|__2_ridgewood_bt";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----
```

```
    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_4e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_5) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
                return;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_ridgewood_suscon";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_5 && !this.sw_7 && !this.sw_9) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
```

```
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_ridgewood_suscon";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_5 && this.sw_7) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
                return;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__1_ridgewood_bt";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_5 && !this.sw_7 && this.sw_9) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
                return;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_4__|__2_ridgewood_bt";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----
```

```
    /**
     * click_sig_6e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     * @param next_block_4, The next block on Track #4
     */
    click_sig_6e(next_block_1, next_block_2, next_block_3,
next_block_4) {
        if (this.sw_7) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
                return;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_3__|__1_ridgewood_bt";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
        else if (this.sw_1 && !this.sw_3 && !this.sw_5 && !this.sw_7
&& !this.sw_9) {
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
                return;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
```

```
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_1__|__1_ridgewood_suscon";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
        else if (this.sw_1 && !this.sw_3 && !this.sw_5 && !this.sw_7
&& this.sw_9) {
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
                return;
            }
            else {
                if (next_block_4 === Occupied || next_block_4 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_4__|__2_ridgewood_bt";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
        else if (this.sw_1 && this.sw_3 && !this.sw_5) {
            if (this.sig_6e) {
                this.route_e_trk_3 = null;
                this.routed_trk_3 = false;
                this.sig_6e = false;
                return;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_2__|__2_ridgewood_suscon";
                this.routed_trk_3 = true;
                this.sig_6e = true;
            }
        }
    }
    // ---- END click_sig_6e() ----
```

```
    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_e_trk_2, this.route_e_trk_3,
            this.route_w_trk_2, this.route_w_trk_4,
this.route_w_trk_3,
            this.route_e_trk_1, this.route_w_trk_1
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_1 = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
```

```
            this.occupied_trk_2 = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * set_trk_3_occupied()
     * @summary Sets track #3 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_3_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_3 = n_state;
            this.routed_trk_3 = false;
            this.trk_3_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_3_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;
        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w_1 = false;
            this.sig_2w_2 = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
```

```
            this.route_w_trk_4 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.occupied_trk_1 = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.occupied_trk_2 = false;
            this.trk_2_time = null;
        }
        // Track #3
        if (current_time - this.trk_3_time > 4 && current_time -
this.trk_3_time< 100000) {
            this.sig_6w = false;
            this.sig_6e = false;

            this.route_w_trk_3 = null;
            this.route_e_trk_3 = null;
            this.routed_trk_3 = false;

            this.occupied_trk_3 = false;
            this.trk_3_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * @summary Function to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
```

```
    }
    // ---- END throw_sw_1() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * @summary Funtion to throw switch #5 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_5() {
        if (this.sw_5 === false) {
            this.sw_5 = true;
        }
        else {
            this.sw_5 = false;
        }
    }
    // ---- END throw_sw_5() ----

    /**
     * @summary Funtion to throw switch #7 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_7() {
        if (this.sw_7 === false) {
            this.sw_7 = true;
        }
        else {
            this.sw_7 = false;
```

```
        }
    }
    // ---- END throw_sw_7() ----

    /**
     * @summary Funtion to throw switch #9 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_9() {
        if (this.sw_9 === false) {
            this.sw_9 = true;
        }
        else {
            this.sw_9 = false;
        }
    }
    // ---- END throw_sw_9() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        var status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_5: this.sw_5,
            sw_7: this.sw_7,
            sw_9: this.sw_9,
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            routed_trk_3: this.routed_trk_3,
            occupied_trk_1: this.occupied_trk_1,
            occupied_trk_2: this.occupied_trk_2,
            occupied_trk_3: this.occupied_trk_3,
            routes: this.get_routes()
        };

        return status;
```

```
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Ridgewood;
```

```
/**
 * @file ctc_sf.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the SF Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the SF Interlocking This class is what
controlls the SF Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e_1 -> Bool if Signal #4e-1 is Lined or Not
 * @member sig_4e_2 -> Bool if Signal #4e-2 is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 * @member route_e_trk_3 = The east bound route for track #3
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_SF {
    /**
     * constructor()
```

```
     * @summary The constructor for the CTC_SF class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e_1 = false;
        this.sig_4e_2 = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        this.route_e_trk_3 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
        this.trk_1_occupied = false;
        this.trk_2_occupied = false;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else {
                return this.route_w_trk_2;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
```

```
            }
            else if (track === "2") {
                return this.route_e_trk_2;
            }
            else {
                return this.route_e_trk_3;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2w(next_block_1, next_block_2, next_block_3) {
        if (!this.sw_3) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_sterling_sf";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
        else if (this.sw_3 && !this.sw_1) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_hilburn_sf";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
        else if (this.sw_3 && this.sw_1) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_3__|__2_yardHilburn_sf";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4w(next_block_2, next_block_3) {
        if (this.sw_3) {
            return;
        }
```

```
        else if (!this.sw_1) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_hilburn_sf";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (this.sw_1) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_3__|__3_yardHilburn_sf";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
```

```
     */
    click_sig_2e(next_block_1) {
        if (this.sw_3) {
            return;
        }
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_sf_wc";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4e_1(next_block_1, next_block_2) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_4e_1) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e_1 = false;
            }
            else {
```

```
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_sf_wc";
                this.routed_trk_2 = true;
                this.sig_4e_1 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_4e_1) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e_1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_sf_wc";
                this.routed_trk_2 = true;
                this.sig_4e_1 = true;
            }
        }
    }
    // ---- END click_sig_4e_1() ----

    /**
     * click_sig_4e_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4e_2(next_block_1, next_block_2) {
        if (!this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
```

```
            if (this.sig_4e_2) {
                this.route_e_trk_3 = null;
                this.routed_trk_2 = false;
                this.sig_4e_2 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_2__|__2_sf_wc";
                this.routed_trk_2 = true;
                this.sig_4e_2 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_4e_2) {
                this.route_e_trk_3 = null;
                this.routed_trk_2 = false;
                this.sig_4e_2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_1__|__1_sf_wc";
                this.routed_trk_2 = true;
                this.sig_4e_2 = true;
            }
        }
    }
    // ---- END click_sig_4e_2() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
```

```
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.trk_2_occupied = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get current time
        let current_time = new Date().getTime() / 1000;

        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;
```

```
            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e_1 = false;
            this.sig_4e_2 = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.route_e_trk_3 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
```

```
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1, this.route_e_trk_2, this.route_e_trk_3
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            routes: this.get_routes(),
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}
```

```
// This is required when using ReactJS
export default CTC_SF;
```

```
/**
 * @file ctc_suscon.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the Suscon Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the Suscon Interlocking This class is what
controlls the Suscon Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_Suscon {
    /**
     * constructor()
     * @summary The constructor for the CTC_Suscon class
     *
```

```
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_2e = false;
        this.sig_4w = false;
        this.sig_4e = false;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.occupied_trk_1 = false;
        this.occupied_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
    }
    // ---- END constructor() ----

    /**
     * click_sig()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param sigNum, The signal number that was clicked
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig(sigNum, next_block_1, next_block_2) {
        if (sigNum === "2W") {
            if (this.sw_3) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_2w) {
                    this.route_w_trk_1 = null;
                    this.routed_trk_1 = false;
```

```
                    this.sig_2w = false;
                    return;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_1 = "W_1_1__|
__1_ridgewood_suscon";
                    this.routed_trk_1 = true;
                    this.sig_2w = true;
                }
            }
            else if (this.sw_1 && !this.sw_3){
                if (this.sig_2w) {
                    this.route_w_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2w = false;
                    return;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_1 = "W_1_2__|
__2_ridgewood_suscon";
                    this.routed_trk_1 = true;
                    this.sig_2w = true;
                }
            }
        }
        else if (sigNum === "4W") {
            if (this.sw_1) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_4w) {
                    this.route_w_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4w = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
```

```
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_2 = "W_2_2__|
__2_ridgewood_suscon";
                    this.routed_trk_2 = true;
                    this.sig_4w = true;
                }
            }
            else if (!this.sw_1 && this.sw_3) {
                if (this.sig_4w) {
                    this.route_w_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4w = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_2 = "W_2_1__|
__1_ridgewood_suscon";
                    this.routed_trk_2 = true;
                    this.sig_4w = true;
                }
            }
        }
        else if (sigNum === "2E") {
            if (this.sw_1) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_2e) {
                    this.route_e_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2e = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_1 = "E_1_1__|__1_suscon_mill";
                    this.routed_trk_1 = true;
                    this.sig_2e = true;
```

```
                }
            }
            else if (!this.sw_1 && this.sw_3) {
                if (this.sig_2e) {
                    this.route_e_trk_1 = null;
                    this.routed_trk_1 = false;
                    this.sig_2e = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_1 = "E_1_2__|__2_suscon_mill";
                    this.routed_trk_1 = true;
                    this.sig_2e = true;
                }
            }
        }
        else if (sigNum === "4E") {
            if (this.sw_3) {
                return;
            }
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_4e) {
                    this.route_e_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4e = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_2 = "E_2_2__|__2_suscon_mill";
                    this.routed_trk_2 = true;
                    this.sig_4e = true;
                }
            }
            else if (this.sw_1 && !this.sw_3) {
                if (this.sig_4e) {
                    this.route_e_trk_2 = null;
                    this.routed_trk_2 = false;
                    this.sig_4e = false;
                }
                else {
```

```
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_2 = "E_2_1__|__1_suscon_mill";
                    this.routed_trk_2 = true;
                    this.sig_4e = true;
                }
            }
        }
    }
    // ---- END click_sig() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_1 = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.occupied_trk_2 = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
```

```
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;
        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.occupied_trk_1 = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e_1 = false;
            this.sig_4e_2 = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.route_e_trk_3 = null;
            this.routed_trk_2 = false;

            this.occupied_trk_2 = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * get_routes()
```

```
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1, this.route_e_trk_2
        ];
        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else {
                return this.route_w_trk_2;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * throw_sw_1()
     * @summary Changes the current state of switch #1, used when user
clicks the switch
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
```

```
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * throw_sw_3()
     * @summary Changes the current state of switch #3, used when user
clicks the switch
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        var status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            occupied_trk_1: this.occupied_trk_1,
            occupied_trk_2: this.occupied_trk_2,
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            routes: this.get_routes()
        };

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
```

```
export default CTC_Suscon;
```

```
/**
 * @file ctc_wc.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the WC Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the WC Interlocking This class is what
controlls the WC Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 * @member sw_5 -> Bool if Switch #5 is Reveresed or Not
 * @member sw_7 -> Bool if Switch #7 is Reveresed or Not
 *
 * @member sig_2w_1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w_2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e_1 -> Bool if Signal #2e-1 is Lined or Not
 * @member sig_2e_2 -> Bool if Signal #2e-2 is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 * @member route_e_trk_3 = The east bound route for track #3
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member routed_trk_3 = Bool if track #3 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_3_time = The time track #3 was occupied, used to know
```

```
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 * @member trk_3_occupied = Bool if track #3 is occupied or not
 */
class CTC_WC {
    /**
     * constructor()
     * @summary The constructor for the CTC_WC class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        this.sw_5 = false;
        this.sw_7 = false;
        // Bools for the signals
        this.sig_2w_2 = false;
        this.sig_2w_1 = false;
        this.sig_4w = false;
        this.sig_2e_2 = false;
        this.sig_2e_1 = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_w_trk_3 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        this.route_e_trk_3 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
        this.trk_1_occupied = false;
        this.trk_2_occupied = false;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
```

```
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else if (track === "2") {
                return this.route_w_trk_2;
            }
            else {
                return this.route_w_trk_3;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else if (track === "2") {
                return this.route_e_trk_2;
            }
            else {
                return this.route_e_trk_3;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w_1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2w_1(next_block_1, next_block_2, next_block_3) {
        if (this.sw_5 || this.sw_7) {
            return;
        }
        else if (!this.sw_1 && !this.sw_3) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
            }
```

```
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_sf_wc";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
        else if (this.sw_1 && !this.sw_3) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_sf_wc";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_3__|__0_yard_wc";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
    }
    // ---- END click_sig_2w_1() ----
```

```
    /**
     * click_sig_2w_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2w_2(next_block_1, next_block_2, next_block_3) {
        if(!this.sw_7 || this.sw_5) {
            return;
        }
        else if (!this.sw_1 && !this.sw_3) {
            if (this.sig_2w_2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_1__|__1_sf_wc";
                this.routed_trk_1 = true;
                this.sig_2w_2 = true;
            }
        }
        else if (this.sw_1 && !this.sw_3) {
            if (this.sig_2w_2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
```

```
                this.route_w_trk_3 = "W_3_2__|__2_sf_wc";
                this.routed_trk_1 = true;
                this.sig_2w_2 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_2w_2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_3__|__0_yard_wc";
                this.routed_trk_1 = true;
                this.sig_2w_2 = true;
            }
        }
    }
    // ---- END click_sig_2w_2() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4w(next_block_1, next_block_2, next_block_3) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_5) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_sf_wc";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (!this.sw_3 && this.sw_5) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_sf_wc";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else if (this.sw_3 && this.sw_5) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_3__|__0_yard_wc";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----
```

```
    /**
     * click_sig_2e_1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2e_1(next_block_1, next_block_2, next_block_3) {
        if (this.sw_1 || this.sw_3) {
            return;
        }
        else if (!this.sw_5 && !this.sw_7) {
            if (this.sig_2e_1) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e_1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2e_1 = true;
            }
        }
        else if (this.sw_5) {
            if (this.sig_2e_1) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e_1 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
```

```
                this.route_e_trk_1 = "E_1_2__|__2_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2e_1 = true;
            }
        }
        else if (!this.sw_5 && this.sw_7) {
            if (this.sig_2e_1) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e_1 = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__3_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2e_1 = true;
            }
        }
    }
    // ---- END click_sig_2e_1() ----

    /**
     * click_sig_2e_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2e_2(next_block_1, next_block_2, next_block_3) {
        if (!this.sw_3) {
            return;
        }
        else if (!this.sw_5 && !this.sw_7) {
            if (this.sig_2e_2) {
                this.route_e_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2e_2 = false;
            }
```

```
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_1__|__1_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2e_2 = true;
            }
        }
        else if (this.sw_5) {
            if (this.sig_2e_2) {
                this.route_e_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2e_2 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_2__|__2_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2e_2 = true;
            }
        }
        else if (!this.sw_5 && this.sw_7) {
            if (this.sig_2e_2) {
                this.route_e_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2e_2 = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_3__|__3_wc_ridgewood";
                this.routed_trk_1 = true;
                this.sig_2e_2 = true;
            }
        }
    }
    // ---- END click_sig_2e_2() ----
```

```
    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4e(next_block_1, next_block_2, next_block_3) {
        if (this.sig_5) {
            return;
        }
        else if (!this.sw_1 && !this.sw_5) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_wc_ridgewood";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_1 && !this.sw_3 && !this.sw_5 && !this.sw_7)
{
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
```

```
                }
                this.route_e_trk_2 = "E_2_1__|__1_wc_ridgewood";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_1 && !this.sw_3 && !this.sw_5 && this.sw_7) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__3_wc_ridgewood";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
```

```
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.trk_2_occupied = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get Current Time
        let current_time = new Date().getTime() / 1000;
        // Track #1 & Track #3
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.route_w_trk_3 = null;
            this.route_e_trk_3 = null;
            this.routed_trk_1 = false;

            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e_1 = false;
            this.sig_4e_2 = false;
```

```
            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * @summary Funtion to throw switch #5 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
```

```
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_5() {
        if (this.sw_5 === false) {
            this.sw_5 = true;
        }
        else {
            this.sw_5 = false;
        }
    }
    // ---- END throw_sw_5() ----

    /**
     * @summary Funtion to throw switch #7 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_7() {
        if (this.sw_7 === false) {
            this.sw_7 = true;
        }
        else {
            this.sw_7 = false;
        }
    }
    // ---- END throw_sw_7() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_3,
            this.route_e_trk_1, this.route_e_trk_3,
            this.route_e_trk_2, this.route_w_trk_2,
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
```

```
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_5: this.sw_5,
            sw_7: this.sw_7,
            routes: this.get_routes(),
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_WC;
```

```
/**
 * @file ctc_westSecaucus.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the West Secacus Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the West Secacus Interlocking This class
is what controlls the West Secacus Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_WestSecaucus {
    /**
     * constructor()
     * @summary The constructor for the CTC_WestSecaucus class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
```

```
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_2e = false;
        this.sig_4w = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * click_sig()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param sigNum, the id of the signal clicked
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig(sigNum, next_block_1, next_block_2) {
        if (sigNum === "2W") {
            if (this.sw_3) {
                return
            }
            // Route W_1_1
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_2w) {
                    this.route_w_trk_1 = null;
                    this.sig_2w = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_1 = "W_1_1__|
```

```
__1_mill_westSecaucus"
                    this.sig_2w = true;
                }
            }
            // Route W_1_2
            else if (this.sw_1 && !this.sw_3) {
                if (this.sig_2w) {
                    this.route_w_trk_1 = null;
                    this.sig_2w = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        return;
                    }
                    this.route_w_trk_1 = "W_1_2__|
__2_mill_westSecaucus"
                    this.sig_2w = true;
                }
            }
        }
        else if (sigNum === "4W") {
            if (!this.sw_3) {
                return;
            }
            // Route W_2_1
            if (!this.sw_1 && this.sw_3) {
                if (this.sig_4w) {
                    this.route_w_trk_2 = null;
                    this.sig_4w = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_2 = "W_2_1__|
__1_mill_westSecaucus"
                    this.sig_4w = true;
                }
            }
            // Route W_2_2
            else if (this.sw_1 && this.sw_3) {
                if (this.sig_4w) {
                    this.route_w_trk_2 = null;
                    this.sig_4w = false;
                }
                else {
```

```
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_w_trk_2 = "W_2_2__|
__2_mill_westSecaucus"
                    this.sig_4w = true;
                }
            }
        }
        else if (sigNum === "2E") {
            if (this.sw_1) {
                return;
            }
            // Route E_1_1
            else if (!this.sw_1 && !this.sw_3) {
                if (this.sig_2e) {
                    this.route_e_trk_1 = null;
                    this.sig_2e = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_1 = "E_1_1__|
__2_westSecaucus_laurel"
                    this.sig_2e = true;
                }
            }
            // Route E_1_2
            else if (!this.sw_1 && this.sw_3) {
                if (this.sig_2e) {
                    this.route_e_trk_1 = null;
                    this.sig_2e = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_1 = "E_1_2__|
__4_westSecaucus_laurel"
                    this.sig_2e = true;
```

```
                }
            }
        }
        else if (sigNum === "4E") {
            if (!this.sw_1) {
                return;
            }
            // Route E_2_1
            else if (this.sw_1 && !this.sw_3) {
                if (this.sig_4e) {
                    this.route_e_trk_2 = null;
                    this.sig_4e = false;
                }
                else {
                    if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_2 = "E_2_1__|
__2_westSecaucus_laurel";
                    this.sig_4e = true;
                }
            }
            // Route E_2_2
            else if (this.sw_1 && this.sw_3) {
                if (this.sig_4e) {
                    this.route_e_trk_2 = null;
                    this.sig_4e = false;
                }
                else {
                    if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                        alert("Cannot Line Route Because Conflict With
Next Block");
                        return;
                    }
                    this.route_e_trk_2 = "E_2_2__|
__4_westSecaucus_laurel"
                    this.sig_4e = true;
                }
            }
        }
    }
    // ---- END click_sig() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
```

```
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2e = false;
            this.sig_4e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.route_e_trk_2 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
```

```
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1, this.route_e_trk_2
        ];
        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_2;
            }
            else {
                return this.route_w_trk_1;
            }
        }
        else {
            if (track === "1") {
                return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
```

```
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            routes: this.get_routes(),
            occupied: this.int_occupied
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_WestSecaucus;
```

```
/**
 * @file ctc_bc.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP BC  Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP BC  Interlocking This class is what
controlls the CP BC  Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_BC {
    /**
     * constructor()
     * @summary The constructor for the CTC_BC class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        // Bools for the signals
        this.sig_2w = false;
```

```
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            return this.route_w_trk_1;
        }
        else {
         if (track === "1") {
                  return this.route_e_trk_1;
         }
         else {
                  return this.route_e_trk_2;
         }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w(next_block_1, next_block_2) {
        if (this.sw_1) {
            if (this.sig_2w) {
```

```
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_pa_bc";
                this.sig_2w = true;
            }
        }
        else {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_port_bc";
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2e(next_block_1) {
        if (this.sw_1) {
            return;
        }
        else {
```

```
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_bc_ov";
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_4e(next_block_1) {
        if (!this.sw_1) {
            return;
        }
        else {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_bc_ov";
                this.sig_4e = true;
            }
        }
```

```
    }
    // ---- END click_sig_4e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get current time
        let current_time = new Date().getTime() / 1000;

        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2e = false;
            this.sig_4e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.route_e_trk_2 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----
```

```
    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
```

```
            occupied: this.int_occupied,
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_BC;
```

```
/**
 * @file ctc_hall.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Hall Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP Hall Interlocking This class is
what controlls the CP Hall Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_Hall {
    /**
     * constructor()
     * @summary The constructor for the CTC_Hall class
     *
     * @description This will initialize all the member variables when
```

```
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
                 this.route_w_trk_2 = null;
                 this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = false;
        this.routed_trk_2 = false;
        this.trk_1_occupied = false;
        this.trk_2_occupied = false;
        this.trk_1_time = null;
        this.trk_2_time = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                 return this.route_w_trk_1;
        }
        else {
                 return this.route_w_trk_2;
        }
        }
        else {
        if (track === "1") {
                 return this.route_e_trk_1;
        }
        else {
                 return this.route_e_trk_2;
        }
        }
    }
```

```
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w(next_block_1) {
        if (this.sw_1) {
            return;
        }
        else {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_howells_hall";
                this.routed_trk_1 = true;
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
```

```
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4w(next_block_1, next_block_2) {
        if (this.sw_1) {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_howells_hall";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
        else {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_yard_hall";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
```

```
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2e(next_block_1, next_block_2) {
        if (this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_hall_hudson";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_hall_hudson";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
```

```
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_4e(next_block_2) {
        if (this.sw_1) {
            return;
        }
        else {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_hall_hudson";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
```

```
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.trk_2_occupied = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check both track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get The Current Time
        let current_time = new Date().getTime() / 1000;

        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
        // Track #2
```

```
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * @summary Funtion to throw switch #21 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
```

```
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            routes: this.get_routes(),
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Hall;
```

```
/**
 * @file ctc_harriman.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Harriman Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupy = '#eb3323';


/**
 * Class is the Backend for the CP Harriman Interlocking This class is
what controlls the CP Harriman Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of
 * the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_21 -> Bool if Switch #21 is Reveresed or Not
 * @member sw_32 -> Bool if Switch #32 is Reveresed or Not
 *
 * @member sig_1w -> Bool if Signal #1w is Lined or Not
 * @member sig_1e -> Bool if Signal #1e is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_3e -> Bool if Signal #3e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 * @member route_e_trk_3 = The east bound route for track #3
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Harriman {
    /**
     * constructor()
     * @summary The constructor for the CTC_Harriman class
     *
     * @description This will initialize all the member variables when
the program is started
     */
        constructor() {
        // Bools for the switches
```

```
                this.sw_21 = false;
                this.sw_32 = false;
        // Bools for the signals
                this.sig_1w = false;
                this.sig_1e = false;
                this.sig_2e = false;
                this.sig_3e = false;
        // Track routes
                this.route_w_trk_1 = null;
                this.route_e_trk_1 = null;
                this.route_e_trk_2 = null;
                this.route_e_trk_3 = null;
        // Used for routing and occupying the tracks
                this.int_occupied = false;
        this.time_occupied = null;
        }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
               return this.route_w_trk_1;
        }
        else {
         if (track === "1") {
                return this.route_e_trk_1;
         }
         else if (track === "2") {
                return this.route_e_trk_2;
         }
         else {
                return this.route_e_trk_3;
         }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
```

```
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
        click_sig_1w(next_block_1, next_block_2, next_block_3) {
                 if (!this.sw_32 && !this.sw_21) {
                         if (this.sig_1w) {
                                 this.route_w_trk_1 = null;
                                 this.sig_1w = false;
                         }
                         else {
                                 if (next_block_1 === Occupy ||
next_block_1 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_valley_harriman";
                this.sig_1w = true;
                         }
                 }
                 else if (this.sw_32) {
                         if (this.sig_1w) {
                                 this.route_w_trk_1 = null;
                                 this.sig_1w = false;
                         }
                         else {
                                 if (next_block_3 === Occupy ||
next_block_3 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_3__|
__3_industrial_harriman";
                this.sig_1w = true;
                         }
                 }
                 else if (!this.sw_32 && this.sw_21) {
                         if (this.sig_1w) {
                                 this.route_w_trk_1 = null;
                                 this.sig_1w = false;
                         }
                         else {
                                 if (next_block_2 === Occupy ||
next_block_2 === Lined) {
                    alert("Cannot Line Route Because Conflict With
```

```
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_valley_harriman";
                this.sig_1w = true;
                        }
                 }
        }
    // ---- END click_sig_1w() ----

    /**
     * click_sig_1e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
        click_sig_1e(next_block_1) {
                 if (this.sw_21 || this.sw_32) {
                        return;
                 }
                 else {
                        if (this.sig_1e) {
                                this.route_e_trk_1 = null;
                                this.sig_1e = false;
                        }
                        else {
                                if (next_block_1 === Occupy ||
next_block_1 === Lined) {
                   alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_harriman_sterling";
                this.sig_1e = true;
                        }
                 }
        }
    // ---- END click_sig_1e() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
```

```
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
        click_sig_2e(next_block_1) {
                if (!this.sw_21) {
                        return;
                }
                else {
                        if (this.sig_2e) {
                                this.route_e_trk_2 = null;
                                this.sig_2e = false;
                        }
                        else {
                                if (next_block_1 === Occupy ||
next_block_1 === Lined) {
                   alert("Cannot Line Route Because Conflict With
Next Block");
                   return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_harriman_sterling";
                this.sig_2e = true;
                        }
                 }
        }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_3e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
        click_sig_3e(next_block_1) {
                if (!this.sw_32) {
                        return;
                }
                else {
                        if (this.sig_3e) {
                                this.route_e_trk_3 = null;
                                this.sig_3e = false;
```

```
                        }
                        else {
                                if (next_block_1 === Occupy ||
next_block_1 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_1__|__1_harriman_sterling";
                this.sig_3e = true;
                        }
                 }
         }
    // ---- END click_sig_3e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true || n_state === false) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get current time
        let current_time = new Date().getTime() / 1000;

        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_1w = false;
```

```
                        this.sig_1e = false;
                        this.sig_2e = false;
                        this.sig_3e = false;

                        this.route_w_trk_1 = null;
                        this.route_e_trk_1 = null;
                        this.route_e_trk_2 = null;
                        this.route_e_trk_3 = null;

                        this.int_occupied = false;
                        this.time_occupied = null;
        }
    }

        /**
     * @summary Funtion to throw switch #21 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_21() {
        if (this.sw_21 === false) {
            this.sw_21 = true;
        }
        else {
            this.sw_21 = false;
        }
    }
    // ---- END throw_sw_21() ----

    /**
     * @summary Funtion to throw switch #32 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_32() {
        if (this.sw_32 === false) {
            this.sw_32 = true;
        }
        else {
            this.sw_32 = false;
        }
    }
    // ---- END throw_sw_32() ----

    /**
     * get_routes()
```

```
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1,
            this.route_e_trk_1, this.route_e_trk_2, this.route_e_trk_3
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_21: this.sw_21,
                        sw_32: this.sw_32,
                        occupied: this.int_occupied,
                        routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Harriman;
```

```
/**
 * @file ctc_howells.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Howells Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';

/**
 * Class is the Backend for the CP Howells Interlocking This class is
what controlls the CP Howells Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_2es -> Bool if Signal #2es is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Howells {
    /**
     * constructor()
     * @summary The constructor for the CTC_Howells class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        this.sw_3 = false;

        this.sig_2w = false;
        this.sig_2e = false;
        this.sig_2es = false;
```

```
        this.route_w_trk_1 = null;
                 this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;

        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            return this.route_w_trk_1;
        }
        else {
         if (track === "1") {
                 return this.route_e_trk_1;
         }
         else {
                 return this.route_e_trk_2;
         }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w(next_block_1, next_block_2) {
        if (this.sw_3) {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_ov_howells";
                this.sig_2w = true;
            }
        }
        else {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_ov_howells";
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2e(next_block_1) {
        if (this.sw_3) {
            return;
        }
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
```

```
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_howells_hall";
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_2es()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2es(next_block_1) {
        if (!this.sw_3) {
            return;
        }
        else {
            if (this.sig_2es) {
                this.route_e_trk_2 = null;
                this.sig_2es = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_howells_hall";
                this.sig_2es = true;
            }
        }
    }
    // ---- END click_sig_4e() ----
```

```
    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get current time
        let current_time = new Date().getTime() / 1000;

        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2e = false;
            this.sig_2es = false;

            this.route_w_trk_1 = null;
                 this.route_e_trk_1 = null;
            this.route_e_trk_2 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
```

```
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_3: this.sw_3,
            routes: this.get_routes(),
            occupied: this.int_occupied
        }
```

```
        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Howells;
```

```
/**
 * @file ctc_.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Hudson Junction
Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP Hudson Junction Interlocking his
class is what controlls the CP Hudson Junction Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible, and the ReactJS Component
class
 * gets information from this class to display the correct status of
the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2ws -> Bool if Signal #2ws is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_2es -> Bool if Signal #2es is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Hudson {
    /**
     * constructor()
     * @summary The constructor for the CTC_Hudson class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
```

```
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_2ws = false;
        this.sig_2e = false;
        this.sig_2es = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_3 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
               return this.route_w_trk_1;
            }
            else {
                return this.route_w_trk_3;
            }
        }
        else {
         if (track === "1") {
                  return this.route_e_trk_1;
            }
            else {
                return this.route_e_trk_2;
            }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
```

```
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w(next_block_1, next_block_2) {
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2w) {
                                    this.route_w_trk_1 = null;
                                    this.sig_2w = false;
                        }
                        else {
                                    if (next_block_1 === Occupied ||
next_block_1 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_hall_hudson";
                this.sig_2w = true;
                        }
        }
        else if (this.sw_1) {
            if (this.sig_2w) {
                                    this.route_w_trk_1 = null;
                                    this.sig_2w = false;
                        }
                        else {
                                    if (next_block_2 === Occupied ||
next_block_2 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_hall_hudson";
                this.sig_2w = true;
                        }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_2ws()
```

```
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2ws(next_block_1, next_block_2) {
        if (!this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2ws) {
                                this.route_w_trk_3 = null;
                                this.sig_2ws = false;
                        }
                        else {
                                if (next_block_1 === Occupied ||
next_block_1 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_1__|__1_hall_hudson";
                this.sig_2ws = true;
                        }
        }
        else if (this.sw_1) {
            if (this.sig_2ws) {
                                this.route_w_trk_3 = null;
                                this.sig_2ws = false;
                        }
                        else {
                                if (next_block_2 === Occupied ||
next_block_2 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_2__|__2_hall_hudson";
                this.sig_2ws = true;
                        }
        }
    }
    // ---- END click_sig_2ws() ----
```

```
    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2e(next_block_1, next_block_3) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2e) {
                                this.route_e_trk_1 = null;
                                this.sig_2e = false;
                        }
                        else {
                                if (next_block_1 === Occupied ||
next_block_1 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_hudson_valley";
                this.sig_2e = true;
                        }
        }
        else if (this.sw_3) {
            if (this.sig_2e) {
                                this.route_e_trk_1 = null;
                                this.sig_2e = false;
                        }
                        else {
                                if (next_block_3 === Occupied ||
next_block_3 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__1_hudson_nysw";
                this.sig_2e = true;
                        }
        }
    }
```

```
    // ---- END click_sig_2e() ----

    /**
     * click_sig_2es()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2es(next_block_1, next_block_3) {
        if (!this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2es) {
                                this.route_e_trk_2 = null;
                                this.sig_2es = false;
                        }
                        else {
                                if (next_block_1 === Occupied ||
next_block_1 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_hudson_valley";
                this.sig_2es = true;
                        }
        }
        else if (this.sw_3) {
            if (this.sig_2es) {
                                this.route_e_trk_2 = null;
                                this.sig_2es = false;
                        }
                        else {
                                if (next_block_3 === Occupied ||
next_block_3 === Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__1_hudson_nysw";
                this.sig_2es = true;
                        }
```

```
        }
    }
    // ---- END click_sig_2es() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        //console.log(new Date().getTime() / 1000 -
this.time_occupied)
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2ws = false;
            this.sig_2e = false;
            this.sig_2es = false;

            this.route_w_trk_1 = null;
            this.route_w_trk_3 = null;
            this.route_e_trk_1 = null;
            this.route_e_trk_2 = null;

            this.int_occupied = false;
            this.time_occupied = null;
```

```
        }
    }
    // ---- END can_clear() ----

    /**
     * get_occupied()
     * @breif Getter for the int_occupied variable
     *
     * @returns If the interlocking is occupied or not
     */
    get_occupied() {
        return this.int_occupied;
    }
    // ---- END get_occupied() ----

    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
```

```
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_3,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            occupied: this.get_occupied(),
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Hudson;
```

```
/**
 * @file ctc_ov.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP OV Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP OV Interlocking This class is what
controlls the CP OV Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2ws -> Bool if Signal #2ws is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The east bound route for track #2
 * @member route_e_trk_1 = The east bound route for track #1
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_OV {
    /**
     * constructor()
     * @summary The constructor for the CTC_OV class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        // Bools for the signals
        this.sig_2w = false;
```

```
        this.sig_2ws = false;
        this.sig_2e = false;
        // Track routes
        this.route_w_trk_1 = null;
                 this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                 return this.route_w_trk_1;
            }
            else {
                 return this.route_w_trk_2;
            }
            }
            else {
            return this.route_e_trk_1;
            }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w(next_block_1) {
        if (this.sw_1) {
            return;
        }
```

```
        else {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_bc_ov";
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_2ws()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2ws(next_block_1) {
        if (!this.sw_1) {
            return;
        }
        else {
            if (this.sig_2ws) {
                this.route_w_trk_2 = null;
                this.sig_2ws = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_bc_ov";
                this.sig_2ws = true;
            }
```

```
        }
    }
    // ---- END click_sig_2ws() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2e(next_block_1, next_block_2) {
        if (this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_ov_howells";
                this.sig_2e = true;
            }
        }
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_ov_howells";
                this.sig_2e = true;
            }
        }
```

```
    }
    // ---- END click_sig_2e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        //console.log(new Date().getTime() / 1000 -
this.time_occupied)
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2ws = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_w_trk_2 = null;
            this.route_e_trk_1 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----
```

```
    /**
     * throw_sw_1()
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
```

```
            sw_1: this.sw_1,
            occupied: this.int_occupied,
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_OV;
```

```
/**
 * @file ctc_pa.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP PA Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP PA Interlocking This class is what
controlls the CP PA Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_PA {
    /**
     * constructor()
```

```
     * @summary The constructor for the CTC_PA class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w_1 = false;
        this.sig_2w_2 = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_w_trk_3 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = null;
        this.routed_trk_2 = null;
        this.trk_1_time = null;
        this.trk_2_time = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                 return this.route_w_trk_1;
        }
        else if (track === "2") {
                 return this.route_w_trk_2;
            }
            else if (track === "3") {
                 return this.route_w_trk_3;
            }
            else {
                 return this.route_w_trk_4;
        }
```

```
        }
        else {
         if (track === "1") {
                  return this.route_e_trk_1;
             }
             else if (track === "2") {
                  return this.route_e_trk_2;
             }
         else {
                  return this.route_e_trk_3;
         }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w_1(next_block_1, next_block_2, next_block_4) {
        if (this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_sparrow_pa";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
```

```
        else if (this.sw_3) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_sparrow_pa";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
    }
    // ---- END click_sig_2w1() ----

    /**
     * click_sig_2w2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w_2(next_block_1, next_block_2, next_block_4) {
        if (!this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2w_2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
```

```
                }
                this.route_w_trk_3 = "W_3_1__|__1_sparrow_pa";
                this.routed_trk_1 = true;
                this.sig_2w_2 = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_2w_2) {
                this.route_w_trk_3 = null;
                this.routed_trk_1 = false;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_2__|__2_sparrow_pa";
                this.routed_trk_1 = true;
                this.sig_2w_2 = true;
            }
        }
    }
    // ---- END click_sig_2w2() ----

    /**
     * click_sig_4w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_4w(next_block_2, next_block_4) {
        if (this.sw_3) {
            return;
        }
        else {
            if (this.sig_4w) {
                this.route_w_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4w = false;
            }
```

```
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_2__|__2_sparrow_pa";
                this.routed_trk_2 = true;
                this.sig_4w = true;
            }
        }
    }
    // ---- END click_sig_4w() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2e(next_block_1, next_block_3) {
        if (this.sw_3) {
            return;
        }
        else if (!this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_pa_port";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
```

```
        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__0_portYard_west";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_4e(next_block_1, next_block_2, next_block_3) {
        if (!this.sw_3) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_pa_bc";
```

```
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_3 && !this.sw_1) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_1__|__1_pa_port";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
        else if (this.sw_3 && this.sw_1) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_3__|__0_portYard_west";
                this.routed_trk_2 = true;
                this.sig_4e = true;
            }
        }
    }
    // ---- END click_sig_4e() ----

    /**
     * set_trk_1_occupied()
     * @summary Sets track #1 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
```

```
     */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_1_occupied() ----

    /**
     * set_trk_2_occupied()
     * @summary Sets track #2 as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_trk_2_occupied(n_state) {
        if (n_state === true) {
            this.trk_2_occupied = n_state;
            this.routed_trk_2 = false;
            this.trk_2_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_trk_2_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get Current Time
        let current_time = new Date().getTime() / 1000;

        // Track #1
        if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
            this.sig_2w = false;
```

```
            this.sig_2e = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;

            this.trk_1_occupied = false;
            this.trk_1_time = null;
        }
        // Track #2
        if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time< 100000) {
            this.sig_4w = false;
            this.sig_4e = false;

            this.route_w_trk_2 = null;
            this.route_e_trk_2 = null;
            this.routed_trk_2 = false;

            this.trk_2_occupied = false;
            this.trk_2_time = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * throw_sw_1()
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * throw_sw_3()
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
```

```
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * throw_sw_5()
     * @summary Funtion to throw switch #5 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_5() {
        if (this.sw_5 === false) {
            this.sw_5 = true;
        }
        else {
            this.sw_5 = false;
        }
    }
    // ---- END throw_sw_5() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
this.route_w_trk_3,
            this.route_e_trk_1, this.route_e_trk_2
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
```

```
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_5: this.sw_5,
            routes: this.get_routes(),
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_PA;
```

```
/**
 * @file ctc_port.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Port Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP Port Interlocking This class is
what controlls the CP Port Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e_1 -> Bool if Signal #2e_1 is Lined or Not
 * @member sig_2e_2 -> Bool if Signal #2e_2 is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_3 = The east bound route for track #3
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Port {
    /**
     * constructor()
     * @summary The constructor for the CTC_Port class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        // Bools for the signals
        this.sig_2w = false;
```

```
        this.sig_2e_1 = false;
        this.sig_2e_2 = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_3 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            return this.route_w_trk_1;
        }
        else {
         if (track === "1") {
                  return this.route_e_trk_1;
         }
         else {
                  return this.route_e_trk_3;
         }
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_2w(next_block_1, next_block_2) {
        if (this.sw_1) {
            if (this.sig_2w) {
```

```
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_3__|__3_yardEast_port";
                this.sig_2w = true;
            }
        }
        else {
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_pa_port";
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_2e_1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2e_1(next_block_1) {
        if (this.sw_1) {
            return;
        }
        else {
```

```
            if (this.sig_2e_1) {
                this.route_e_trk_1 = null;
                this.sig_2e_1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_port_bc";
                this.sig_2e_1 = true;
            }
        }
    }
    // ---- END click_sig_2e_1() ----

    /**
     * click_sig_2e_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2e_2(next_block_1) {
        if (!this.sw_1) {
            return;
        }
        else {
            if (this.sig_2e_2) {
                this.route_e_trk_3 = null;
                this.sig_2e_2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_3 = "E_3_1__|__1_port_bc";
                this.sig_2e_2 = true;
            }
        }
```

```
    }
    // ---- END click_sig_2e_2() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get Current Time
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2e_1 = false;
            this.sig_2e_2 = false;

            this.route_w_trk_1 = null;
            this.route_e_trk_1 = null;
            this.route_e_trk_3 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----
```

```
    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1,
            this.route_e_trk_1, this.route_e_trk_3
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            occupied: this.int_occupied,
```

```
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Port;
```

```
/**
 * @file ctc_sparrow.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Sparrow Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP Sparrow Interlocking This class is
what controlls the CP Sparrow Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Sparrow {
    /**
     * constructor()
     * @summary The constructor for the CTC_Sparrow class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
```

```
        this.sig_2w_1 = false;
        this.sig_2w_2 = false;
        this.sig_2w_3 = false;
        this.sig_2e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_w_trk_3 = null;
        this.route_e_trk_1 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else if (track === "2") {
                return this.route_w_trk_2;
            }
            else {
                return this.route_w_trk_3;
            }
        }
        else {
            return this.route_e_trk_1;
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w_1()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
```

```
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w_1(next_block_1) {
        if (this.sw_3 || this.sw_1) {
            return;
        }
        else {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_bingo_sparrow"
                this.sig_2w_1 = true;
            }
        }
    }
    // ---- END click_sig_2w_1() ----

    /**
     * click_sig_2w_2()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w_2(next_block_1) {
        if (!this.sw_1) {
            return;
        }
        else if (!this.sw_3) {
            if (this.sig_2w_2) {
                this.route_w_trk_3 = null;
                this.sig_2w_2 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
```

```
Next Block");
                    return;
                }
                this.route_w_trk_3 = "W_3_1__|__1_bingo_sparrow"
                this.sig_2w_2 = true;
            }
        }
    }
    // ---- END click_sig_2w_2() ----

    /**
     * click_sig_2w_3()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w_3(next_block_1) {
        if (!this.sw_3) {
            return;
        }
        else {
            if (this.sig_2w_3) {
                this.route_w_trk_2 = null;
                this.sig_2w_3 = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_bingo_sparrow"
                this.sig_2w_3 = true;
            }
        }
    }
    // ---- END click_sig_2w_3() ----

    /**
     * click_sig_2e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
```

```
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
     */
    click_sig_2e(next_block_1, next_block_2, next_block_3) {
        if (!this.sw_3 && !this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_sparrow_pa"
                this.sig_2e = true;
            }
        }
        else if (this.sw_3) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_sparrow_pa"
                this.sig_2e = true;
            }
        }
        if (!this.sw_3 && this.sw_1) {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.sig_2e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
```

```
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__0_sparrow_cripple"
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // The Current Time
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w_1 = false;
            this.sig_2w_2 = false;
            this.sig_2w_3 = false;
            this.sig_2e = false;

            this.route_w_trk_1 = null;
```

```
            this.route_w_trk_2 = null;
            this.route_w_trk_3 = null;
            this.route_e_trk_1 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----

    /**
     * @summary Funtion to throw switch #1 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_1() {
        if (this.sw_1 === false) {
            this.sw_1 = true;
        }
        else {
            this.sw_1 = false;
        }
    }
    // ---- END throw_sw_1() ----

    /**
     * @summary Funtion to throw switch #3 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_3() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
    // ---- END throw_sw_3() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
```

```
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
this.route_w_trk_3,
            this.route_e_trk_1
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            occupied: this.int_occupied,
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Sparrow;
```

```
/**
 * @file ctc_sterling.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Sterling Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP Sterling Interlocking This class is
what controlls the CP Sterling Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
display the correct status of the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Sterling {
    /**
     * constructor()
     * @summary The constructor for the CTC_Sterling class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_21 = false;
        // Bools for the signals
        this.sig_2w = false;
        this.sig_2ws = false;
```

```
        this.sig_1e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "0") {
                return this.route_w_trk_2;
            }
            else {
                return this.route_w_trk_1;
            }
        }
        else {
            return this.route_e_trk_1;
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w(next_block_1) {
        if (this.sw_21) {
            return;
        }
        else {
```

```
            if (this.sig_2w) {
                this.route_w_trk_1 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_harriman_sterling";
                this.sig_2w = true;
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_2ws()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2ws(next_block_1) {
        if (!this.sw_21) {
            return;
        }
        else {
            if (this.sig_2ws) {
                this.route_w_trk_1 = null;
                this.sig_2ws = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_2_1__|__1_harriman_sterling";
                this.sig_2ws = true;
            }
        }
```

```
    }
    // ---- END click_sig_2ws() ----

    /**
     * click_sig_1e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_1e(next_block_1, next_block_2) {
        if (!this.sw_21) {
            if (this.sig_1e) {
                this.route_e_trk_1 = null;
                this.sig_1e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_sterling_hilburn";
                this.sig_1e = true;
            }
        }
        else {
            if (this.sig_1e) {
                this.route_e_trk_1 = null;
                this.sig_1e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_sterling_sf";
                this.sig_1e = true;
            }
        }
    }
```

```
    // ---- END click_sig_1e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true || n_state === false) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get Current Time
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_2w = false;
            this.sig_2ws = false;
            this.sig_1e = false;

            this.route_w_trk_1 = null;
            this.route_w_trk_2 = null;
            this.route_e_trk_1 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----

    /**
```

```
     * @summary Funtion to throw switch #21 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_21() {
        if (this.sw_21 === false) {
            this.sw_21 = true;
        }
        else {
            this.sw_21 = false;
        }
    }
    // ---- END throw_sw_21() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_21: this.sw_21,
            occupied: this.int_occupied,
            routes: this.get_routes()
```

```
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Sterling;
```

```
/**
 * @file ctc_valley.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP Central Valley
Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * Class is the Backend for the CP Central Valley Interlocking This
class is what controlls the CP Central Valley Interlocking,
 * it is sort of like a backen, but is the controller, this is what
makes all the train movements possible, and the ReactJS Component
class
 * gets information from this class to display the correct status of
the interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 *
 * @member sig_2w -> Bool if Signal #2w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member time_occupied = The time the track was occupied, used to
know when to clear the route
 * @member int_occupied = Bool if the track is occupied or not
 */
class CTC_Valley {
    /**
     * constructor()
     * @summary The constructor for the CTC_Valley class
     *
     * @description This will initialize all the member variables when
the program is started
     */
    constructor() {
        // Bools for the switches
        this.sw_21 = false;
        // Bools for the signals
```

```
        this.sig_1w = false;
        this.sig_2w = false;
        this.sig_1e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_e_trk_1 = null;
        // Used for routing and occupying the tracks
        this.int_occupied = false;
        this.time_occupied = null;
    }
    // ---- END constructor() ----

    /**
     * get_train_route()
     * @summary Returns the route for the train at a given track
     *
     * @param direction, The direction the train is moving
     * @param track, The Track number of the train
     */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                 return this.route_w_trk_1;
         }
         else if (track === "2") {
                 return this.route_w_trk_2;
         }
        }
        else {
         return this.route_e_trk_1;
        }
    }
    // ---- END get_train_route() ----

    /**
     * click_sig_1w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_1w(next_block_1) {
        if (this.sw_21) {
            return;
```

```
        }
        else {
            if (this.sig_1w) {
                this.route_w_trk_1 = null;
                this.sig_1w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_1__|__1_hudson_valley";
                this.sig_1w = true;
            }
        }
    }
    // ---- END click_sig_1w() ----

    /**
     * click_sig_2w()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     */
    click_sig_2w(next_block_1) {
        if (!this.sw_21) {
            return;
        }
        else {
            if (this.sig_2w) {
                this.route_w_trk_2 = null;
                this.sig_2w = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_2 = "W_2_1__|__1_hudson_valley";
                this.sig_2w = true;
```

```
            }
        }
    }
    // ---- END click_sig_2w() ----

    /**
     * click_sig_1e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description When the function is called it will determine if a
route can be created,
     * and if so what the route is and sets it based off of the switch
status
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     */
    click_sig_1e(next_block_1, next_block_2) {
        if (this.sw_21) {
            if (this.sig_1e) {
                this.route_e_trk_1 = null;
                this.sig_1e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_2__|__2_valley_harriman";
                this.sig_1e = true;
            }
        }
        else {
            if (this.sig_1e) {
                this.route_e_trk_1 = null;
                this.sig_1e = false;
            }
            else {
                if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_1__|__1_valley_harriman";
                this.sig_1e = true;
            }
```

```
        }
    }
    // ---- END click_sig_1e() ----

    /**
     * set_occupied()
     * @summary Sets the track as occupied
     *
     * @param n_state, The new state of the track
     * This was used to test, and never removed passing the state as a
paramemter, which is not needed anymore
     */
    set_occupied(n_state) {
        if (n_state === true || n_state === false) {
            this.int_occupied = n_state;
            this.time_occupied = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
    // ---- END set_occupied() ----

    /**
     * can_clear()
     * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
     *
     * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
     * clears that track
     */
    can_clear() {
        // Get the current time
        let current_time = new Date().getTime() / 1000;
        if (current_time - this.time_occupied > 4 && current_time -
this.time_occupied < 100000) {
            this.sig_1w = false;
            this.sig_2w = false;
            this.sig_1e = false;

            this.route_w_trk_1 = null;
            this.route_w_trk_2 = null;
            this.route_e_trk_1 = null;

            this.int_occupied = false;
            this.time_occupied = null;
        }
    }
    // ---- END can_clear() ----
```

```
    /**
     * get_occupied()
     * @summary Getter for the int_occupied
     */
    get_occupied() {
        return this.int_occupied;
    }
    // ---- END get_occupied() ----

    /**
     * @summary Funtion to throw switch #21 in the interlocking
     *
     * The function sets the status of the switch, whether it is is
the normal possition
     * of reversed, (True = Reversed / False = Normal)
     */
    throw_sw_21() {
        if (this.sw_21 === false) {
            this.sw_21 = true;
        }
        else {
            this.sw_21 = false;
        }
    }
    // ---- END throw_sw_21() ----

    /**
     * get_routes()
     * @summary Gets all the routes from the interlocking
     *
     * @returns An Array holding every route variable from the
interlocking
     */
    get_routes() {
        let routes = [
            this.route_w_trk_1, this.route_w_trk_2,
            this.route_e_trk_1
        ];

        return routes;
    }
    // ---- END get_routes() ----

    /**
     * get_interlocking_status()
     * @summary returns the status of the interlocking that would be
needed by the ReactJS Components
     *
     * @description All the information that is returned here is what
```

```
is needed by the ReactJS Component
     * for the interlocking that is need to draw the interlocking to
the screen
     *
     * @returns Object with the status of the interlocking
     */
    get_interlocking_status() {
        let status = {
            sw_21: this.sw_21,
            sw_32: this.sw_32,
            occupied: this.get_occupied(),
            routes: this.get_routes()
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_Valley;
```

```
/**
 * @file MainLine.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for the entire Pannel
 *
 * @description Extends the React Component Class and is the UI of the
entrie Pannel, this component
 * contains all the other components, and holds the functions that
allows each component to
 * change the back end class for each enterlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import My Own Clock Class which takes care of trains running
import Clock from '../../scripts/Trains/clock.js';
// To Control All The Trains
import MaineLine_CTC from '../../scripts/CTC/mainLine_ctc.js';
// Import My Train Class
import Train from '../../scripts/Trains/train.js';

// Import the Main Line Components
import MainLineTracks from '../Panel/Main_Line/MainLineTracks.jsx';
import Hilburn from '../Panel/Main_Line/Hilburn.jsx';
import SF from '../Panel/Main_Line/SF.jsx';
import WC from '../Panel/Main_Line/WC.jsx';
import RidgewoodJunction from '../Panel/Main_Line/
RidgewoodJunction.jsx';
import Suscon from '../Panel/Main_Line/Suscon.jsx';
import Mill from '../Panel/Main_Line/Mill.jsx';
import WestSecaucus from '../Panel/Main_Line/WestSecaucus.jsx';
import Laurel from '../Panel/Main_Line/Laurel.jsx';

// Import the Bergen County Line Components
import BergenTracks from '../Panel/Bergen_County_Line/
BergenTracks.jsx';
import BT from '../Panel/Bergen_County_Line/BT.jsx';
import PascackJunction from '../Panel/Bergen_County_Line/
PascackJct.jsx';
import HX from '../Panel/Bergen_County_Line/HX.jsx';

// Import the Southern Tier Line Components
import SouthernTierTracks from '../Panel/Southern_Tier_Line/
SouthernTierTracks.jsx';
import Sparrow from '../Panel/Southern_Tier_Line/Sparrow.jsx';
import PA from '../Panel/Southern_Tier_Line/PA.jsx';
import Port from '../Panel/Southern_Tier_Line/Port.jsx';
import BC from '../Panel/Southern_Tier_Line/BC.jsx';
import OV from '../Panel/Southern_Tier_Line/OV.jsx';
```

```
import Howells from '../Panel/Southern_Tier_Line/Howells.jsx';
import Hall from '../Panel/Southern_Tier_Line/Hall.jsx';
import HudsonJunction from '../Panel/Southern_Tier_Line/
HudsonJunction.jsx';
import CentralValley from '../Panel/Southern_Tier_Line/
CentralValley.jsx';
import Harriman from '../Panel/Southern_Tier_Line/Harriman.jsx';
import Sterling from '../Panel/Southern_Tier_Line/Sterling.jsx';


// Create A new Clock for the Game
var clock = new Clock();
// Create the CTC controler for the game, passing it the clock we
created above
var ctc = new MaineLine_CTC(clock);

// Initialize the clock
clock.startClock;


setTimeout(function(){
    ctc.add_train(new Train("[E] 49", "3_yardEast_port", "EAST", 10));
    ctc.add_train(new Train("3", "3_laurel_westEnd", "WEST", 10));
    ctc.add_train(new Train("1", "1_laurel_westEnd", "WEST", 10));
    ctc.add_train(new Train("2", "2_laurel_westEnd", "WEST", 10));
    ctc.add_train(new Train("4", "4_laurel_westEnd", "WEST", 10));
    ctc.add_train(new Train("50", "3_yardHilburn_sf", "EAST", 10));
    ctc.add_train(new Train("[E] SU100", "1_bingo_sparrow", "EAST",
10));
    ctc.occupy_blocks();
}, 1500);


/**
 * The React JSX Component Class for the entire Maine Line Dispatcher
Panel This class is a JSX React Component for the Maine Line Dispatch
Panel,
 * this will control all the other components that make up the pannel.
This also controls the functions that allow each component to change
their respected
 * back end functions.
 */
class MainLine extends Component {

    /**
     * constructor()
     * @summary The Constructor for the MainLine JSX class.
     *
     * All this does is set that state for every thing getting the
information fro the CTC controller, the state here
```

```
     * is used to send to the child components so they can render the
correct information
     *
     * @param props, Required as park of ReactJS, but is not used here
     */
    constructor(props) {
        super(props);
        /**
         * State
         * @summary Object that holds the state or status information
for the component
         *
         * This object holds all the information for everything on the
pannel that is required to display the routes
         * correctly
         */
        this.state = {
            // Southern Tier Interlockings Status
            status_sparrow:
ctc.get_sparrow().get_interlocking_status(),
            status_pa: ctc.get_pa().get_interlocking_status(),
            status_port: ctc.get_port().get_interlocking_status(),
            status_bc: ctc.get_bc().get_interlocking_status(),
            status_ov: ctc.get_ov().get_interlocking_status(),
            status_howells:
ctc.get_howells().get_interlocking_status(),
            status_hall: ctc.get_hall().get_interlocking_status(),
            status_hudson: ctc.get_hudson().get_interlocking_status(),
            status_valley: ctc.get_valley().get_interlocking_status(),
            status_harriman:
ctc.get_harriman().get_interlocking_status(),
            status_sterling:
ctc.get_sterling().get_interlocking_status(),

            // Main Line Interlockings Status
            status_hilburn:
ctc.get_hilburn().get_interlocking_status(),
            status_sf: ctc.get_sf().get_interlocking_status(),
            status_wc: ctc.get_wc().get_interlocking_status(),
            status_ridgewood:
ctc.get_ridgewood().get_interlocking_status(),
            status_suscon: ctc.get_suscon().get_interlocking_status(),
            status_mill: ctc.get_mill().get_interlocking_status(),
            status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status(),
            status_laurel: ctc.get_laurel().get_interlocking_status(),

            // Bergen County Interlocking Status
            status_bt: ctc.get_bt().get_interlocking_status(),
            status_pascack:
```

```
ctc.get_pascack().get_interlocking_status(),
            status_hx: ctc.get_hx().get_interlocking_status(),

            // Main Line Tracks & Symbols
            status_mainLine: ctc.get_mainLine_blocks_status(),
            symbols_mailLine: ctc.get_mainLine_symbols(),
            // Bergen County Track & Symbols
            status_bergenLine: ctc.get_bergen_blocks_status(),
            symbols_bergenLine: ctc.get_bergen_symbols(),
            // Southern Tier Tracks & Symbols
            status_tier: ctc.get_tier_block_status(),
            symbols_tier: ctc.get_tier_symbols()
        };
    }

    /**
     * update_blocks()
     * @summary This function is called every 0.5 Seconds and updates
all the tracks blocks
     *
     * @description When this function is called it call 2 functions
in the CTC controler class.
     * The first one will check find all the routes at each
interlocking and set the correct
     * next block to routed, so the route can be displayed on the
pannel
     * The second will get all the trains current locations and make
those blocks as occupied,
     * to show the correct location of each train on the pannel
     */
    update_blocks = () => {
        // Update All The Routes
        ctc.update_route_blocks();
        // Update All The Trains
        ctc.occupy_blocks();
        // Set the Component State
        this.setState({
            // Main Line Tracks & Symbols
            status_mainLine: ctc.get_mainLine_blocks_status(),
            symbols_mailLine: ctc.get_mainLine_symbols(),
            // Bergen County Tracks & Symbols
            status_bergenLine: ctc.get_bergen_blocks_status(),
            symbols_bergenLine: ctc.get_bergen_symbols(),
            // Southern Tier Tracks & Symbols
            status_tier: ctc.get_tier_block_status(),
            symbols_tier: ctc.get_tier_symbols()
        });
    }
    // ---- END update_blocks() ----
```

```
    /**
     * update_trains()
     * @summary This function is called every 2 Seconds and updates
all the Trains locations
     *
     * @description When this function is called it will call 2
functions in the CTC controler
     * The first function updates the trains allowing them to move to
the next location if the
     * correct time has be spend in their current block
     * The second function updates the interlockings showing if they
are occupied or cleared if the
     * correct time has passed
     */
    update_trains = () => {
        // Allow trains to update their location if possible
        ctc.update_trains();
        // Update the interlockings
        ctc.update_interlockings();
        // Set The State of the Component
        this.setState({
            // Main Line Tracks & Symbols
            status_mainLine: ctc.get_mainLine_blocks_status(),
            symbols_mailLine: ctc.get_mainLine_symbols(),
            // Bergen County Tracks & Symbols
            status_bergenLine: ctc.get_bergen_blocks_status(),
            symbols_bergenLine: ctc.get_bergen_symbols(),
            // Southern Tier Tracks & Symbols
            status_tier: ctc.get_tier_block_status(),
            symbols_tier: ctc.get_tier_symbols(),

            // Southern Tier Interlockings
            status_sparrow:
ctc.get_sparrow().get_interlocking_status(),
            status_pa: ctc.get_pa().get_interlocking_status(),
            status_port: ctc.get_port().get_interlocking_status(),
            status_bc: ctc.get_bc().get_interlocking_status(),
            status_ov: ctc.get_ov().get_interlocking_status(),
            status_howells:
ctc.get_howells().get_interlocking_status(),
            status_hall: ctc.get_hall().get_interlocking_status(),
            status_hudson: ctc.get_hudson().get_interlocking_status(),
            status_valley: ctc.get_valley().get_interlocking_status(),
            status_harriman:
ctc.get_harriman().get_interlocking_status(),
            status_sterling:
ctc.get_sterling().get_interlocking_status(),

            // Main Line Interlockings
            status_hilburn:
```

```
ctc.get_hilburn().get_interlocking_status(),
            status_sf: ctc.get_sf().get_interlocking_status(),
            status_wc: ctc.get_wc().get_interlocking_status(),
            status_ridgewood:
ctc.get_ridgewood().get_interlocking_status(),
            status_suscon: ctc.get_suscon().get_interlocking_status(),
            status_mill: ctc.get_mill().get_interlocking_status(),
            status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status(),
            status_laurel: ctc.get_laurel().get_interlocking_status(),

            // Bergen County Interlockings
            status_bt: ctc.get_bt().get_interlocking_status(),
            status_pascack:
ctc.get_pascack().get_interlocking_status(),
            status_hx: ctc.get_hx().get_interlocking_status(),
        });
    }

    /**
     * componentDidMount()
     * @summary ReactJS function that allows you do set the intervals
for when certin functions are called
     *
     * @description This function sets the intervals for each function
that is called repeadely after a amount of time
     * Will call the update_blocks() function every 0.5 Seconds
     * Will call the update_trains() function every 2 Seconds
     */
    componentDidMount() {
        // update_blocks() Interval [0.5 Seconds]
        this.interval_update_blocks = setInterval(() =>
this.update_blocks(), 500);
        // update_trains() Interval [2 Seconds]
        this.interval_update_trains = setInterval(() =>
this.update_trains(), 2000);
    }
    // ---- END componentDidMount()

    /**
     * componentWillUnmount()
     * @summary ReactJS function that removes the intervals, this is
never called in this program
     *
     * @description This function deletes the intervals that are used
to update the blocks & trains
     * This is never called in this program
     */
    componentWillUnmount() {
        clearInterval(this.interval_update_blocks);
```

```
        clearInterval(this.interval_update_trains);
    }
    // ---- END componentWillUnmount() ----

    /**
     * render()
     * @summary standard React function that draws all the other
interlockings and track components to the screen
     *
     * @description This will draw all the components to the screen to
assemble the pannel, it also passes all the function
     * and information to each components through their properties or
(props)
     */
    render() {
        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* SOUTHERN TIER SECTION */}
                {/* Tracks */}
                <SouthernTierTracks
                    blocks={this.state.status_tier}
                    symbols={this.state.symbols_tier}
                />
                {/* Interlockings */}
                <Sparrow
                    status={this.state.status_sparrow}
                    click_sig_2w_1={this.sparrow_click_sig_2w_1}
                    click_sig_2w_2={this.sparrow_click_sig_2w_2}
                    click_sig_2w_3={this.sparrow_click_sig_2w_3}
                    click_sig_2e={this.sparrow_click_sig_2e}
                    throw_sw_1={this.sparrow_throw_sw_1}
                    throw_sw_3={this.sparrow_throw_sw_3}
                />
                <PA
                    status={this.state.status_pa}
                    click_sig_2w_1={this.pa_click_sig_2w_1}
                    click_sig_2w_2={this.pa_click_sig_2w_2}
                    click_sig_4w={this.pa_click_sig_4w}
                    click_sig_2e={this.pa_click_sig_2e}
                    click_sig_4e={this.pa_click_sig_4e}
                    throw_sw_1={this.pa_throw_sw_1}
                    throw_sw_3={this.pa_throw_sw_3}
                />
                <Port
                    status={this.state.status_port}
                    click_sig_2w={this.port_click_sig_2w}
                    click_sig_2e_1={this.port_click_sig_2e_1}
                    click_sig_2e_2={this.port_click_sig_2e_2}
```

```
    throw_sw_1={this.port_throw_sw_1}
/>
<BC
    status={this.state.status_bc}
    click_sig_2w={this.bc_click_sig_2w}
    click_sig_2e={this.bc_click_sig_2e}
    click_sig_4e={this.bc_click_sig_4e}
    throw_sw_1={this.bc_throw_sw_1}
/>
<OV
    status={this.state.status_ov}
    click_sig_2w={this.ov_click_sig_2w}
    click_sig_2ws={this.ov_click_sig_2ws}
    click_sig_2e={this.ov_click_sig_2e}
    throw_sw_1={this.ov_throw_sw_1}
/>
<Howells
    status={this.state.status_howells}
    click_sig_2w={this.howells_click_sig_2w}
    click_sig_2e={this.howells_click_sig_2e}
    click_sig_2es={this.howells_click_sig_2es}
    throw_sw_3={this.howells_throw_sw_3}
/>
<Hall
    status={this.state.status_hall}
    click_sig_2w={this.hall_click_sig_2w}
    click_sig_4w={this.hall_click_sig_4w}
    click_sig_2e={this.hall_click_sig_2e}
    click_sig_4e={this.hall_click_sig_4e}
    throw_sw_1={this.hall_throw_sw_1}
/>
<HudsonJunction
    status={this.state.status_hudson}
    click_sig_2w={this.hudson_click_sig_2w}
    click_sig_2ws={this.hudson_click_sig_2ws}
    click_sig_2e={this.hudson_click_sig_2e}
    click_sig_2es={this.hudson_click_sig_2es}
    throw_sw_1={this.hudson_throw_sw_1}
    throw_sw_3={this.hudson_throw_sw_3}
/>
<CentralValley
    status={this.state.status_valley}
    click_sig_1w={this.valley_click_sig_1w}
    click_sig_2w={this.valley_click_sig_2w}
    click_sig_1e={this.valley_click_sig_1e}
    throw_sw_21={this.valley_throw_sw_21}
/>
<Harriman
    status={this.state.status_harriman}
    click_sig_1w={this.harriman_click_sig_1w}
```

```
    click_sig_1e={this.harriman_click_sig_1e}
    click_sig_2e={this.harriman_click_sig_2e}
    click_sig_3e={this.harriman_click_sig_3e}
    throw_sw_21={this.harriman_throw_sw_21}
    throw_sw_32={this.harriman_throw_sw_32}
/>
<Sterling
    status={this.state.status_sterling}
    click_sig_2w={this.sterling_click_sig_2w}
    click_sig_2ws={this.sterling_click_sig_2ws}
    click_sig_1e={this.sterling_click_sig_1e}
    throw_sw_21={this.sterling_throw_sw_21}
/>


{/* BERGEN COUNTY LINE SECTION */}
{/* Tracks */}
<BergenTracks
    blocks={this.state.status_bergenLine}
    symbols={this.state.symbols_bergenLine}
/>
{/* Interlockings */}
<BT
    status={this.state.status_bt}
    click_sig_2w1={this.bt_click_sig_2w1}
    click_sig_2w2={this.bt_click_sig_2w2}
    click_sig_4w={this.bt_click_sig_4w}
    click_sig_2e={this.bt_click_sig_2e}
    click_sig_4e={this.bt_click_sig_4e}
    throw_sw_1={this.bt_throw_sw_1}
    throw_sw_3={this.bt_throw_sw_3}
    throw_sw_5={this.bt_throw_sw_5}
/>
<PascackJunction
    status={this.state.status_pascack}
    click_sig_2w={this.pascack_click_sig_2w}
    click_sig_4w={this.pascack_click_sig_4w}
    click_sig_2e={this.pascack_click_sig_2e}
    click_sig_4e={this.pascack_click_sig_4e}
    throw_sw_1={this.pascack_throw_sw_1}
    throw_sw_3={this.pascack_throw_sw_3}
/>
<HX
    status={this.state.status_hx}
    click_sig_2w1={this.hx_click_sig_2w1}
    click_sig_2w2={this.hx_click_sig_2w2}
    click_sig_2w3={this.hx_click_sig_2w3}
    click_sig_4w={this.hx_click_sig_4w}
    click_sig_2e={this.hx_click_sig_2e}
    click_sig_4e={this.hx_click_sig_4e}
```

```
    throw_sw_1={this.hx_throw_sw_1}
    throw_sw_3={this.hx_throw_sw_3}
    throw_sw_5={this.hx_throw_sw_5}
/>


{/* MAIN LINE SECTION */}
{/* Tracks */}
<MainLineTracks
    blocks={this.state.status_mainLine}
    symbols={this.state.symbols_mailLine}
/>
{/* Interlockings */}
<Hilburn
    status={this.state.status_hilburn}
    click_sig_2w_1={this.hilburn_click_sig_2w_1}
    click_sig_2w_2={this.hilburn_click_sig_2w_2}
    click_sig_2e={this.hilburn_click_sig_2e}
    click_sig_4e={this.hilburn_click_sig_4e}
    throw_sw_1={this.hilburn_throw_sw_1}
/>
<SF
    status={this.state.status_sf}
    click_sig_2w={this.sf_click_sig_2w}
    click_sig_4w={this.sf_click_sig_4w}
    click_sig_2e={this.sf_click_sig_2e}
    click_sig_4e_1={this.sf_click_sig_4e_1}
    click_sig_4e_2={this.sf_click_sig_4e_2}
    throw_sw_1={this.sf_throw_sw_1}
    throw_sw_3={this.sf_throw_sw_3}
/>
<WC
    status={this.state.status_wc}
    click_sig_2w_1={this.wc_click_sig_2w_1}
    click_sig_2w_2={this.wc_click_sig_2w_2}
    click_sig_4w={this.wc_click_sig_4w}
    click_sig_2e_1={this.wc_click_sig_2e_1}
    click_sig_2e_2={this.wc_click_sig_2e_2}
    click_sig_4e={this.wc_click_sig_4e}
    throw_sw_1={this.wc_throw_sw_1}
    throw_sw_3={this.wc_throw_sw_3}
    throw_sw_5={this.wc_throw_sw_5}
    throw_sw_7={this.wc_throw_sw_7}
/>
<RidgewoodJunction
    status={this.state.status_ridgewood}
    click_sig_2w_1={this.ridgewood_click_sig_2w_1}
    click_sig_2w_2={this.ridgewood_click_sig_2w_2}
    click_sig_4w={this.ridgewood_click_sig_4w}
    click_sig_6w={this.ridgewood_click_sig_6w}
```

```
    click_sig_2e={this.ridgewood_click_sig_2e}
    click_sig_4e={this.ridgewood_click_sig_4e}
    click_sig_6e={this.ridgewood_click_sig_6e}
    throw_sw_1={this.ridgewood_throw_sw_1}
    throw_sw_3={this.ridgewood_throw_sw_3}
    throw_sw_5={this.ridgewood_throw_sw_5}
    throw_sw_7={this.ridgewood_throw_sw_7}
    throw_sw_9={this.ridgewood_throw_sw_9}
/>
<Suscon
    status={this.state.status_suscon}
    click_sig_2w={this.suscon_click_sig_2w}
    click_sig_2e={this.suscon_click_sig_2e}
    click_sig_4w={this.suscon_click_sig_4w}
    click_sig_4e={this.suscon_click_sig_4e}
    throw_sw_1={this.suscon_throw_sw_1}
    throw_sw_3={this.suscon_throw_sw_3}
/>
<Mill
    status={this.state.status_mill}
    click_sig_2w={this.mill_click_sig_2w}
    click_sig_2e={this.mill_click_sig_2e}
    click_sig_4w={this.mill_click_sig_4w}
    click_sig_4e={this.mill_click_sig_4e}
    throw_sw_1={this.mill_throw_sw_1}
    throw_sw_3={this.mill_throw_sw_3}
/>
<WestSecaucus
    status={this.state.status_westSecaucus}
    click_sig_2w={this.westSecaucus_click_sig_2w}
    click_sig_2e={this.westSecaucus_click_sig_2e}
    click_sig_4w={this.westSecaucus_click_sig_4w}
    click_sig_4e={this.westSecaucus_click_sig_4e}
    throw_sw_1={this.westSecaucus_throw_sw_1}
    throw_sw_3={this.westSecaucus_throw_sw_3}
/>
<Laurel
    status={this.state.status_laurel}
    click_sig_2w={this.laurel_click_sig_2w}
    click_sig_4w={this.laurel_click_sig_4w}
    click_sig_8w={this.laurel_click_sig_8w}
    click_sig_10w={this.laurel_click_sig_10w}
    click_sig_6e={this.laurel_click_sig_6e}
    click_sig_12e={this.laurel_click_sig_12e}
    click_sig_4e={this.laurel_click_sig_4e}
    click_sig_8e={this.laurel_click_sig_8e}
    throw_sw_1={this.laurel_throw_sw_1}
    throw_sw_3={this.laurel_throw_sw_3}
    throw_sw_7={this.laurel_throw_sw_7}
    throw_sw_9={this.laurel_throw_sw_9}
```

```
                    throw_sw_11={this.laurel_throw_sw_11}
                    throw_sw_13={this.laurel_throw_sw_13}
                />
            </div>
        );
    }
    // ---- END render() ----

    //
-------------------------------------------------------------------------
----------------------
    // All of the following function are the only way to get the event
handers (below) and passed
    // into each component to access the fuctions in the CTC
controler, it's a very cumbersum way
    // to accomplish this, but its the only way I was able to find. I
would like to change this
    // one day in the future if I find a more streamlined way
    //
-------------------------------------------------------------------------
----------------------

    /* Bergen County Line Event Handlers */
    /* Functions for the HX Interlocking */
    /**
     * hx_click_sig_2w1()
     * @summary The event handler for Signal #2w-1
     */
    hx_click_sig_2w1 = () => {
        // Get the backend function for corresponding signal
        // Passing reference the next blocks
        ctc.get_hx().click_sig_2w1(
            this.state.status_bergenLine.block_pascack_hx_1,
            this.state.status_bergenLine.block_pascack_hx_2
        );
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_click_sig_2w1() ----

    /**
     * hx_click_sig_2w2()
     * @summary The event handler for the Signal #2w2
     */
    hx_click_sig_2w2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hx().click_sig_2w2(
            this.state.status_bergenLine.block_pascack_hx_1,
```

```
            this.state.status_bergenLine.block_pascack_hx_2
        );
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_click_sig_2w2() ----

    /**
     * hx_click_sig_2w3()
     * @summary The event handler for the Signal #2w3
     */
    hx_click_sig_2w3 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hx().click_sig_2w3(
            this.state.status_bergenLine.block_pascack_hx_1,
            this.state.status_bergenLine.block_pascack_hx_2
        );
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_click_sig_2w3() ----

    /**
     * hx_click_sig_4w()
     * @summary The event handler for the Signal #4w
     */
    hx_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hx().click_sig_4w(
            this.state.status_bergenLine.block_pascack_hx_2
        );
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_click_sig_4w() ----

    /**
     * hx_click_sig_2e()
     * @summary The event handler for the Signal 2e
     */
    hx_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hx().click_sig_2e(
            this.state.status_bergenLine.block_hx_laurel_1,
```

```
            this.state.status_bergenLine.block_hx_croxton_2,
            this.state.status_bergenLine.block_hx_croxton_1
        );
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_click_sig_2e() ----

    /**
     * hx_click_sig_4e()
     * @summary The event handler for the Signal 4e
     */
    hx_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hx().click_sig_4e(
            this.state.status_bergenLine.block_hx_laurel_1,
            this.state.status_bergenLine.block_hx_laurel_2,
            this.state.status_bergenLine.block_hx_croxton_2,
            this.state.status_bergenLine.block_hx_croxton_1
        );
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_click_sig_4e() ----

    /**
     * hx_throw_sw_1()
     * @summary The event handler for switch #1
     */
    hx_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hx().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_throw_sw_1() ----

    /**
     * hx_throw_sw_3()
     * @summary The event handler for switch #3
     */
    hx_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hx().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_hx:
```

```
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_throw_sw_3() ----

    /**
     * hx_throw_sw_5()
     * @summary The event handler for switch #5
     */
    hx_throw_sw_5 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hx().throw_sw_5();
        // Set the state of the Interlocking
        this.setState({status_hx:
ctc.get_hx().get_interlocking_status()});
    }
    // ---- END hx_throw_sw_5() ----
    /* END Functions for the HX Interlocking */


    /* Functions for the Pascack Junction Interlocking */
    /**
     * pascack_click_sig_2w()
     * @summary Event handler for the signal #2w
     */
    pascack_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pascack().click_sig_2w(
            this.state.status_bergenLine.block_bt_pascack_1,
            this.state.status_bergenLine.block_bt_pascack_2
        );
        // Set the state of the Interlocking
        this.setState({status_pascack:
ctc.get_pascack().get_interlocking_status()});
    }
    // ---- END pascack_click_sig_2w() ----

    /**
     * pascack_click_sig_4w()
     * @summary Event handler for the signal #4w
     */
    pascack_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pascack().click_sig_4w(
            this.state.status_bergenLine.block_bt_pascack_1,
            this.state.status_bergenLine.block_bt_pascack_2
        );
        // Set the state of the Interlocking
        this.setState({status_pascack:
```

```
ctc.get_pascack().get_interlocking_status()});
    }
    // ---- END pascack_click_sig_4w() ----

    /**
     * pascack_click_sig_2e()
     * @summary Event handler for the signal #2e
     */
    pascack_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pascack().click_sig_2e(
            this.state.status_bergenLine.block_pascack_hx_1,
            this.state.status_bergenLine.block_pascack_hx_2
        );
        // Set the state of the Interlocking
        this.setState({status_pascack:
ctc.get_pascack().get_interlocking_status()});
    }
    // ---- END pascack_click_sig_2e() ----

    /**
     * pascack_click_sig_4e()
     * @summary Event handler for the signal #4e
     */
    pascack_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pascack().click_sig_4e(
            this.state.status_bergenLine.block_pascack_hx_1,
            this.state.status_bergenLine.block_pascack_hx_2
        );
        // Set the state of the Interlocking
        this.setState({status_pascack:
ctc.get_pascack().get_interlocking_status()});
    }
    // ---- END pascack_click_sig_4e() ----

    /**
     * pascack_throw_sw_1()
     * @summary The event handler for switch #1
     */
    pascack_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_pascack().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_pascack:
ctc.get_pascack().get_interlocking_status()});
    }
    // ---- END pascack_throw_sw_1() ----
```

```
    /**
     * pascack_throw_sw_3()
     * @summary The event handler for switch #3
     */
    pascack_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_pascack().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_pascack:
ctc.get_pascack().get_interlocking_status()});
    }
    // ---- END pascack_throw_sw_1() ----
    /* END Functions for the Pascack Junction Interlocking */


    /* Functions for the BT Interlocking */
    /**
     * bt_click_sig_2w1()
     * @summary Event handler for the signal #2w1
     */
    bt_click_sig_2w1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bt().click_sig_2w1(
            this.state.status_bergenLine.block_ridgewood_bt_1,
            this.state.status_bergenLine.block_ridgewood_bt_2
        );
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_click_sig_2w1() ----

    /**
     * bt_click_sig_2w2()
     * @summary Event handler for the signal #2w2
     */
    bt_click_sig_2w2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bt().click_sig_2w2(
            this.state.status_bergenLine.block_ridgewood_bt_1,
            this.state.status_bergenLine.block_ridgewood_bt_2
        );
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_click_sig_2w1() ----
```

```
    /**
     * bt_click_sig_4w()
     * @summary Event handler for the signal #4
     */
    bt_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bt().click_sig_4w(
            this.state.status_bergenLine.block_ridgewood_bt_1,
            this.state.status_bergenLine.block_ridgewood_bt_2
        );
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_click_sig_2w1() ----

    /**
     * bt_click_sig_2e()
     * @summary Event handler for the signal #2e
     */
    bt_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bt().click_sig_2e(
            this.state.status_bergenLine.block_bt_pascack_1,
            this.state.status_bergenLine.block_bt_pascack_2,
            this.state.status_bergenLine.block_bt_nysw
        );
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_click_sig_2w1() ----

    /**
     * bt_click_sig_4e()
     * @summary Event handler for the signal #4e
     */
    bt_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bt().click_sig_4e(
            this.state.status_bergenLine.block_bt_pascack_1,
            this.state.status_bergenLine.block_bt_pascack_2,
            this.state.status_bergenLine.block_bt_nysw
        );
        // Set the state of the Interlocking
        this.setState({status_bt:
```

```
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_click_sig_2w1() ----

    /**
     * bt_throw_sw_1()
     * @summary The event handler for switch #1
     */
    bt_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_bt().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_throw_sw_1() ----

    /**
     * bt_throw_sw_3()
     * @summary The event handler for switch #3
     */
    bt_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_bt().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_throw_sw_3() ----

    /**
     * bt_throw_sw_5()
     * @summary The event handler for switch #5
     */
    bt_throw_sw_5 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_bt().throw_sw_5();
        // Set the state of the Interlocking
        this.setState({status_bt:
ctc.get_bt().get_interlocking_status()});
    }
    // ---- END bt_throw_sw_5() ----
    /* END Functions for the BT Interlocking */
    /* END Bergen County Line Event Handlers */




    /* Southern Tier Event Handlers */
    /* Functions for CP Sparrow */
```

```
    /**
     * sparrow_click_sig_2w_1()
     * @summary The event handler for Signal #2w_1
     */
    sparrow_click_sig_2w_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sparrow().click_sig_2w_1(
            this.state.status_tier.block_bingo_sparrow
        );
        // Set the state of the Interlocking
        this.setState({status_sparrow:
ctc.get_sparrow().get_interlocking_status()});
    }
    // ---- END sparrow_click_sig_2w_1() ----

    /**
     * sparrow_click_sig_2w_2()
     * @summary The event handler for Signal #2w_2
     */
    sparrow_click_sig_2w_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sparrow().click_sig_2w_2(
            this.state.status_tier.block_bingo_sparrow
        );
        // Set the state of the Interlocking
        this.setState({status_sparrow:
ctc.get_sparrow().get_interlocking_status()});
    }
    // ---- END sparrow_click_sig_2w_2() ----

    /**
     * sparrow_click_sig_2w_3()
     * @summary The event handler for Signal #2w_3
     */
    sparrow_click_sig_2w_3 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sparrow().click_sig_2w_3(
            this.state.status_tier.block_bingo_sparrow
        );
        // Set the state of the Interlocking
        this.setState({status_sparrow:
ctc.get_sparrow().get_interlocking_status()});
    }
    // ---- END sparrow_click_sig_2w_3() ----

    /**
     * sparrow_click_sig_2e()
```

```
     * @summary The event handler for Signal #2e
     */
    sparrow_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sparrow().click_sig_2e(
            this.state.status_tier.block_sparrow_pa_1,
            this.state.status_tier.block_sparrow_pa_2,
            this.state.status_tier.block_sparrow_cripple
        );
        // Set the state of the Interlocking
        this.setState({status_sparrow:
ctc.get_sparrow().get_interlocking_status()});
    }
    // ---- END sparrow_click_sig_2e() ----

    /**
     * sparrow_throw_sw_1()
     * @summary The event handler for switch #1
     */
    sparrow_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_sparrow().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_sparrow:
ctc.get_sparrow().get_interlocking_status()});
    }
    // ---- END sparrow_throw_sw_1() ----

    /**
     * sparrow_throw_sw_3()
     * @summary The event handler for switch #3
     */
    sparrow_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_sparrow().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_sparrow:
ctc.get_sparrow().get_interlocking_status()});
    }
    // ---- END sparrow_throw_sw_3() ----
    /* END Functions for CP Sparrow */


    /* Functions for CP PA */
    /**
     * pa_click_sig_2w_1()
     * @summary The event handler for Signal #2w_1
     */
    pa_click_sig_2w_1 = () => {
```

```
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pa().click_sig_2w_1(
            this.state.status_tier.block_sparrow_pa_1,
            this.state.status_tier.block_sparrow_pa_2,
            this.state.status_tier.block_buckleys_west
        );
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_click_sig_2w_1() ----

    /**
     * pa_click_sig_2w_2()
     * @summary The event handler for Signal #2w_2
     */
    pa_click_sig_2w_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pa().click_sig_2w_2(
            this.state.status_tier.block_sparrow_pa_1,
            this.state.status_tier.block_sparrow_pa_2,
            this.state.status_tier.block_buckleys_west
        );
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_click_sig_2w_2() ----

    /**
     * pa_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    pa_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pa().click_sig_4w(
            this.state.status_tier.block_sparrow_pa_2,
            this.state.status_tier.block_buckleys_west
        );
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_click_sig_4w() ----

    /**
     * pa_click_sig_2e()
```

```
     * @summary The event handler for Signal #2e
     */
    pa_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pa().click_sig_2e(
            this.state.status_tier.block_pa_port_1,
            this.state.status_tier.block_port_yard_west
        );
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_click_sig_2e() ----

    /**
     * pa_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    pa_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_pa().click_sig_4e(
            this.state.status_tier.block_pa_port_1,
            this.state.status_tier.block_pa_bc_2,
            this.state.status_tier.block_port_yard_west
        );
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_click_sig_4e() ----

    /**
     * pa_throw_sw_1()
     * @summary The event handler for switch #1
     */
    pa_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_pa().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_throw_sw_1() ----

    /**
     * pa_throw_sw_3()
     * @summary The event handler for switch #3
     */
```

```
    pa_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_pa().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_pa:
ctc.get_pa().get_interlocking_status()});
    }
    // ---- END pa_throw_sw_3() ----
    /* END Functions for CP PA */


    /* Functions for CP Port */
    /**
     * pa_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    port_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_port().click_sig_2w(
            this.state.status_tier.block_pa_port_1,
            this.state.status_tier.block_port_yard_east
        );
        // Set the state of the Interlocking
        this.setState({status_port:
ctc.get_port().get_interlocking_status()});
    }
    // ---- END port_click_sig_2w() ----

    /**
     * pa_click_sig_2e_1()
     * @summary The event handler for Signal #2e_1
     */
    port_click_sig_2e_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_port().click_sig_2e_1(
            this.state.status_tier.block_port_bc_1
        );
        // Set the state of the Interlocking
        this.setState({status_port:
ctc.get_port().get_interlocking_status()});
    }
    // ---- END port_click_sig_2e_1() ----

    /**
     * pa_click_sig_2e_2()
     * @summary The event handler for Signal #2e_2
     */
    port_click_sig_2e_2 = () => {
```

```
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_port().click_sig_2e_2(
            this.state.status_tier.block_port_bc_1
        );
        // Set the state of the Interlocking
        this.setState({status_port:
ctc.get_port().get_interlocking_status()});
    }
    // ---- END port_click_sig_2e_2() ----

    /**
     * port_throw_sw_1()
     * @summary The event handler for switch #1
     */
    port_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_port().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_port:
ctc.get_port().get_interlocking_status()});
    }
    // ---- END port_throw_sw_1() ----
    /* END Functions for CP Port */


    /* Functions for CP BC */
    /**
     * bc_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    bc_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bc().click_sig_2w(
            this.state.status_tier.block_port_bc_1,
            this.state.status_tier.block_pa_bc_2
        );
        // Set the state of the Interlocking
        this.setState({status_bc:
ctc.get_bc().get_interlocking_status()});
    }
    // ---- END port_click_sig_2w() ----

    /**
     * bc_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    bc_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
```

```
        // Passing reference the next blocks
        ctc.get_bc().click_sig_2e(
            this.state.status_tier.block_bc_ov_1
        );
        // Set the state of the Interlocking
        this.setState({status_bc:
ctc.get_bc().get_interlocking_status()});
    }
    // ---- END port_click_sig_2e() ----

    /**
     * bc_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    bc_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_bc().click_sig_4e(
            this.state.status_tier.block_bc_ov_1
        );
        // Set the state of the Interlocking
        this.setState({status_bc:
ctc.get_bc().get_interlocking_status()});
    }
    // ---- END port_click_sig_4e() ----

    /**
     * bc_throw_sw_1()
     * @summary The event handler for switch #1
     */
    bc_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_bc().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_bc:
ctc.get_bc().get_interlocking_status()});
    }
    // ---- END bc_throw_sw_1() ----
    /* END Functions for CP BC */


    /* Functions for CP OV */
    /**
     * ov_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    ov_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ov().click_sig_2w(
```

```
            this.state.status_tier.block_bc_ov_1
        );
        // Set the state of the Interlocking
        this.setState({status_ov:
ctc.get_ov().get_interlocking_status()});
    }
    // ---- END ov_click_sig_2w() ----

    /**
     * ov_click_sig_2ws()
     * @summary The event handler for Signal #2ws
     */
    ov_click_sig_2ws = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ov().click_sig_2ws(
            this.state.status_tier.block_bc_ov_1
        );
        // Set the state of the Interlocking
        this.setState({status_ov:
ctc.get_ov().get_interlocking_status()});
    }
    // ---- END ov_click_sig_2ws() ----

    /**
     * ov_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    ov_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ov().click_sig_2e(
            this.state.status_tier.block_ov_howells_1,
            this.state.status_tier.block_ov_howells_2
        );
        // Set the state of the Interlocking
        this.setState({status_ov:
ctc.get_ov().get_interlocking_status()});
    }
    // ---- END ov_click_sig_2e() ----

    /**
     * ov_throw_sw_1()
     * @summary The event handler for switch #1
     */
    ov_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_ov().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_ov:
```

```
ctc.get_ov().get_interlocking_status()});
    }
    // ---- END ov_throw_sw_1() ----
    /* END Functions for CP OV */


    /* Functions for CP Howells */
    /**
     * howells_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    howells_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_howells().click_sig_2w(
            this.state.status_tier.block_ov_howells_1,
            this.state.status_tier.block_ov_howells_2
        );
        // Set the state of the Interlocking
        this.setState({status_howells:
ctc.get_howells().get_interlocking_status()});
    }
    // ---- END howells_click_sig_2w() ----

    /**
     * howells_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    howells_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_howells().click_sig_2e(
            this.state.status_tier.block_howells_hall_1
        );
        // Set the state of the Interlocking
        this.setState({status_howells:
ctc.get_howells().get_interlocking_status()});
    }
    // ---- END howells_click_sig_2e() ----

    /**
     * howells_click_sig_2es()
     * @summary The event handler for Signal #2es
     */
    howells_click_sig_2es = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_howells().click_sig_2es(
            this.state.status_tier.block_howells_hall_1
        );
```

```
        // Set the state of the Interlocking
        this.setState({status_howells:
ctc.get_howells().get_interlocking_status()});
    }
    // ---- END howells_click_sig_2es() ----

    /**
     * howells_throw_sw_3()
     * @summary The event handler for switch #3
     */
    howells_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_howells().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_howells:
ctc.get_howells().get_interlocking_status()});
    }
    // ---- END howells_throw_sw_3() ----
    /* END Functions for CP Howells */


    /* Functions for CP Hall */
    /**
     * hall_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    hall_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hall().click_sig_2w(
            this.state.status_tier.block_howells_hall_1
        );
        // Set the state of the Interlocking
        this.setState({status_hall:
ctc.get_hall().get_interlocking_status()});
    }
    // ---- END hall_click_sig_2w() ----

    /**
     * hall_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    hall_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hall().click_sig_4w(
            this.state.status_tier.block_howells_hall_1,
            this.state.status_tier.block_hall_yard
        );
        // Set the state of the Interlocking
```

```
        this.setState({status_hall:
ctc.get_hall().get_interlocking_status()});
    }
    // ---- END hall_click_sig_4w() ----

    /**
     * hall_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    hall_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hall().click_sig_2e(
            this.state.status_tier.block_hall_hudson_1,
            this.state.status_tier.block_hall_hudson_2
        );
        // Set the state of the Interlocking
        this.setState({status_hall:
ctc.get_hall().get_interlocking_status()});
    }
    // ---- END hall_click_sig_2e() ----

    /**
     * hall_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    hall_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hall().click_sig_4e(
            this.state.status_tier.block_hall_hudson_2
        );
        // Set the state of the Interlocking
        this.setState({status_hall:
ctc.get_hall().get_interlocking_status()});
    }
    // ---- END hall_click_sig_4e() ----

    /**
     * hall_throw_sw_1()
     * @summary The event handler for switch #1
     */
    hall_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hall().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_hall:
ctc.get_hall().get_interlocking_status()});
    }
    // ---- END hall_throw_sw_1() ----
```

```
    /* END Functions for CP Hall */


    /* Functions for CP Hudson Junction */
    /**
     * hudson_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    hudson_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hudson().click_sig_2w(
            this.state.status_tier.block_hall_hudson_1,
            this.state.status_tier.block_hall_hudson_2
        );
        // Set the state of the Interlocking
        this.setState({status_hudson:
ctc.get_hudson().get_interlocking_status()});
    }
    // ---- END hudson_click_sig_2w() ----

    /**
     * hudson_click_sig_2ws()
     * @summary The event handler for Signal #2ws
     */
    hudson_click_sig_2ws = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hudson().click_sig_2ws(
            this.state.status_tier.block_hall_hudson_1,
            this.state.status_tier.block_hall_hudson_2
        );
        // Set the state of the Interlocking
        this.setState({status_hudson:
ctc.get_hudson().get_interlocking_status()});
    }
    // ---- END hudson_click_sig_2ws() ----

    /**
     * hudson_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    hudson_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hudson().click_sig_2e(
            this.state.status_tier.block_hudson_valley_1,
            this.state.status_tier.block_hudson_nysw
        );
        // Set the state of the Interlocking
```

```
        this.setState({status_hudson:
ctc.get_hudson().get_interlocking_status()});
    }
    // ---- END hudson_click_sig_2e() ----

    /**
     * hudson_click_sig_2es()
     * @summary The event handler for Signal #2es
     */
    hudson_click_sig_2es = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hudson().click_sig_2es(
            this.state.status_tier.block_hudson_valley_1,
            this.state.status_tier.block_hudson_nysw
        );
        // Set the state of the Interlocking
        this.setState({status_hudson:
ctc.get_hudson().get_interlocking_status()});
    }
    // ---- END hudson_click_sig_2es() ----

    /**
     * hudson_throw_sw_1()
     * @summary The event handler for switch #1
     */
    hudson_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hudson().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_hudson:
ctc.get_hudson().get_interlocking_status()});
    }
    // ---- END hudson_throw_sw_1() ----

    /**
     * hudson_throw_sw_3()
     * @summary The event handler for switch #3
     */
    hudson_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hudson().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_hudson:
ctc.get_hudson().get_interlocking_status()});
    }
    // ---- END hudson_throw_sw_3() ----
    /* END Functions for CP Hudson Junction */
```

```
    /* Functions for CP Central Valley */
    /**
     * valley_click_sig_1w()
     * @summary The event handler for Signal #1w
     */
    valley_click_sig_1w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_valley().click_sig_1w(
            this.state.status_tier.block_hudson_valley_1
        );
        // Set the state of the Interlocking
        this.setState({status_valley:
ctc.get_valley().get_interlocking_status()});
    }
    // ---- END valley_click_sig_1w() ----

    /**
     * valley_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    valley_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_valley().click_sig_2w(
            this.state.status_tier.block_hudson_valley_1
        );
        // Set the state of the Interlocking
        this.setState({status_valley:
ctc.get_valley().get_interlocking_status()});
    }
    // ---- END valley_click_sig_2w() ----

    /**
     * valley_click_sig_1e()
     * @summary The event handler for Signal #1e
     */
    valley_click_sig_1e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_valley().click_sig_1e(
            this.state.status_tier.block_valley_harriman_1,
            this.state.status_tier.block_valley_harriman_2
        );
        // Set the state of the Interlocking
        this.setState({status_valley:
ctc.get_valley().get_interlocking_status()});
    }
    // ---- END valley_click_sig_1e() ----
```

```
    /**
     * valley_throw_sw_21()
     * @summary The event handler for switch #21
     */
    valley_throw_sw_21 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_valley().throw_sw_21();
        // Set the state of the Interlocking
        this.setState({status_valley:
ctc.get_valley().get_interlocking_status()});
    }
    // ---- END valley_throw_sw_21() ----
    /* END Functions for CP Central Valley */


    /* Functions for CP Harriman */
    /**
     * harriman_click_sig_1w()
     * @summary The event handler for Signal #1w
     */
    harriman_click_sig_1w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_harriman().click_sig_1w(
            this.state.status_tier.block_valley_harriman_1,
            this.state.status_tier.block_valley_harriman_2,
            this.state.status_tier.block_harriman_industrial
        );
        // Set the state of the Interlocking
        this.setState({status_harriman:
ctc.get_harriman().get_interlocking_status()});
    }
    // ---- END harriman_click_sig_1w() ----

    /**
     * harriman_click_sig_1e()
     * @summary The event handler for Signal #1e
     */
    harriman_click_sig_1e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_harriman().click_sig_1e(
            this.state.status_tier.block_harriman_sterling_1
        );
        // Set the state of the Interlocking
        this.setState({status_harriman:
ctc.get_harriman().get_interlocking_status()});
    }
    // ---- END harriman_click_sig_1e() ----
```

```
    /**
     * harriman_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    harriman_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_harriman().click_sig_2e(
            this.state.status_tier.block_harriman_sterling_1
        );
        // Set the state of the Interlocking
        this.setState({status_harriman:
ctc.get_harriman().get_interlocking_status()});
    }
    // ---- END harriman_click_sig_2e() ----

    /**
     * harriman_click_sig_3e()
     * @summary The event handler for Signal #3e
     */
    harriman_click_sig_3e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_harriman().click_sig_3e(
            this.state.status_tier.block_harriman_sterling_1
        );
        // Set the state of the Interlocking
        this.setState({status_harriman:
ctc.get_harriman().get_interlocking_status()});
    }
    // ---- END harriman_click_sig_3e() ----

    /**
     * harriman_throw_sw_21()
     * @summary The event handler for switch #21
     */
    harriman_throw_sw_21 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_harriman().throw_sw_21();
        // Set the state of the Interlocking
        this.setState({status_harriman:
ctc.get_harriman().get_interlocking_status()});
    }
    // ---- END harriman_throw_sw_21() ----

    /**
     * harriman_throw_sw_32()
     * @summary The event handler for switch #32
     */
    harriman_throw_sw_32 = () => {
```

```
        // Get the backend function for the corresponding switch
        ctc.get_harriman().throw_sw_32();
        // Set the state of the Interlocking
        this.setState({status_harriman:
ctc.get_harriman().get_interlocking_status()});
    }
    // ---- END harriman_throw_sw_32() ----
    /* END Functions for CP Harriman */


    /* Functions for CP Sterling */
    /**
     * sterling_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    sterling_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sterling().click_sig_2w(
            this.state.status_tier.block_harriman_sterling_1
        );
        // Set the state of the Interlocking
        this.setState({status_sterling:
ctc.get_sterling().get_interlocking_status()});
    }
    // ---- END sterling_click_sig_2w() ----

    /**
     * sterling_click_sig_2ws()
     * @summary The event handler for Signal #2ws
     */
    sterling_click_sig_2ws = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sterling().click_sig_2ws(
            this.state.status_tier.block_harriman_sterling_1
        );
        // Set the state of the Interlocking
        this.setState({status_sterling:
ctc.get_sterling().get_interlocking_status()});
    }
    // ---- END sterling_click_sig_2ws() ----

    /**
     * sterling_click_sig_1e()
     * @summary The event handler for Signal #1e
     */
    sterling_click_sig_1e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
```

```
        ctc.get_sterling().click_sig_1e(
            this.state.status_tier.block_sterling_sf,
            this.state.status_tier.block_sterling_hilburn
        );
        // Set the state of the Interlocking
        this.setState({status_sterling:
ctc.get_sterling().get_interlocking_status()});
    }
    // ---- END sterling_click_sig_1e() ----

    /**
     * sterling_throw_sw_21()
     * @summary The event handler for switch #21
     */
    sterling_throw_sw_21 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_sterling().throw_sw_21();
        // Set the state of the Interlocking
        this.setState({status_sterling:
ctc.get_sterling().get_interlocking_status()});
    }
    // ---- END sterling_throw_sw_21() ----
    /* END Functions for CP Sterling */
    /* END Southern Tier Event Handlers */




    /* Main Line Event Handlers */
    /* Functions for Hilburn Interlocking */
    /**
     * hilburn_click_sig_2w_1()
     * @summary The event handler for Signal #2w_1
     */
    hilburn_click_sig_2w_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hilburn().click_sig_2w_1(
            this.state.status_mainLine.block_sterling_hilburn
        );
        // Set the state of the Interlocking
        this.setState({status_hilburn:
ctc.get_hilburn().get_interlocking_status()});
    }
    // ---- END hilburn_click_sig_2w_1() ----

    /**
     * hilburn_click_sig_2w_2()
     * @summary The event handler for Signal #2w_2
     */
```

```
    hilburn_click_sig_2w_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hilburn().click_sig_2w_2(
            this.state.status_mainLine.block_sterling_hilburn
        );
        // Set the state of the Interlocking
        this.setState({status_hilburn:
ctc.get_hilburn().get_interlocking_status()});
    }
    // ---- END hilburn_click_sig_2w_2() ----

    /**
     * hilburn_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    hilburn_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_hilburn().click_sig_2e(
            this.state.status_mainLine.block_hilburn_sf,
            this.state.status_mainLine.block_hilburn_yard_west
        );
        // Set the state of the Interlocking
        this.setState({status_hilburn:
ctc.get_hilburn().get_interlocking_status()});
    }
    // ---- END hilburn_click_sig_2e() ----

    /**
     * hilburn_throw_sw_1()
     * @summary The event handler for switch #1
     */
    hilburn_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_hilburn().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_hilburn:
ctc.get_hilburn().get_interlocking_status()});
    }
    // ---- END hilburn_throw_sw_1() ----
    /* END Functions for Hilburn Interlocking */


    /* Functions for SF Interlocking */
    /**
     * sf_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    sf_click_sig_2w = () => {
```

```
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sf().click_sig_2w(
            this.state.status_mainLine.block_sterling_sf,
            this.state.status_mainLine.block_hilburn_sf,
            this.state.status_mainLine.block_hilburn_yard_east
        );
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_click_sig_2w() ----

    /**
     * sf_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    sf_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sf().click_sig_4w(
            this.state.status_mainLine.block_hilburn_sf,
            this.state.status_mainLine.block_hilburn_yard_east
        );
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_click_sig_4w() ----

    /**
     * sf_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    sf_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sf().click_sig_2e(
            this.state.status_mainLine.block_sf_wc_1
        );
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_click_sig_2e() ----

    /**
     * sf_click_sig_4e_1()
     * @summary The event handler for Signal #4e_1
     */
```

```
    sf_click_sig_4e_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sf().click_sig_4e_1(
            this.state.status_mainLine.block_sf_wc_1,
            this.state.status_mainLine.block_sf_wc_2
        );
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_click_sig_4e_1() ----

    /**
     * sf_click_sig_4e_2()
     * @summary The event handler for Signal #4e_2
     */
    sf_click_sig_4e_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_sf().click_sig_4e_2(
            this.state.status_mainLine.block_sf_wc_1,
            this.state.status_mainLine.block_sf_wc_2
        );
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_click_sig_4e_2() ----

    /**
     * sf_throw_sw_1()
     * @summary The event handler for switch #1
     */
    sf_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_sf().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_throw_sw_1() ----

    /**
     * sf_throw_sw_3()
     * @summary The event handler for switch #3
     */
    sf_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_sf().throw_sw_3();
```

```
        // Set the state of the Interlocking
        this.setState({status_sf:
ctc.get_sf().get_interlocking_status()});
    }
    // ---- END sf_throw_sw_3() ----
    /* END Functions for SF Interlocking */


    /* Functions for WC Interlocking */
    /**
     * wc_click_sig_2w_1()
     * @summary The event handler for Signal #2w_1
     */
    wc_click_sig_2w_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_wc().click_sig_2w_1(
            this.state.status_mainLine.block_sf_wc_1,
            this.state.status_mainLine.block_sf_wc_2,
            this.state.status_mainLine.block_wc_yard
        );
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_click_sig_2w_1() ----

    /**
     * wc_click_sig_2w_2()
     * @summary The event handler for Signal #2w_2
     */
    wc_click_sig_2w_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_wc().click_sig_2w_2(
            this.state.status_mainLine.block_sf_wc_1,
            this.state.status_mainLine.block_sf_wc_2,
            this.state.status_mainLine.block_wc_yard
        );
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_click_sig_2w_2() ----

    /**
     * wc_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    wc_click_sig_4w = () => {
```

```
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_wc().click_sig_4w(
            this.state.status_mainLine.block_sf_wc_1,
            this.state.status_mainLine.block_sf_wc_2,
            this.state.status_mainLine.block_wc_yard
        )
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_click_sig_4w() ----

    /**
     * wc_click_sig_2e_1()
     * @summary The event handler for Signal #2e_1
     */
    wc_click_sig_2e_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_wc().click_sig_2e_1(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
            this.state.status_mainLine.block_wc_ridgewood_3
        );
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_click_sig_2e_1() ----

    /**
     * wc_click_sig_2e_2()
     * @summary The event handler for Signal #2e_2
     */
    wc_click_sig_2e_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_wc().click_sig_2e_2(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
            this.state.status_mainLine.block_wc_ridgewood_3
        );
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_click_sig_2e_2() ----

    /**
```

```
     * wc_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    wc_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_wc().click_sig_4e(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
            this.state.status_mainLine.block_wc_ridgewood_3
        );
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_click_sig_4e() ----

    /**
     * wc_throw_sw_1()
     * @summary The event handler for switch #1
     */
    wc_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_wc().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_throw_sw_1() ----

    /**
     * wc_throw_sw_3()
     * @summary The event handler for switch #3
     */
    wc_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_wc().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_throw_sw_3() ----

    /**
     * wc_throw_sw_5()
     * @summary The event handler for switch #5
     */
    wc_throw_sw_5 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_wc().throw_sw_5();
```

```
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_throw_sw_5() ----

    /**
     * wc_throw_sw_7()
     * @summary The event handler for switch #7
     */
    wc_throw_sw_7 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_wc().throw_sw_7();
        // Set the state of the Interlocking
        this.setState({status_wc:
ctc.get_wc().get_interlocking_status()});
    }
    // ---- END wc_throw_sw_7() ----
    /* END Functions for WC Interlocking */


    /* Functions for Ridgewood Junction Interlocking */
    /**
     * ridgewood_click_sig_2w_1()
     * @summary The event handler for Signal #2w_1
     */
    ridgewood_click_sig_2w_1 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_2w1(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
            this.state.status_mainLine.block_wc_ridgewood_3,
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_2w_1() ----

    /**
     * ridgewood_click_sig_2w_2()
     * @summary The event handler for Signal #2w_2
     */
    ridgewood_click_sig_2w_2 = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_2w2(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
```

```
            this.state.status_mainLine.block_wc_ridgewood_3,
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_2w_2() ----

    /**
     * ridgewood_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    ridgewood_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_4w(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
            this.state.status_mainLine.block_wc_ridgewood_3,
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_4w() ----

    /**
     * ridgewood_click_sig_6w()
     * @summary The event handler for Signal #6w
     */
    ridgewood_click_sig_6w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_6w(
            this.state.status_mainLine.block_wc_ridgewood_1,
            this.state.status_mainLine.block_wc_ridgewood_2,
            this.state.status_mainLine.block_wc_ridgewood_3,
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_6w() ----

    /**
     * ridgewood_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    ridgewood_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
```

```
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_2e(
            this.state.status_mainLine.block_ridgewood_suscon_1,
            this.state.status_mainLine.block_ridgewood_suscon_2,
            this.state.status_mainLine.block_ridgewood_suscon_3,
            this.state.status_mainLine.block_ridgewood_suscon_4
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_2e() ----

    /**
     * ridgewood_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    ridgewood_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_4e(
            this.state.status_mainLine.block_ridgewood_suscon_1,
            this.state.status_mainLine.block_ridgewood_suscon_2,
            this.state.status_mainLine.block_ridgewood_suscon_3,
            this.state.status_mainLine.block_ridgewood_suscon_4
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_4e() ----

    /**
     * ridgewood_click_sig_6e()
     * @summary The event handler for Signal #6e
     */
    ridgewood_click_sig_6e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_ridgewood().click_sig_6e(
            this.state.status_mainLine.block_ridgewood_suscon_1,
            this.state.status_mainLine.block_ridgewood_suscon_2,
            this.state.status_mainLine.block_ridgewood_suscon_3,
            this.state.status_mainLine.block_ridgewood_suscon_4
        );
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_click_sig_6e() ----
```

```
    /**
     * ridgewood_throw_sw_1()
     * @summary The event handler for switch #1
     */
    ridgewood_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_ridgewood().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_throw_sw_1() ----

    /**
     * ridgewood_throw_sw_3()
     * @summary The event handler for switch #3
     */
    ridgewood_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_ridgewood().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_throw_sw_3() ----

    /**
     * ridgewood_throw_sw_5()
     * @summary The event handler for switch #5
     */
    ridgewood_throw_sw_5 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_ridgewood().throw_sw_5();
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_throw_sw_5() ----

    /**
     * ridgewood_throw_sw_7()
     * @summary The event handler for switch #7
     */
    ridgewood_throw_sw_7 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_ridgewood().throw_sw_7();
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
```

```
    }
    // ---- END ridgewood_throw_sw_7() ----

    /**
     * ridgewood_throw_sw_9()
     * @summary The event handler for switch #9
     */
    ridgewood_throw_sw_9 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_ridgewood().throw_sw_9();
        // Set the state of the Interlocking
        this.setState({status_ridgewood:
ctc.get_ridgewood().get_interlocking_status()});
    }
    // ---- END ridgewood_throw_sw_9() ----
    /* END Functions for Ridgewood Junction Interlocking */


    /* Functions for Suscon Interlocking */
    /**
     * suscon_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    suscon_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_suscon().click_sig(
            "2W",
            this.state.status_mainLine.block_ridgewood_suscon_1,
            this.state.status_mainLine.block_ridgewood_suscon_2
        );
        // Set the state of the Interlocking
        this.setState({status_suscon:
ctc.get_suscon().get_interlocking_status()});
    }
    // ---- END suscon_click_sig_2w() ----

    /**
     * suscon_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    suscon_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_suscon().click_sig(
            "2E",
            this.state.status_mainLine.block_suscon_mill_1,
            this.state.status_mainLine.block_suscon_mill_2
        );
        // Set the state of the Interlocking
```

```
        this.setState({status_suscon:
ctc.get_suscon().get_interlocking_status()});
    }
    // ---- END suscon_click_sig_2e() ----

    /**
     * suscon_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    suscon_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_suscon().click_sig(
            "4W",
            this.state.status_mainLine.block_ridgewood_suscon_1,
            this.state.status_mainLine.block_ridgewood_suscon_2
        );
        // Set the state of the Interlocking
        this.setState({status_suscon:
ctc.get_suscon().get_interlocking_status()});
    }
    // ---- END suscon_click_sig_4w() ----

    /**
     * suscon_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    suscon_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_suscon().click_sig(
            "4E",
            this.state.status_mainLine.block_suscon_mill_1,
            this.state.status_mainLine.block_suscon_mill_2
        );
        // Set the state of the Interlocking
        this.setState({status_suscon:
ctc.get_suscon().get_interlocking_status()});
    }
    // ---- END suscon_click_sig_4e() ----

    /**
     * suscon_throw_sw_1()
     * @summary The event handler for switch #1
     */
    suscon_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_suscon().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_suscon:
```

```
ctc.get_suscon().get_interlocking_status()});
    }
    // ---- END suscon_throw_sw_1() ----

    /**
     * suscon_throw_sw_3()
     * @summary The event handler for switch #3
     */
    suscon_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_suscon().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_suscon:
ctc.get_suscon().get_interlocking_status()});
    }
    // ---- END suscon_throw_sw_3() ----
    /* END Functions for Suscon Interlocking */


    /* Functions for Mill Interlocking */
    /**
     * mill_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    mill_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_mill().click_sig(
            "2W",
            this.state.status_mainLine.block_suscon_mill_1,
            this.state.status_mainLine.block_suscon_mill_2
        );
        // Set the state of the Interlocking
        this.setState({status_mill:
ctc.get_mill().get_interlocking_status()});
    }
    // ---- END mill_click_sig_2w() ----

    /**
     * mill_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    mill_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_mill().click_sig(
            "2E",
            this.state.status_mainLine.block_mill_westSecaucus_1,
            this.state.status_mainLine.block_mill_westSecaucus_2
        );
```

```
        // Set the state of the Interlocking
        this.setState({status_mill:
ctc.get_mill().get_interlocking_status()});
    }
    // ---- END mill_click_sig_2e() ----

    /**
     * mill_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    mill_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_mill().click_sig(
            "4W",
            this.state.status_mainLine.block_suscon_mill_1,
            this.state.status_mainLine.block_suscon_mill_2
        );
        // Set the state of the Interlocking
        this.setState({status_mill:
ctc.get_mill().get_interlocking_status()});
    }
    // ---- END mill_click_sig_4w() ----

    /**
     * mill_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    mill_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_mill().click_sig(
            "4E",
            this.state.status_mainLine.block_mill_westSecaucus_1,
            this.state.status_mainLine.block_mill_westSecaucus_2
        );
        // Set the state of the Interlocking
        this.setState({status_mill:
ctc.get_mill().get_interlocking_status()});
    }
    // ---- END mill_click_sig_4e() ----

    /**
     * mill_throw_sw_1()
     * @summary The event handler for switch #1
     */
    mill_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_mill().throw_sw_1();
        // Set the state of the Interlocking
```

```
        this.setState({status_mill:
ctc.get_mill().get_interlocking_status()});
    }
    // ---- END mill_throw_sw_1() ----

    /**
     * mill_throw_sw_3()
     * @summary The event handler for switch #3
     */
    mill_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_mill().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_mill:
ctc.get_mill().get_interlocking_status()});
    }
    // ---- END mill_throw_sw_3() ----
    /* END Functions for Mill Interlocking */


    /* Functions for West Secaucus Interlocking */
    /**
     * westSecaucus_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    westSecaucus_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_westSecaucus().click_sig(
            "2W",
            this.state.status_mainLine.block_mill_westSecaucus_1,
            this.state.status_mainLine.block_mill_westSecaucus_2
        );
        // Set the state of the Interlocking
        this.setState({status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status()});
    }
    // ---- END westSecaucus_click_sig_2w() ----

    /**
     * westSecaucus_click_sig_2e()
     * @summary The event handler for Signal #2e
     */
    westSecaucus_click_sig_2e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_westSecaucus().click_sig(
            "2E",
            this.state.status_mainLine.block_westSecaucus_laurel_1,
            this.state.status_mainLine.block_westSecaucus_laurel_2
```

```
        );
        // Set the state of the Interlocking
        this.setState({status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status()});
    }
    // ---- END westSecaucus_click_sig_2e() ----

    /**
     * westSecaucus_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    westSecaucus_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_westSecaucus().click_sig(
            "4W",
            this.state.status_mainLine.block_mill_westSecaucus_1,
            this.state.status_mainLine.block_mill_westSecaucus_2
        );
        // Set the state of the Interlocking
        this.setState({status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status()});
    }
    // ---- END westSecaucus_click_sig_4w() ----

    /**
     * westSecaucus_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    westSecaucus_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_westSecaucus().click_sig(
            "4E",
            this.state.status_mainLine.block_westSecaucus_laurel_1,
            this.state.status_mainLine.block_westSecaucus_laurel_2
        );
        // Set the state of the Interlocking
        this.setState({status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status()});
    }
    // ---- END westSecaucus_click_sig_4e() ----

    /**
     * westSecaucus_throw_sw_1()
     * @summary The event handler for switch #1
     */
    westSecaucus_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_westSecaucus().throw_sw_1();
```

```
        // Set the state of the Interlocking
        this.setState({status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status()});
    }
    // ---- END westSecaucus_throw_sw_1() ----

    /**
     * westSecaucus_throw_sw_3()
     * @summary The event handler for switch #3
     */
    westSecaucus_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_westSecaucus().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_westSecaucus:
ctc.get_westSecaucus().get_interlocking_status()});
    }
    // ---- END westSecaucus_throw_sw_3() ----
    /* END Functions for West Secaucus Interlocking */


    /* Functions for Laurel Interlocking */
    /**
     * laurel_click_sig_2w()
     * @summary The event handler for Signal #2w
     */
    laurel_click_sig_2w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_2w(
            this.state.status_mainLine.block_hx_laurel_2,
            this.state.status_mainLine.block_westSecaucus_laurel_1,
            this.state.status_mainLine.block_hx_laurel_1
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_2w() ----

    /**
     * laurel_click_sig_4w()
     * @summary The event handler for Signal #4w
     */
    laurel_click_sig_4w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_4w(
            this.state.status_mainLine.block_hx_laurel_2,
            this.state.status_mainLine.block_westSecaucus_laurel_1,
```

```
            this.state.status_mainLine.block_hx_laurel_1
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_4w() ----

    /**
     * laurel_click_sig_8w()
     * @summary The event handler for Signal #8w
     */
    laurel_click_sig_8w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_8w(
            this.state.status_mainLine.block_hx_laurel_2,
            this.state.status_mainLine.block_westSecaucus_laurel_1,
            this.state.status_mainLine.block_hx_laurel_1,
            this.state.status_mainLine.block_westSecaucus_laurel_2
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_8w() ----

    /**
     * laurel_click_sig_10w()
     * @summary The event handler for Signal #10w
     */
    laurel_click_sig_10w = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_10w(
            this.state.status_mainLine.block_hx_laurel_2,
            this.state.status_mainLine.block_westSecaucus_laurel_1,
            this.state.status_mainLine.block_hx_laurel_1,
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_10w() ----

    /**
     * laurel_click_sig_6e()
     * @summary The event handler for Signal #6e
     */
    laurel_click_sig_6e = () => {
```

```
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_6e(
            this.state.status_mainLine.block_westEnd_laurel_1,
            this.state.status_mainLine.block_westEnd_laurel_2,
            this.state.status_mainLine.block_westEnd_laurel_3,
            this.state.status_mainLine.block_westEnd_laurel_4
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_6e() ----

    /**
     * laurel_click_sig_12e()
     * @summary The event handler for Signal #12e
     */
    laurel_click_sig_12e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_12e(
            this.state.status_mainLine.block_westEnd_laurel_1,
            this.state.status_mainLine.block_westEnd_laurel_2,
            this.state.status_mainLine.block_westEnd_laurel_3,
            this.state.status_mainLine.block_westEnd_laurel_4
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_12e() ----

    /**
     * laurel_click_sig_4e()
     * @summary The event handler for Signal #4e
     */
    laurel_click_sig_4e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_4e(
            this.state.status_mainLine.block_westEnd_laurel_1,
            this.state.status_mainLine.block_westEnd_laurel_2,
            this.state.status_mainLine.block_westEnd_laurel_3,
            this.state.status_mainLine.block_westEnd_laurel_4
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
```

```
    // ---- END laurel_click_sig_4e() ----

    /**
     * laurel_click_sig_8e()
     * @summary The event handler for Signal #8e
     */
    laurel_click_sig_8e = () => {
        // Get the backend function for the corresponding signal
        // Passing reference the next blocks
        ctc.get_laurel().click_sig_8e(
            this.state.status_mainLine.block_westEnd_laurel_4
        );
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_click_sig_8e() ----

    /**
     * laurel_throw_sw_1()
     * @summary The event handler for switch #1
     */
    laurel_throw_sw_1 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_laurel().throw_sw_1();
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_throw_sw_1() ----

    /**
     * laurel_throw_sw_3()
     * @summary The event handler for switch #3
     */
    laurel_throw_sw_3 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_laurel().throw_sw_3();
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_throw_sw_3() ----

    /**
     * laurel_throw_sw_7()
     * @summary The event handler for switch #7
     */
    laurel_throw_sw_7 = () => {
        // Get the backend function for the corresponding switch
```

```
        ctc.get_laurel().throw_sw_7();
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_throw_sw_7() ----

    /**
     * laurel_throw_sw_11()
     * @summary The event handler for switch #11
     */
    laurel_throw_sw_11 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_laurel().throw_sw_11();
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_throw_sw_11() ----

    /**
     * laurel_throw_sw_13()
     * @summary The event handler for switch #13
     */
    laurel_throw_sw_13 = () => {
        // Get the backend function for the corresponding switch
        ctc.get_laurel().throw_sw_13();
        // Set the state of the Interlocking
        this.setState({status_laurel:
ctc.get_laurel().get_interlocking_status()});
    }
    // ---- END laurel_throw_sw_13() ----
    /* END Functions for Laurel Interlocking */
}

// Export the panel to be drawn on the screen
export default MainLine;
```

```
/**
 * @file BergenTracks.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for The Tracks of the
Bergen County Line
 * Extends the React Component Class and is the UI part of the Bergen
County Line Tracks,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Bergen_County_Line/bergenCounty.css';


/**
 * The React JSX Component Class for the Tracks in the Bergen County
Line portion
 * his class is a JSX React Component for the Bergen County Line
Tracks, this will control all the UI for the comonent,
 * showing what blocks are occupied by a train
 */
class BergenTracks extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     * This object holds all the information for the tracks that is
required to display the routes
     * correctly Anything that has "this.props." is passed down from
the CTC interlocking class
     */
    state = {
        // Symbols
        symbol_ridgewood_bt_1:
this.props.symbols.symbol_ridgewood_bt_1,
        symbol_ridgewood_bt_2:
this.props.symbols.symbol_ridgewood_bt_2,
        symbol_bt_pascack_1: this.props.symbols.symbol_bt_pascack_1,
        symbol_bt_pascack_2: this.props.symbols.symbol_bt_pascack_2,
        symbol_bt_nysw: this.props.symbols.symbol_bt_nysw,
        symbol_pascack_hx_1: this.props.symbols.symbol_pascack_hx_1,
        symbol_pascack_hx_2: this.props.symbols.symbol_pascack_hx_2,
        symbol_hx_laurel_1: this.props.symbols.symbol_hx_laurel_1,
        symbol_hx_laurel_2: this.props.symbols.symbol_hx_laurel_2,
        symbol_hx_croxton_1: this.props.symbols.symbol_hx_croxton_1,
```

```
        symbol_hx_croxton_2: this.props.symbols.symbol_hx_croxton_2,

        // Blocks
        block_hx_laurel_1: this.props.blocks.block_hx_laurel_1,
        block_hx_laurel_2: this.props.blocks.block_hx_laurel_2,

        block_pascack_hx_1: this.props.blocks.block_pascack_hx_1,
        block_pascack_hx_2: this.props.blocks.block_pascack_hx_2,

        block_bt_pascack_1: this.props.blocks.block_bt_pascack_1,
        block_bt_pascack_2: this.props.blocks.block_bt_pascack_2,

        block_ridgewood_bt_1: this.props.blocks.block_ridgewood_bt_1,
        block_ridgewood_bt_2: this.props.blocks.block_ridgewood_bt_2,

        block_bt_nysw: this.props.blocks.block_bt_nysw,
        block_hx_croxton_1: this.props.blocks.block_hx_croxton_1,
        block_hx_croxton_1: this.props.blocks.block_hx_croxton_2
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps) {
        this.setState({
            // Symbols
            symbol_ridgewood_bt_1:
nextProps.symbols.symbol_ridgewood_bt_1,
            symbol_ridgewood_bt_2:
nextProps.symbols.symbol_ridgewood_bt_2,
            symbol_bt_pascack_1:
nextProps.symbols.symbol_bt_pascack_1,
            symbol_bt_pascack_2:
nextProps.symbols.symbol_bt_pascack_2,
            symbol_bt_nysw: nextProps.symbols.symbol_bt_nysw,
            symbol_pascack_hx_1:
nextProps.symbols.symbol_pascack_hx_1,
            symbol_pascack_hx_2:
nextProps.symbols.symbol_pascack_hx_2,
            symbol_hx_laurel_1: nextProps.symbols.symbol_hx_laurel_1,
            symbol_hx_laurel_2: nextProps.symbols.symbol_hx_laurel_2,
            symbol_hx_croxton_1:
nextProps.symbols.symbol_hx_croxton_1,
            symbol_hx_croxton_2:
nextProps.symbols.symbol_hx_croxton_2,
```

```
            // Blocks
            block_hx_laurel_1: nextProps.blocks.block_hx_laurel_1,
            block_hx_laurel_2: nextProps.blocks.block_hx_laurel_2,

            block_pascack_hx_1: nextProps.blocks.block_pascack_hx_1,
            block_pascack_hx_2: nextProps.blocks.block_pascack_hx_2,

            block_bt_pascack_1: nextProps.blocks.block_bt_pascack_1,
            block_bt_pascack_2: nextProps.blocks.block_bt_pascack_2,

            block_ridgewood_bt_1:
nextProps.blocks.block_ridgewood_bt_1,
            block_ridgewood_bt_2:
nextProps.blocks.block_ridgewood_bt_2,

            block_bt_nysw: nextProps.blocks.block_bt_nysw,
            block_hx_croxton_1: nextProps.blocks.block_hx_croxton_1,
            block_hx_croxton_2: nextProps.blocks.block_hx_croxton_2
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        return (
            <div>
                {/* Tags */}
                <div className="bt_nysw_tag">NYS&W RR</div>
                <div className="hx_line_tag">Norfolk Southern Croxton
Yard</div>

                {/* Symbols */}
                <div
className="symbol_ridgewood_bt_1">{this.state.symbol_ridgewood_bt_1}</
div>
                <div
className="symbol_ridgewood_bt_2">{this.state.symbol_ridgewood_bt_2}</
div>
                <div
className="symbol_bt_pascack_1">{this.state.symbol_bt_pascack_1}</div>
                <div
className="symbol_bt_pascack_2">{this.state.symbol_bt_pascack_2}</div>
                <div
className="symbol_bt_nysw">{this.state.symbol_bt_nysw}</div>
                <div
```

```
className="symbol_pascack_hx_1">{this.state.symbol_pascack_hx_1}</div>
                <div
className="symbol_pascack_hx_2">{this.state.symbol_pascack_hx_2}</div>
                <div
className="symbol_hx_laurel_1">{this.state.symbol_hx_laurel_1}</div>
                <div
className="symbol_hx_laurel_2">{this.state.symbol_hx_laurel_2}</div>
                <div
className="symbol_hx_croxton_1">{this.state.symbol_hx_croxton_1}</div>
                <div
className="symbol_hx_croxton_2">{this.state.symbol_hx_croxton_2}</div>

                {/* Tracks */}
                {/* Yard Leads */}
                <div className="b_croxton_1" style={{background:
this.state.block_hx_croxton_1}}></div>
                <div className="b_croxton_2" style={{background:
this.state.block_hx_croxton_2}}></div>
                <div className="b_nysw" style={{background:
this.state.block_bt_nysw}}></div>

                {/* Laurel to HX */}
                <div className="b_laurel_hx_1_west"
style={{background: this.state.block_hx_laurel_1}}></div>
                <div className="b_laurel_hx_1_diag"
style={{background: this.state.block_hx_laurel_1}}></div>
                <div className="b_laurel_hx_1_east"
style={{background: this.state.block_hx_laurel_1}}></div>
                <div className="b_laurel_hx_2_west"
style={{background: this.state.block_hx_laurel_2}}></div>
                <div className="b_laurel_hx_2_diag"
style={{background: this.state.block_hx_laurel_2}}></div>
                <div className="b_laurel_hx_2_east"
style={{background: this.state.block_hx_laurel_2}}></div>

                {/* HX to Pascack Junction */}
                <div className="b_hx_pascack_1" style={{background:
this.state.block_pascack_hx_1}}></div>
                <div className="b_hx_pascack_2" style={{background:
this.state.block_pascack_hx_2}}></div>

                {/* Pascack Junction to BT */}
                <div className="b_pascack_bt_1" style={{background:
this.state.block_bt_pascack_1}}></div>
                <div className="b_pascack_bt_2" style={{background:
this.state.block_bt_pascack_2}}></div>

                {/* BT to Ridgewood Junction */}
                <div className="b_bt_ridgewood_1_east"
style={{background: this.state.block_ridgewood_bt_1}}></div>
```

```
                <div className="b_bt_ridgewood_1_diag"
style={{background: this.state.block_ridgewood_bt_1}}></div>
                <div className="b_bt_ridgewood_1_west"
style={{background: this.state.block_ridgewood_bt_1}}></div>
                <div className="b_bt_ridgewood_2_east"
style={{background: this.state.block_ridgewood_bt_2}}></div>
                <div className="b_bt_ridgewood_2_diag"
style={{background: this.state.block_ridgewood_bt_2}}></div>
                <div className="b_bt_ridgewood_2_west"
style={{background: this.state.block_ridgewood_bt_2}}></div>
            </div>
        );
    }
    // ---- END render() ----
}

// Export the tracks to be drawn on the screen
export default BergenTracks;
```

```
/**
 * @file BT.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for BT Interlocking
 *
 * @description Extends the React Component Class and is the UI part
of the BT Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Bergen_County_Line/bt.css';

// Import Images
// Switch Images
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
```

```
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the BT Interlocking
 *
 * This class is a JSX React Component for the BT Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
```

```
class BT extends Component {
    /**
     * @summary Object that holds the state or status information for
the component
     *
     * @description This object holds all the information for the
interlocking that is required to display the routes
     * correctly Anything that has "this.props." is passed down from
the CTC interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        sw_5: this.props.status.sw_5,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_135,
        sw_3_src: CX_225,
        sw_5_src: SW_U_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        tail_3_e: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Information For Interlocking Routes
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_1,
        route_2: this.props.status.routed_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component The
data that is being changed is passed down from the CTC classes in the
simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
```

```
            sw_3: nextProps.status.sw_3,
            sw_5: nextProps.status.sw_5,

            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_1,
            route_2: nextProps.status.routed_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_images();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="bt_title">BT</div>
                <div className="bt_milepost">MP 14.2</div>
                {/* West Side Tail Tracks */}
                <div className="bt_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="bt_2_west" style={{background:
this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="bt_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="bt_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="bt_SW_5"
onClick={this.props.throw_sw_5}><img src={this.state.sw_5_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="bt_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="bt_2_east" style={{background:
this.state.tail_2_e}}></div>
```

```
                <div className="bt_3_east" style={{background:
this.state.tail_3_e}}></div>
                {/* Signals */}
                <div className="bt_sig_2w-2"
onClick={this.props.click_sig_2w2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="bt_sig_2w-1"
onClick={this.props.click_sig_2w1}><img src={this.state.sig_2w1_src}/
></div>
                <div className="bt_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="bt_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="bt_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * @description Function takes what routes are currently set in
the Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }
```

```
        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__3_ridgewood_bt" ||
this.state.routes[i] === "E_1_1__|__1_bt_pascack") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                // The route is occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Occupied;

                    // Crossovers that could change based off Track #2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                        this.state.sw_3_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The route is NOT occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Lined;

                    // Crossovers that could change based off of Track
#2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    // Track #2 Occupied
```

```
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
                        this.state.sw_3_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__3_ridgewood_bt") {
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_1__|
__3_ridgewood_bt" || this.state.routes[i] === "E_1_3__|__3_bt_nysw") {
                // Tail Tracks
                this.state.tail_3_e = color_1;
                this.state.tail_1_w = color_1;

                // If The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Occupied;

                    // Crossovers that can change depending on the
other track
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                        this.state.sw_3_src = CX_225_Occupied_Both;
```

```
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                        this.state.sw_3_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is Not Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Lined;

                    // Crossovers that can change depending on the
other track
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
                        this.state.sw_3_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_1__|
__3_ridgewood_bt") {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|
```

```
__4_ridgewood_bt" || this.state.routes[i] === "E_2_2__|
__2_bt_pascack") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                // If the Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    // Crossovers that can change depending on the
other track
                    if (this.state.route_1) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                        this.state.sw_3_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route In Not Occupied
                else {
                    // Switches
                    // Crossovers that can change depending on the
other track
                    if (this.state.route_1) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                        this.state.sw_3_src = CX_225_Lined_Bottom;
                    }
```

```
                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__4_ridgewood_bt") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__4_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Occupied;
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Lined;
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Clear;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|
```

```
__1_bt_pascack") {
                // Tail Tracks
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Occupied;
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Lined;
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_3_2__|
__4_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_3_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Occupied;
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
```

```
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e_src = SIG_E_Stop;
            this.state.sig_4e_src = SIG_E_Stop;
        }
        // The Route Is NOT Occupied
        else {
            // Switches
            this.state.sw_5_src = SW_U_E_R_Lined;
            this.state.sw_3_src = CX_225_R_Lined;
            this.state.sw_1_src = CX_135_Lined_Bottom;

            // Signals
            this.state.sig_2w1_src = SIG_W_Stop;
            this.state.sig_2w2_src = SIG_W_Clear;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e_src = SIG_E_Stop;
            this.state.sig_4e_src = SIG_E_Stop;
        }
    }
    else if (this.state.routes[i] === "E_2_3__|__3_bt_nysw") {
        // Tail Tracks
        this.state.tail_3_e = color_2;
        this.state.tail_2_w = color_2;

        // The Route Is Occupied
        if (this.state.occupied_2) {
            // Switches
            this.state.sw_5_src = SW_U_E_R_Occupied;
            this.state.sw_3_src = CX_225_R_Occupied;
            this.state.sw_1_src = CX_135_Occupied_Bottom;

            // Signals
            this.state.sig_2w1_src = SIG_W_Stop;
            this.state.sig_2w2_src = SIG_W_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e_src = SIG_E_Stop;
            this.state.sig_4e_src = SIG_E_Stop;
        }
        // The Route Is NOT Occupied
        else {
            // Switches
            this.state.sw_5_src = SW_U_E_R_Lined;
            this.state.sw_3_src = CX_225_R_Lined;
            this.state.sw_1_src = CX_135_Lined_Bottom;

            // Signals
            this.state.sig_2w1_src = SIG_W_Stop;
            this.state.sig_2w2_src = SIG_W_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e_src = SIG_E_Stop;
```

```
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__3_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_1_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_3_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_bt_pascack") {
                // Tail Tracks
                this.state.tail_2_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
```

```
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * @description This function uses the data passed in through
status from the CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_images() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_135_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_135;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_225_R;
        }
```

```
        // SW #3 Normal
        else {
            this.state.sw_3_src = CX_225;
        }

        // Set SW #5
        // SW #5 Reversed
        if (this.state.sw_5) {
            this.state.sw_5_src = SW_U_E_R;
        }
        // SW #5 Normal
        else {
            this.state.sw_5_src = SW_U_E;
        }
    }
    // ---- END set_switch_image() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * @description This function is need, because if the player was
to remove a route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_3_e = Empty;

        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default BT;
```

```
/**
 * @file HX.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for HX Interlocking
 *
 * @description Extends the React Component Class and is the UI part
of the HX Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Bergen_County_Line/hx.css';

// Import Images
// Switch Images
import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
```

```
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';

/**
 * The React JSX Component Class for the HX Interlocking
 * This class is a JSX React Component for the HX Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class HX extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * @description This object holds all the information for the
interlocking that is required to display the routes
     * correctly Anything that has "this.props." is passed down from
the CTC interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        sw_5: this.props.status.sw_5,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_225,
        sw_3_src: SW_U_E,
        sw_5_src: SW_U_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
```

```
        tail_3_e: Empty,
        tail_4_e: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_2w3_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Information For Interlocking Routes
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_1,
        route_2: this.props.status.routed_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * @description The data that is being changed is passed down from
the CTC classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            sw_5: nextProps.status.sw_5,

            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_1,
            route_2: nextProps.status.routed_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
```

```
        // Set the switch images based off the state of each crossover
        this.set_switch_images();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="hx_title">HX</div>
                <div className="hx_milepost">MP 5.4</div>
                {/* West Side Tail Tracks */}
                <div className="hx_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="hx_2_west" style={{background:
this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="hx_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="hx_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="hx_SW_5"
onClick={this.props.throw_sw_5}><img src={this.state.sw_5_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="hx_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="hx_2_east" style={{background:
this.state.tail_2_e}}></div>
                <div className="hx_croxton_1" style={{background:
this.state.tail_4_e}}></div>
                <div className="hx_croxton_2" style={{background:
this.state.tail_3_e}}></div>
                {/* Signals */}
                <div className="hx_sig_2w-3"
onClick={this.props.click_sig_2w3}><img src={this.state.sig_2w3_src}/
></div>
                <div className="hx_sig_2w-2"
onClick={this.props.click_sig_2w2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="hx_sig_2w-1"
onClick={this.props.click_sig_2w1}><img src={this.state.sig_2w1_src}/
></div>
                <div className="hx_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="hx_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="hx_sig_4e"
```

```
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * @description Function takes what routes are currently set in
the Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Set Track colors depending on if they are routed or
occupied
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }

        // Loop Through All The Routes
        for (let i = 0; i < this.state.routes.length; i++) {
            // West and East normal on Track 1
            if (this.state.routes[i] === "W_1_1__|__1_pascack_hx" ||
this.state.routes[i] === "E_1_1__|__3_hx_laurel") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = SW_U_E_Occupied;

                    // Crossovers that can change based on other
tracks status
```

```
                    // Trk2 Lined
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Trk2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                    }
                    // Nothing Trk2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = SW_U_E_Lined;

                    // Crossovers that can change based on other
tracks status
                    // Trk2 Lined
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                    }
                    // Trk2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Trk2
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_pascack_hx") {
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
```

```
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            // West and East normal on Track 2
            else if (this.state.routes[i] === "W_2_2__|__2_pascack_hx"
|| this.state.routes[i] === "E_2_2__|__1_hx_laurel") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    // Crossovers that can change base on track 1
                    // Trk1 Lined
                    if (this.state.route_1) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Trk1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                    }
                    // Nothing Trk1
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Crossovers that can change base on track 1
                    // Trk1 Lined
                    if (this.state.route_1) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                    }
                    // Trk1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
```

```
                    }
                    // Nothing Trk1
                    else {
                        this.state.sw_1_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_pascack_hx") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_pascack_hx") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route In Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = SW_U_E_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = SW_U_E_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Clear;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
```

```
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|__3_hx_laurel")
{
                // Tail Tracks
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route In Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_3_src = SW_U_E_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = SW_U_E_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_3_2__|
__2_pascack_hx") {
                // Tail Tracks
                this.state.tail_3_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Occupied;
```

```
                    this.state.sw_3_src = SW_U_E_R_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Lined;
                    this.state.sw_3_src = SW_U_E_R_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_3__|
__3_hx_croxton") {
                // Tail Tracks
                this.state.tail_3_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Occupied;
                    this.state.sw_3_src = SW_U_E_R_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
```

```
                    // Switches
                    this.state.sw_5_src = SW_U_E_Lined;
                    this.state.sw_3_src = SW_U_E_R_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_4_2__|
__2_pascack_hx") {
                // Tail Tracks
                this.state.tail_4_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Occupied;
                    this.state.sw_3_src = SW_U_E_R_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Lined;
                    this.state.sw_3_src = SW_U_E_R_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
```

```
            }
            else if (this.state.routes[i] === "E_2_4__|
__4_hx_croxton") {
                // Tail Tracks
                this.state.tail_4_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Occupied;
                    this.state.sw_3_src = SW_U_E_R_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Lined;
                    this.state.sw_3_src = SW_U_E_R_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_3_1__|
__1_pascack_hx"|| this.state.routes[i] === "E_1_3__|__3_hx_croxton") {
                // Tail Tracks
                this.state.tail_3_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Occupied;
                    this.state.sw_3_src = SW_U_E_R_Occupied;
```

```
                    // Crossovers that can change based on track 2
state
                    // Trk2 Lined
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Trk2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                    }
                    // Nothing Trk2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_Lined;
                    this.state.sw_3_src = SW_U_E_R_Lined;

                    // Crossovers that can change based on track 2
state
                    // Trk2 Lined
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                    }
                    // Trk2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Trk2
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_1__|
__1_pascack_hx") {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Clear;
```

```
                        this.state.sig_2w3_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_4_1__|
__1_pascack_hx"|| this.state.routes[i] === "E_1_4__|__4_hx_croxton") {
                // Tail Tracks
                this.state.tail_4_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Occupied;
                    this.state.sw_3_src = SW_U_E_R_Occupied;

                    // Crossovers that can change based on track 2
state
                    // Trk2 Lined
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Trk2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                    }
                    // Nothing Trk2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = SW_U_E_R_Lined;
```

```
                    this.state.sw_3_src = SW_U_E_R_Lined;

                    // Crossovers that can change based on track 2
state
                    // Trk2 Lined
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                    }
                    // Trk2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Trk2
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_4_1__|
__1_pascack_hx") {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_images()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * @description This function uses the data passed in through
status from the CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
```

```
     */
    set_switch_images() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_225_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_225;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = SW_U_E_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = SW_U_E;
        }

        // Set SW #5
        // SW #5 Reversed
        if (this.state.sw_5) {
            this.state.sw_5_src = SW_U_E_R;
        }
        // SW #5 Normal
        else {
            this.state.sw_5_src = SW_U_E;
        }
    }
    // ---- END set_switch_images() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * @description This function is need, because if the player was
to remove a route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_3_e = Empty;
```

```
        this.state.tail_4_e = Empty;

        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_2w3_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default HX;
```

```
/**
 * @file PascackJct.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Pascack Junction
Interlocking
 *
 * @description Extends the React Component Class and is the UI part
of the Pascack Junction Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Bergen_County_Line/pascack_jct.css';

// Import Images
// Switch Images
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
```

```
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Pascack Junction Interlocking
 * This class is a JSX React Component for the Pascack Junction
Interlocking, this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class PascackJct extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * @description This object holds all the information for the
interlocking that is required to display the routes
     * correctly Anything that has "this.props." is passed down from
the CTC interlocking class
```

```
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_225,
        sw_3_src: CX_135,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Information For Interlocking Routes
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_1,
        route_2: this.props.status.routed_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * @description The data that is being changed is passed down from
the CTC classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,

            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_1,
            route_2: nextProps.status.routed_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
```

```
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_images();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="pascack_title">PASCACK</div>
                <div className="pascack_milepost">MP 7.6</div>
                {/* West Side Tail Tracks */}
                <div className="pascack_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="pascack_2_west" style={{background:
this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="pascack_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="pascack_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="pascack_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="pascack_2_east" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="pascack_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="pascack_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="pascack_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="pascack_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
            </div>
        );
```

```
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * @description Function takes what routes are currently set in
the Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_bt_pascack" ||
this.state.routes[i] === "E_1_1__|__1_pascack_hx") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                // Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
```

```
                    }
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // Route Is Not Occupied
                else {
                    // Switches
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_bt_pascack") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|__2_bt_pascack"
|| this.state.routes[i] === "E_2_2__|__2_pascack_hx") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_2) {
                    // Switches
```

```
                    // Switches
                    if (this.state.route_1) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Bottom;
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    // Switches
                    if (this.state.route_1) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_225_Lined_Bottom;
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_bt_pascack") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
```

```
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_bt_pascack") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Top;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Top;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|
__1_pascack_hx") {
                // Tail Tracks
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Top;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
```

```
                    this.state.sw_1_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Top;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__1_bt_pascack") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_1_w = color_2;

                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_1_src = CX_225_Occupied_Top;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_1_src = CX_225_Lined_Top;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_pascack_hx") {
                // Tail Tracks
                this.state.tail_2_e = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_1_src = CX_225_Occupied_Top;
                    this.state.sw_3_src = CX_135_R_Occupied;
```

```
                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_1_src = CX_225_Lined_Top;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_images()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * @description This function uses the data passed in through
status from the CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_images() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_225_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_225;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_135_R;
        }
        // SW #3 Normal
```

```
        else {
            this.state.sw_3_src = CX_135;
        }
    }
    // ---- END set_switch_images() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * @description This function is need, because if the player was
to remove a route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;

        this.state.sig_2w_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default PascackJct;
```

```
/**
 * @file Hilburn.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Hilburn Interlocking
 *
 * @description Extends the React Component Class and is the UI part
of the Hilburn Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/hilburn.css';

// Import Images
// Switch Images
import SW_D_E from '../../../../public/images/SW_D_E.png';
import SW_D_E_Lined from '../../../../public/images/SW_D_E_Lined.png';
import SW_D_E_Occupied from '../../../../public/images/
SW_D_E_Occupied.png'
import SW_D_E_R from '../../../../public/images/SW_D_E_R.png';
import SW_D_E_R_Lined from '../../../../public/images/
SW_D_E_R_Lined.png';
import SW_D_E_R_Occupied from '../../../../public/images/
SW_D_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Hilburn Interlocking
 * This class is a JSX React Component for the Hilburn Interlocking,
this will control all the UI for the component,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
```

```
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Hilburn extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * @description This object holds all the information for the
interlocking that is required to display the routes
     * correctly Anything that has "this.props." is passed down from
the CTC interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_D_E,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_2e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_w: Empty,
        tail_e: Empty,
        tail_yard: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * @description The data that is being changed is passed down from
the CTC classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----
```

```
    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="hilburn_title">HILBURN</div>
                <div className="hilburn_milepost">MP 32.3</div>
                {/* West Side Tail Tracks */}
                <div className="hilburn_west" style={{background:
this.state.tail_w}}></div>
                {/* Switches */}
                <div className="hilburn_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="hilburn_east" style={{background:
this.state.tail_e}}></div>
                <div className="hilburn_yard" style={{background:
this.state.tail_yard}}></div>
                {/* Signals */}
                <div className="hilburn_sig_2w-1"
onClick={this.props.click_sig_2w_1}><img src={this.state.sig_2w1_src}/
></div>
                <div className="hilburn_sig_2w-2"
onClick={this.props.click_sig_2w_2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="hilburn_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
```

```
     *
     * @description Function takes what routes are currently set in
the Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            // Routes with Track 1 on both the West and East sides
            if (this.state.routes[i] === "W_1_1__|
__2_sterling_hilburn" || this.state.routes[i] === "E_1_1__|
__2_hilburn_sf") {
                // Tail Tracks
                this.state.tail_e = color;
                this.state.tail_w = color;

                // Drawing if the interlocking is occupied
                if (this.state.occupied) {
                    // Switch Image
                    this.state.sw_1_src = SW_D_E_Occupied;

                    // Signal Images
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // Routing is not occupied
                else {
                    // Switch Image
                    this.state.sw_1_src = SW_D_E_Lined;

                    // Signal Images
                    // West Bound
                    if (this.state.routes[i] === "W_1_1__|
__2_sterling_hilburn") {
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound
```

```
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            // Routes With Track 2 on West Side and Track 1 on East
Side
            else if (this.state.routes[i] === "W_2_1__|
__2_sterling_hilburn" || this.state.routes[i] === "E_1_2__|
__0_hilburn_yardWest") {
                // Tail Tracks
                this.state.tail_yard = color;
                this.state.tail_w = color;

                // Drawing if the interlocking is occupied
                if (this.state.occupied) {
                    // Switch Image
                    this.state.sw_1_src = SW_D_E_R_Occupied;

                    // Signal Images
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // Routing that is not occupied
                else {
                    // Switch Image
                    this.state.sw_1_src = SW_D_E_R_Lined;

                    // Signal Images
                    // West Bound Route
                    if (this.state.routes[i] === "W_2_1__|
__2_sterling_hilburn") {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Route
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----
```

```
    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_D_E_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_D_E;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * @description This function is need, because if the player was
to remove a route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2e_src = SIG_E;

        this.state.tail_w = Empty;
        this.state.tail_e = Empty;
        this.state.tail_yard = Empty;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Hilburn;
```

```
/**
 * @file Laurel.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Laurel Interlocking
 *
 * Extends the React Component Class and is the UI part of the Laurel
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS Style Sheet
import '../../../css/Main_Line/laurel.css';

// Import Images
// Switch Images
// Images for a 135 Crossover
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

// Images for a 225 Crossover
import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
```

```
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Laurel Interlocking
 *
 * This class is a JSX React Component for the Laurel Interlocking,
this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Laurel extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
```

```
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        sw_7: this.props.status.sw_7,
        sw_9: this.props.status.sw_9,
        sw_11: this.props.status.sw_11,
        sw_13: this.props.status.sw_13,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_135,
        sw_3_src: CX_135,
        sw_7_src: CX_225,
        sw_11_src: CX_225,
        sw_13_src: CX_135,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_8w_src: SIG_W,
        sig_10w_src: SIG_W,
        sig_4e_src: SIG_E,
        sig_6e_src: SIG_E,
        sig_8e_src: SIG_E,
        sig_12e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_3_e: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        tail_4_e: Empty,
        tail_3_center: Empty,
        tail_3_w: Empty,
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_4_w: Empty,
        // Information For Interlocking Routes
        routes: this.props.status.routes,
        routed_1: this.props.status.routed_1,
        routed_2: this.props.status.routed_2,
        routed_3: this.props.status.routed_3,
        routed_4: this.props.status.routed_4,
        occupied_1: this.props.status.occupied_1,
        occupied_2: this.props.status.occupied_2,
        occupied_3: this.props.status.occupied_3,
        occupied_4: this.props.status.occupied_4,
    };
```

```
    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            sw_7: nextProps.status.sw_7,
            sw_11: nextProps.status.sw_11,
            sw_13: nextProps.status.sw_13,

            routed_1: nextProps.status.routed_1,
            routed_2: nextProps.status.routed_2,
            routed_3: nextProps.status.routed_3,
            routed_4: nextProps.status.routed_4,
            occupied_1: nextProps.status.occupied_1,
            occupied_2: nextProps.status.occupied_2,
            occupied_3: nextProps.status.occupied_3,
            occupied_4: nextProps.status.occupied_4,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----


    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
```

```
                <div className="laurel_title">LAUREL</div>
                <div className="laurel_milepost">MP 4.3</div>

                {/* West Side Tail Tracks */}
                <div className="b_laurel_3_west" style={{background:
this.state.tail_3_w}}></div>
                <div className="b_laurel_2_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="m_laurel_2_west" style={{background:
this.state.tail_2_w}}></div>
                <div className="m_laurel_4_west" style={{background:
this.state.tail_4_w}}></div>

                {/* Switches */}
                <div className="laurel_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="laurel_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="laurel_SW_7"
onClick={this.props.throw_sw_7}><img src={this.state.sw_7_src}/></div>
                <div className="laurel_SW_11"
onClick={this.props.throw_sw_11}><img src={this.state.sw_11_src}/></
div>
                <div className="laurel_SW_13"
onClick={this.props.throw_sw_13}><img src={this.state.sw_13_src}/></
div>

                {/* Center Tail Tracks */}
                <div className="m_laurel_3_center" style={{background:
this.state.tail_3_center}}></div>

                {/* East Side Tail Tracks */}
                <div className="m_laurel_3_east" style={{background:
this.state.tail_3_e}}></div>
                <div className="m_laurel_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="m_laurel_2_east" style={{background:
this.state.tail_2_e}}></div>
                <div className="m_laurel_4_east" style={{background:
this.state.tail_4_e}}></div>


                {/* Signals */}
                {/* West Signals */}
                <div className="laurel_sig_10w"
onClick={this.props.click_sig_10w}><img src={this.state.sig_10w_src}/
></div>
                <div className="laurel_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
```

```
                <div className="laurel_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="laurel_sig_8w"
onClick={this.props.click_sig_8w}><img src={this.state.sig_8w_src}/></
div>
                {/* East Signals */}
                <div className="laurel_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
                <div className="laurel_sig_6e"
onClick={this.props.click_sig_6e}><img src={this.state.sig_6e_src}/></
div>
                <div className="laurel_sig_8e"
onClick={this.props.click_sig_8e}><img src={this.state.sig_8e_src}/></
div>
                <div className="laurel_sig_12e"
onClick={this.props.click_sig_12e}><img src={this.state.sig_12e_src}/
></div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * set_route_drawings()
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or
not.
     *
     * There are a lot of possible drawings for this interlocking,
which is why the function is so long, I'm not sure if there
     * is a quicker or faster way to accomplish what this function
does
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;
        let color_3 = Empty;
        let color_4 = Empty;

        // Set Track Colors
        // If each track has a route
        if (this.state.routed_1) {
            color_1 = Green;
        }
```

```
        if (this.state.routed_2) {
            color_2 = Green;
        }
        if (this.state.routed_3) {
            color_3 = Green;
        }
        if (this.state.routed_4) {
            color_4 = Green;
        }
        // If each track is occupied
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }
        if (this.state.occupied_3) {
            color_3 = Red;
        }
        if (this.state.occupied_4) {
            color_4 = Red;
        }

        // Loop Through All The Routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__2_hx_laurel" ||
this.state.routes[i] === "E_1_1__|__1_laurel_westEnd") {
                // Setting Tail Track Color
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_1) {
                    // Switches
                    // Crossovers that could change based off of Track
#2
                    if (this.state.routes.includes("W_2_2__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_2_2__|
__2_laurel_westEnd")) {
                        // Track 2 Routed
                        if (this.state.routed_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                        // Track 2 Occupied
                        else if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
```

```
CX_225_Occupied_Both;
                        }
                    }
                    // Nothing On Track 2
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                        this.state.sw_7_src = CX_225_Occupied_Top;
                    }
                    // Crossovers that could changed based off of
Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track 3 Routed
                        if (this.state.routed_3) {
                            this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_11_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Track 3 Occupied
                        else if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
                            this.state.sw_11_src =
CX_225_Occupied_Both;
                        }
                    }
                    // Nothing on Track 3
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                        this.state.sw_11_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    // Crossovers that could change based off of Track
#2
                    if (this.state.routes.includes("W_2_2__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_2_2__|
__2_laurel_westEnd") || this.state.routes.includes("E_2_4__|
__4_laurel_westEnd")) {
                        // Track 2 Routed
                        if (this.state.routed_2) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
```

```
                        }
                        // Track 2 Occupied
                        else if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("W_4_2__|
__2_westSecaucus_laurel")) {
                        // Track 2 Routed
                        if (this.state.routed_4) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                        // Track 2 Occupied
                        else if (this.state.occupied_4) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing On Track 2
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
                        this.state.sw_7_src = CX_225_Lined_Top;
                    }
                    // Crossovers that could changed based off of
Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track 3 Routed
                        if (this.state.routed_3) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                            this.state.sw_11_src = CX_225_Lined_Both;
                        }
                        // Track 3 Occupied
                        else if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_11_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing on Track 3
                    else {
                        this.state.sw_3_src = CX_135_Lined_Bottom;
```

```
                        this.state.sw_11_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__2_hx_laurel") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_12e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_12e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_3__|__3_hx_laurel"
|| this.state.routes[i] === "E_3_3__|__3_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_3;
                this.state.tail_3_center = color_3;
                this.state.tail_3_w = color_3;

                // If The Route Is Occupied
                if (this.state.occupied_3) {
                    // Switches
                    this.state.sw_11_src = CX_225_Occupied_Top;

                    // Crossovers that could change based of the
status of other Track #1
                    if (this.state.routes.includes("W_4_1__|
__2_hx_laurel")) {
                        // Track #1 Is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
                        }
                        // Track #1 Is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_1_4__|
__4_laurel_westEnd")) {
                        // Track #1 Is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
```

```
                        }
                        // Track #1 Is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_11_src = CX_225_Lined_Top;

                    // Crossovers that could change based of the
status of other Track #1
                    if (this.state.routes.includes("W_4_1__|
__2_hx_laurel")) {
                        // Track #1 Is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #1 Is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_1_4__|
__4_laurel_westEnd")) {
                        // Track #1 Is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #1 Is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                        }
                    }
                    else {
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }
```

```
                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_3__|
__3_hx_laurel") {
                        this.state.sig_10w_src = SIG_W_Clear;
                        this.state.sig_6e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_10w_src = SIG_W_Stop;
                        this.state.sig_6e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|
__2_westSecaucus_laurel" || this.state.routes[i] === "E_2_2__|
__2_laurel_westEnd") {
                // Set Tail Track Color
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                // If The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    // Crossovers that could change based off of Tack
#1
                    if (this.state.routes.includes("W_1_1__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_1_1__|
__1_laurel_westEnd")) {
                        // Track 1 Routed
                        if (this.state.routed_1) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Track 1 Occupied
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    else if (this.state.routes.includes("W_3_1__|
__1_hx_laurel")) {
                        if (this.state.routed_3) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
```

```
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        else if (this.state.occupied_3) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_1_3__|
__3_laurel_westEnd")) {
                        if (this.state.routed_1) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    else if (this.state.routes.includes("W_1_3__|
__3_hx_laurel")) {
                        if (this.state.routed_1) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_3_1__|
__1_laurel_westEnd")) {
                        if (this.state.routed_3) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        else if (this.state.occupied_3) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
```

```
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    // Nothing Track 1
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                    }
                    // Crossovers that could change based off of Track
#4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track 4 Routed
                        if (this.state.routed_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track 4 Occupied
                        else if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Both;
                        }
                    }
                    // Nothing on Track 3
                    else {
                        this.state.sw_13_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    // Crossovers that could change based off of Tack
#1
                    if (this.state.routes.includes("W_1_1__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_1_1__|
__1_laurel_westEnd")) {
                        // Track 1 Routed
                        if (this.state.routed_1) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                        // Track 1 Occupied
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
```

```
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("W_3_1__|
__1_hx_laurel")) {
                        if (this.state.routed_3) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                        else if (this.state.occupied_3) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_1_3__|
__3_laurel_westEnd")) {
                        if (this.state.routed_1) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("W_1_3__|
__3_hx_laurel")) {
                        if (this.state.routed_1) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_3_1__|
__1_laurel_westEnd")) {
                        if (this.state.routed_1) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
```

```
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        else if (this.state.occupied_1) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                    }
                    // Nothing Track 1
                    else {
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }
                    // Crossovers that could changed based off of
Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track 4 Routed
                        if (this.state.routed_4) {
                            this.state.sw_13_src = CX_135_Lined_Both;
                        }
                        // Track 4 Occupied
                        else if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing on Track 3
                    else {
                        this.state.sw_13_src = CX_135_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_westSecaucus_laurel") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_4_4__|
```

```
__4_westSecaucus_laurel" || this.state.routes[i] === "E_4_4__|
__4_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_4_e = color_4;
                this.state.tail_4_w = color_4;

                // If The Route Is Occupied
                if (this.state.occupied_4) {
                    // Switches
                    // Crossovers that could change based on the
status of Track #4
                    if (this.state.routes.includes("E_3_2__|
__2_laurel_westEnd")) {
                        // Track #4 Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_13_src =
CX_135_Occupied_Both;
                        }
                        // Track #4 Routed
                        else if (this.state.routed_3) {
                            this.state.sw_13_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                }
                // The Route is NOT Occupied
                else {
                    // Switches
                    // Crossovers that could change based on the
status of Track #4
                    if (this.state.routes.includes("E_3_2__|
__2_laurel_westEnd")) {
                        // Track #4 Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_13_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #4 Routed
                        else if (this.state.routed_3) {
                            this.state.sw_13_src = CX_135_Lined_Both;
                        }
                    }
```

```
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_4_4__|
__4_westSecaucus_laurel") {
                        this.state.sig_8w_src = SIG_W_Clear;
                        this.state.sig_8e_src = SIG_E_Stop
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_8w_src = SIG_W_Stop;
                        this.state.sig_8e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_1__|__1_hx_laurel")
{
                // Set Tail Track Colors
                this.state.tail_3_e = color_3;
                this.state.tail_1_w = color_3;

                // The Route Is Occupied
                if (this.state.occupied_3) {
                    // Switches
                    this.state.sw_3_src = CX_135_Occupied_Bottom;
                    this.state.sw_11_src = CX_225_R_Occupied;

                    if (this.state.routes.includes("W_4_2__|
__2_westSecaucus_laurel")) {
                        if (this.state.occupied_4) {
                            this.state.sw_1_src =
CX_135_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Bottom;
                        }
                        else if (this.state.routed_4) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                        this.state.sw_7_src = CX_225_Occupied_Top;
                    }
```

```
                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_135_Lined_Bottom;
                    this.state.sw_11_src = CX_225_R_Lined;

                    // Crossovers that could change based on other
tracks
                    if (this.state.routes.includes("W_4_2__|
__2_westSecaucus_laurel")) {
                        // Other track is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other track is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_2_4__|
__4_laurel_westEnd")) {
                        // Other track is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other track is Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
                        this.state.sw_7_src = CX_225_Lined_Top;
                    }
```

```
                    // Signals
                    this.state.sig_10w_src = SIG_W_Clear;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_3__|
__3_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_135_Occupied_Bottom;
                    this.state.sw_11_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Top;
                    this.state.sw_7_src = CX_225_Occupied_Top;

                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_135_Lined_Bottom;
                    this.state.sw_11_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Top;
                    this.state.sw_7_src = CX_225_Lined_Top;

                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_3_2__|
__2_westSecaucus_laurel") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_3;
                this.state.tail_2_w = color_3;

                // The Route Is Occupied
                if (this.state.occupied_3) {
```

```
                    // Switches
                    this.state.sw_11_src = CX_225_R_Occupied;
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_11_src = CX_225_R_Lined;
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_10w_src = SIG_W_Clear;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_3__|
__3_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_11_src = CX_225_R_Occupied;
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
```

```
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_11_src = CX_225_R_Lined;
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_westSecaucus_laurel") {
                // Set Tail Track Colors
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Switches
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Crossovers that could change based of Track #3
Status
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Occupied Track 3
                        if (this.state.occupied_3) {
                            this.state.sw_11_src =
CX_225_Occupied_Both;
                        }
                        // Lined Track 3
                        else if (this.state.routed_3) {
                            this.state.sw_11_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track 3
                    else {
```

```
                        this.state.sw_11_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Crossovers that could change based of Track #3
Status
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Occupied Track 3
                        if (this.state.occupied_3) {
                            this.state.sw_11_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                        // Lined Track 3
                        else if (this.state.routed_3) {
                            this.state.sw_11_src = CX_225_Lined_Both;
                        }
                    }
                    // Nothing Track 3
                    else {
                        this.state.sw_11_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|
__1_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
```

```
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    // Switches
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Crossovers that could change based of Track #3
Status
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Occupied Track 3
                        if (this.state.occupied_3) {
                            this.state.sw_11_src =
CX_225_Occupied_Both;
                        }
                        // Lined Track 3
                        else if (this.state.routed_3) {
                            this.state.sw_11_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track 3
                    else {
                        this.state.sw_11_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    // Crossovers that could change based of Track #3
Status
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Occupied Track 3
                        if (this.state.occupied_3) {
                            this.state.sw_11_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                        // Lined Track 3
```

```
                        else if (this.state.routed_3) {
                            this.state.sw_11_src = CX_225_Lined_Both;
                        }
                    }
                    // Nothing Track 3
                    else {
                        this.state.sw_11_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_1_3__|__3_hx_laurel")
{
                // Set Tail Track Colors
                this.state.tail_1_e = color_1;
                this.state.tail_3_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_11_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Crossovers that could change based off of Track
#3 status
                    if (this.state.routes.includes("W_2_2__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_2_2__|
__2_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Another Possible Route
                    else if (this.state.routes.includes("W_4_2__|
```

```
__2_westSecaucus_laurel")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                        // Other Track Routed
                        else if (this.state.routed_4) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_2_4__|
__4_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing On The Other Track
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                        this.state.sw_7_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_11_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;
```

```
                    // Crossovers that could change based off of Track
#3 status
                    if (this.state.routes.includes("W_2_2__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_2_2__|
__2_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    // Another Possible Route
                    else if (this.state.routes.includes("W_4_2__|
__2_westSecaucus_laurel")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other Track Routed
                        else if (this.state.routed_4) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_2_4__|
__4_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
```

```
                    // Nothing On The Other Track
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
                        this.state.sw_7_src = CX_225_Lined_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Clear;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_1__|
__1_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_1_e = color_3;
                this.state.tail_3_w = color_3;

                // The Route Is Occupied
                if (this.state.occupied_3) {
                    // Switches
                    this.state.sw_11_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Crossovers that could change based off of Track
#3 status
                    if (this.state.routes.includes("W_2_2__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_2_2__|
__2_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Another Possible Route
                    else if (this.state.routes.includes("W_4_2__|
__2_westSecaucus_laurel")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_4) {
```

```
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                        // Other Track Routed
                        else if (this.state.routed_4) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_2_4__|
__4_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing On The Other Track
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                        this.state.sw_7_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_11_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Crossovers that could change based off of Track
#3 status
```

```
                    if (this.state.routes.includes("W_2_2__|
__2_westSecaucus_laurel") || this.state.routes.includes("E_2_2__|
__2_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    // Another Possible Route
                    else if (this.state.routes.includes("W_4_2__|
__2_westSecaucus_laurel")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other Track Routed
                        else if (this.state.routed_4) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_2_4__|
__4_laurel_westEnd")) {
                        // Other Track Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Other Track Routed
                        else if (this.state.routed_2) {
                            this.state.sw_1_src = CX_135_Lined_Both;
                            this.state.sw_7_src = CX_225_Lined_Both;
                        }
                    }
                    // Nothing On The Other Track
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
```

```
                        this.state.sw_7_src = CX_225_Lined_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Clear;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_2_1__|__2_hx_laurel")
{
                // Set Tail Track Colors
                this.state.tail_2_e = color_2;
                this.state.tail_1_w = color_2;

                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_1_src = CX_135_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;

                    // Crossovers that could change based on the
status of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
                        }
                        // Track #3 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track #3
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Crossovers that could change based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track #4 is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
```

```
CX_135_Occupied_Both;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_1_src = CX_135_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;

                    // Crossovers that could change based on the
status of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #3 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #3
                    else {
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Crossovers that could change based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track #4 is Occupied
```

```
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_2_e = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_1_src = CX_135_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;

                    // Crossovers that could change based on the
status of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
                        }
                        // Track #3 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track #3
                    else {
```

```
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Crossovers that could change based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track #4 is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Both;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_1_src = CX_135_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;

                    // Crossovers that could change based on the
status of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #3 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                        }
                    }
```

```
                    // Nothing Track #3
                    else {
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Crossovers that could change based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track #4 is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_12e_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_2_3__|__3_hx_laurel")
{
                // Set Tail Track Colors
                this.state.tail_2_e = color_2;
                this.state.tail_3_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Crossovers taht could changed based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
```

```
                        // Track #4 is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Both;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Crossovers taht could changed based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track #4 is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Lined_Top;
                    }
```

```
                    // Signals
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_2__|
__2_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_2_e = color_3;
                this.state.tail_3_w = color_3;

                // The Route Is Occupied
                if (this.state.occupied_3) {
                    // Switches
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Crossovers taht could changed based on the
status of Track #4
                    if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                        // Track #4 is Occupied
                        if (this.state.occupied_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Both;
                        }
                        // Track #4 is Routed
                        else if (this.state.routed_4) {
                            this.state.sw_13_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing Track #4
                    else {
                        this.state.sw_13_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
```

```
                this.state.sig_4e_src = SIG_E_Stop;
            }
            // The Route Is NOT Occupied
            else {
                // Switches
                this.state.sw_7_src = CX_225_Lined_Bottom;
                this.state.sw_1_src = CX_135_R_Lined;
                this.state.sw_3_src = CX_135_R_Lined;

                // Crossovers taht could changed based on the
status of Track #4
                if (this.state.routes.includes("W_4_4__|
__4_westSecaucus_laurel") || this.state.routes.includes("E_4_4__|
__4_laurel_westEnd")) {
                    // Track #4 is Occupied
                    if (this.state.occupied_4) {
                        this.state.sw_13_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    // Track #4 is Routed
                    else if (this.state.routed_4) {
                        this.state.sw_13_src = CX_135_Lined_Both;
                    }
                }
                // Nothing Track #4
                else {
                    this.state.sw_13_src = CX_135_Lined_Top;
                }

                // Signals
                this.state.sig_4w_src = SIG_W_Stop;
                this.state.sig_2w_src = SIG_W_Stop;
                this.state.sig_10w_src = SIG_W_Stop;
                this.state.sig_6e_src = SIG_E_Clear;
                this.state.sig_12e_src = SIG_E_Stop;
                this.state.sig_4e_src = SIG_E_Stop;
            }
        }
        else if (this.state.routes[i] === "W_4_2__|
__2_westSecaucus_laurel") {
            // Set Tail Track Colors
            this.state.tail_4_e = color_4;
            this.state.tail_2_w = color_4;

            if (this.state.occupied_4) {
                // Switches
                this.state.sw_13_src = CX_135_R_Occupied;

                // Crossovers that could change based on the
status of Track #1
```

```
                    if (this.state.routes.includes("W_1_1__|
__1_hx_laurel") || this.state.routes.includes("E_1_1__|
__1_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_3_1__|
__1_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_1_3__|
__3_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
```

```
                    }
                }
                // Nothing Track #1
                else {
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;
                }

                // Signals
                this.state.sig_8w_src = SIG_W_Stop;
                this.state.sig_4w_src = SIG_W_Stop;
                this.state.sig_4e_src = SIG_E_Stop;
                this.state.sig_8e_src = SIG_E_Stop;
            }
            else {
                // Switches
                this.state.sw_13_src = CX_135_R_Lined;

                // Crossovers that could change based on the
status of Track #1
                if (this.state.routes.includes("W_1_1__|
__1_hx_laurel") || this.state.routes.includes("E_1_1__|
__1_laurel_westEnd")) {
                    // Track #1 is Occupied
                    if (this.state.occupied_1) {
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    // Track #1 is Routed
                    else if (this.state.routed_1) {
                        this.state.sw_7_src = CX_225_Lined_Both;
                        this.state.sw_1_src = CX_135_Lined_Both;
                    }
                }
                else if (this.state.routes.includes("E_3_1__|
__1_laurel_westEnd")) {
                    // Track #1 is Occupied
                    if (this.state.occupied_3) {
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    // Track #1 is Routed
                    else if (this.state.routed_3) {
                        this.state.sw_7_src = CX_225_Lined_Both;
                        this.state.sw_1_src = CX_135_Lined_Both;
                    }
```

```
                    }
                    else if (this.state.routes.includes("E_1_3__|
__3_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src = CX_225_Lined_Both;
                            this.state.sw_1_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_4__|
__4_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_4_e = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Occupied;

                    // Crossovers that could change based on the
status of Track #1
                    if (this.state.routes.includes("W_1_1__|
__1_hx_laurel") || this.state.routes.includes("E_1_1__|
__1_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                            this.state.sw_1_src =
CX_135_Occupied_Both;
```

```
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_3_1__|
__1_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    else if (this.state.routes.includes("E_1_3__|
__3_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Both;
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
```

```
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Lined;

                    // Crossovers that could change based on the
status of Track #1
                    if (this.state.routes.includes("W_1_1__|
__1_hx_laurel") || this.state.routes.includes("E_1_1__|
__1_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src = CX_225_Lined_Both;
                            this.state.sw_1_src = CX_135_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_3_1__|
__1_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_7_src = CX_225_Lined_Both;
                            this.state.sw_1_src = CX_135_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("E_1_3__|
__3_laurel_westEnd")) {
                        // Track #1 is Occupied
                        if (this.state.occupied_1) {
                            this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
```

```
                        }
                        // Track #1 is Routed
                        else if (this.state.routed_1) {
                            this.state.sw_7_src = CX_225_Lined_Both;
                            this.state.sw_1_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                    this.state.sig_8e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_4_1__|__2_hx_laurel")
{
                // Set Tail Track Colors
                this.state.tail_4_e = color_4;
                this.state.tail_1_w = color_4;

                // The Route Is Occupied
                if (this.state.occupied_4) {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Crossovers that could change based on the state
of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
                        }
                        // Track #3 Is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track #3
```

```
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Crossovers that could change based on the state
of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #3 Is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #3
                    else {
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_4__|
__4_laurel_westEnd") {
```

```
                // Set Tail Track Colors
                this.state.tail_4_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Crossovers that could change based on the state
of Track #3
                    if (this.state.routes.includes("W_3_3__|
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Both;
                        }
                        // Track #3 Is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        }
                    }
                    // Nothing Track #3
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Crossovers that could change based on the state
of Track #3
                    if (this.state.routes.includes("W_3_3__|
```

```
__3_hx_laurel") || this.state.routes.includes("E_3_3__|
__3_laurel_westEnd")) {
                        // Track #3 is Occupied
                        if (this.state.occupied_3) {
                            this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        // Track #3 Is Routed
                        else if (this.state.routed_3) {
                            this.state.sw_3_src = CX_135_Lined_Both;
                        }
                    }
                    // Nothing Track #3
                    else {
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_4_3__|__3_hx_laurel")
{
                // Set Tail Track Colors
                this.state.tail_4_e = color_4;
                this.state.tail_3_w = color_4;

                if (this.state.occupied_4) {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                else {
```

```
                    // Switches
                    this.state.sw_13_src = CX_135_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_8w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_4__|
__4_laurel_westEnd") {
                // Set Tail Track Colors
                this.state.tail_4_e = color_3;
                this.state.tail_3_w = color_3;

                if (this.state.occupied_3) {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_13_src = CX_135_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_8w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
```

```
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_10w_src = SIG_W_Stop;
                    this.state.sig_8e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Clear;
                    this.state.sig_12e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----


    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set the state of SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_135_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_135;
        }

        // Set the state of SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_135_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = CX_135;
        }

        // Set the state of SW #7
        // SW #7 Reversed
        if (this.state.sw_7) {
            this.state.sw_7_src = CX_225_R;
        }
```

```
        // SW #7 Normal
        else {
            this.state.sw_7_src = CX_225;
        }

        // Set the state of SW #9
        // SW #9 Reversed
        if (this.state.sw_9) {
            this.state.sw_9_src = CX_135_R;
        }
        // SW #9 Normal
        else {
            this.state.sw_9_src = CX_135;
        }

        // Set the state of SW #11
        // SW #11 Reversed
        if (this.state.sw_11) {
            this.state.sw_11_src = CX_225_R;
        }
        // SW #11 Normal
        else {
            this.state.sw_11_src = CX_225;
        }

        // Set the state of SW #13
        // SW #13 Reversed
        if (this.state.sw_13) {
            this.state.sw_13_src = CX_135_R;
        }
        // SW #13 Normal
        else {
            this.state.sw_13_src = CX_135;
        }
    }
    // ---- END set_switch_image() ----


    /**
     * reset_drawings()
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
```

```
        this.state.sig_2w_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_8w_src = SIG_W;
        this.state.sig_10w_src = SIG_W;
        this.state.sig_4e_src = SIG_E;
        this.state.sig_6e_src = SIG_E;
        this.state.sig_8e_src = SIG_E;
        this.state.sig_12e_src = SIG_E;

        this.state.tail_3_e = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_4_e = Empty;
        this.state.tail_3_center = Empty;
        this.state.tail_3_w = Empty;
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_4_w = Empty;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Laurel;
```

```
/**
 * @file MainLineTracks.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @brief React JSX Component Class that is for The Tracks of the Main
Line
 *
 * Extends the React Component Class and is the UI part of the Main
Line Tracks,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/mainLine.css';


/**
 * The React JSX Component Class for the Tracks in the Main Line
portion
 *
 * This class is a JSX React Component for the Main Line Tracks, this
will control all the UI for the comonent,
 * showing what blocks are occupied by a train
 */
class MainLineTracks extends Component {
    /**
     * State
     * @brief Object that holds the state or status information for
the component
     *
     * This object holds all the information for the tracks that is
required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Symbols
        symbol_sterling_sf_1: this.props.symbols.symbol_sterling_sf_1,
        symbol_sterling_hilburn_2:
this.props.symbols.symbol_sterling_hilburn_2,
        symbol_hilburn_sf_2: this.props.symbols.symbol_hilburn_sf_2,
        symbol_hilburn_yardWest:
this.props.symbols.symbol_hilburn_yardWest,
        symbol_hilburn_yardEast:
```

```
this.props.symbols.symbol_hilburn_yardEast,
        symbol_sf_wc_1: this.props.symbols.symbol_sf_wc_1,
        symbol_sf_wc_2: this.props.symbols.symbol_sf_wc_2,
        symbol_wc_yard: this.props.symbols.symbol_wc_yard,
        symbol_wc_ridgewood_1:
this.props.symbols.symbol_wc_ridgewood_1,
        symbol_wc_ridgewood_2:
this.props.symbols.symbol_wc_ridgewood_2,
        symbol_wc_ridgewood_3:
this.props.symbols.symbol_wc_ridgewood_3,
        // Second Row
        symbol_ridgewood_suscon_1:
this.props.symbols.symbol_ridgewood_suscon_1,
        symbol_ridgewood_suscon_2:
this.props.symbols.symbol_ridgewood_suscon_2,
        symbol_suscon_mill_1: this.props.symbols.symbol_suscon_mill_1,
        symbol_suscon_mill_2: this.props.symbols.symbol_suscon_mill_2,
        symbol_mill_westSecaucus_1:
this.props.symbols.symbol_mill_westSecaucus_1,
        symbol_mill_westSecaucus_2:
this.props.symbols.symbol_mill_westSecaucus_2,
        symbol_westSecaucus_laurel_1:
this.props.symbols.symbol_westSecaucus_laurel_1,
        symbol_westSecaucus_laurel_2:
this.props.symbols.symbol_westSecaucus_laurel_2,
        symbol_laurel_westEnd_1:
this.props.symbols.symbol_westEnd_laurel_1,
        symbol_laurel_westEnd_2:
this.props.symbols.symbol_westEnd_laurel_2,
        symbol_laurel_westEnd_3:
this.props.symbols.symbol_westEnd_laurel_3,
        symbol_laurel_westEnd_4:
this.props.symbols.symbol_westEnd_laurel_4,


        // Blocks
        westEnd_laurel_1: this.props.blocks.block_westEnd_laurel_1,
        westEnd_laurel_2: this.props.blocks.block_westEnd_laurel_2,
        westEnd_laurel_3: this.props.blocks.block_westEnd_laurel_3,
        westEnd_laurel_4: this.props.blocks.block_westEnd_laurel_4,

        laurel_westSecaucus_1:
this.props.blocks.block_laurel_westSecaucus_1,
        laurel_westSecaucus_2:
this.props.blocks.block_laurel_westSecaucus_2,

        mill_westSecaucus_1:
this.props.blocks.block_mill_westSecaucus_1,
        mill_westSecaucus_2:
this.props.blocks.block_mill_westSecaucus_2,
```

```
        suscon_mill_1: this.props.blocks.block_suscon_mill_1,
        suscon_mill_2: this.props.blocks.block_suscon_mill_2,

        ridgewood_suscon_1:
this.props.blocks.block_ridgewood_suscon_1,
        ridgewood_suscon_2:
this.props.blocks.block_ridgewood_suscon_2,

        wc_ridgewood_3: this.props.blocks.block_wc_ridgewood_3,
        wc_ridgewood_1: this.props.blocks.block_wc_ridgewood_1,
        wc_ridgewood_2: this.props.blocks.block_wc_ridgewood_2,

        sf_wc_1: this.props.blocks.block_sf_wc_1,
        sf_wc_2: this.props.blocks.block_sf_wc_2,

        hilburn_sf: this.props.blocks.block_hilburn_sf,

        sterling_sf: this.props.blocks.block_sterling_sf,
        sterling_hilburn: this.props.blocks.block_sterling_hilburn,

        //Yard Leads
        hilburn_yard_west: this.props.blocks.block_hilburn_yard_west,
        hilburn_yard_east: this.props.blocks.block_hilburn_yard_east,
        wc_yard: this.props.blocks.block_wc_yard
    };

    /**
     * componentWillReceiveProps()
     * @brief Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            // Symbols
            // Symbols
            symbol_sterling_sf_1:
nextProps.symbols.symbol_sterling_sf_1,
            symbol_sterling_hilburn_2:
nextProps.symbols.symbol_sterling_hilburn_2,
            symbol_hilburn_sf_2:
nextProps.symbols.symbol_hilburn_sf_2,
            symbol_hilburn_yardWest:
nextProps.symbols.symbol_hilburn_yardWest,
            symbol_hilburn_yardEast:
nextProps.symbols.symbol_hilburn_yardEast,
```

```
            symbol_sf_wc_1: nextProps.symbols.symbol_sf_wc_1,
            symbol_sf_wc_2: nextProps.symbols.symbol_sf_wc_2,
            symbol_wc_yard: nextProps.symbols.symbol_wc_yard,
            symbol_wc_ridgewood_1:
nextProps.symbols.symbol_wc_ridgewood_1,
            symbol_wc_ridgewood_2:
nextProps.symbols.symbol_wc_ridgewood_2,
            symbol_wc_ridgewood_3:
nextProps.symbols.symbol_wc_ridgewood_3,
            // Second Row
            symbol_ridgewood_suscon_1:
nextProps.symbols.symbol_ridgewood_suscon_1,
            symbol_ridgewood_suscon_2:
nextProps.symbols.symbol_ridgewood_suscon_2,
            symbol_suscon_mill_1:
nextProps.symbols.symbol_suscon_mill_1,
            symbol_suscon_mill_2:
nextProps.symbols.symbol_suscon_mill_2,
            symbol_mill_westSecaucus_1:
nextProps.symbols.symbol_mill_westSecaucus_1,
            symbol_mill_westSecaucus_2:
nextProps.symbols.symbol_mill_westSecaucus_2,
            symbol_westSecaucus_laurel_1:
nextProps.symbols.symbol_westSecaucus_laurel_1,
            symbol_westSecaucus_laurel_2:
nextProps.symbols.symbol_westSecaucus_laurel_2,
            symbol_laurel_westEnd_1:
nextProps.symbols.symbol_laurel_westEnd_1,
            symbol_laurel_westEnd_2:
nextProps.symbols.symbol_laurel_westEnd_2,
            symbol_laurel_westEnd_3:
nextProps.symbols.symbol_laurel_westEnd_3,
            symbol_laurel_westEnd_4:
nextProps.symbols.symbol_laurel_westEnd_4,

            // Blocks
            westEnd_laurel_1: nextProps.blocks.block_westEnd_laurel_1,
            westEnd_laurel_2: nextProps.blocks.block_westEnd_laurel_2,
            westEnd_laurel_3: nextProps.blocks.block_westEnd_laurel_3,
            westEnd_laurel_4: nextProps.blocks.block_westEnd_laurel_4,

            laurel_westSecaucus_1:
nextProps.blocks.block_laurel_westSecaucus_1,
            laurel_westSecaucus_2:
nextProps.blocks.block_laurel_westSecaucus_2,

            mill_westSecaucus_1:
nextProps.blocks.block_mill_westSecaucus_1,
            mill_westSecaucus_2:
nextProps.blocks.block_mill_westSecaucus_2,
```

```
            suscon_mill_1: nextProps.blocks.block_suscon_mill_1,
            suscon_mill_2: nextProps.blocks.block_suscon_mill_2,

            ridgewood_suscon_1:
nextProps.blocks.block_ridgewood_suscon_1,
            ridgewood_suscon_2:
nextProps.blocks.block_ridgewood_suscon_2,

            wc_ridgewood_3: nextProps.blocks.block_wc_ridgewood_3,
            wc_ridgewood_1: nextProps.blocks.block_wc_ridgewood_1,
            wc_ridgewood_2: nextProps.blocks.block_wc_ridgewood_2,

            sf_wc_1: nextProps.blocks.block_sf_wc_1,
            sf_wc_2: nextProps.blocks.block_sf_wc_2,

            hilburn_sf: nextProps.blocks.block_hilburn_sf,

            sterling_sf: nextProps.blocks.block_sterling_sf,
            sterling_hilburn: nextProps.blocks.block_sterling_hilburn,

            //Yard Leads
            hilburn_yard_west:
nextProps.blocks.block_hilburn_yard_west,
            hilburn_yard_east:
nextProps.blocks.block_hilburn_yard_east,
            wc_yard: nextProps.blocks.block_wc_yard
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @brief standard React function that draws the interlocking to
the screen
     */
    render() {
        return (
            <div>
                {/* Tags */}
                <div className="wc_yard_tag">Waldwick Yard</div>
                <div className="hilburn_yard_tag">Hilburn Yard</div>

                {/* Symbols */}
                {/* First Row */}
                <div
className="symbol_sterling_sf_1">{this.state.symbol_sterling_sf_1}</
div>
                <div
className="symbol_sterling_hilburn_2">{this.state.symbol_sterling_hilb
```

```
urn_2}</div>
                <div
className="symbol_hilburn_sf_2">{this.state.symbol_hilburn_sf_2}</div>
                <div
className="symbol_hilburn_yardWest">{this.state.symbol_hilburn_yardWes
t}</div>
                <div
className="symbol_hilburn_yardEast">{this.state.symbol_hilburn_yardEas
t}</div>
                <div
className="symbol_sf_wc_1">{this.state.symbol_sf_wc_1}</div>
                <div
className="symbol_sf_wc_2">{this.state.symbol_sf_wc_2}</div>
                <div
className="symbol_wc_yard">{this.state.symbol_wc_yard}</div>
                <div
className="symbol_wc_ridgewood_1">{this.state.symbol_wc_ridgewood_1}</
div>
                <div
className="symbol_wc_ridgewood_2">{this.state.symbol_wc_ridgewood_2}</
div>
                <div
className="symbol_wc_ridgewood_3">{this.state.symbol_wc_ridgewood_3}</
div>
                {/* Second Row */}
                <div
className="symbol_ridgewood_suscon_1">{this.state.symbol_ridgewood_sus
con_1}</div>
                <div
className="symbol_ridgewood_suscon_2">{this.state.symbol_ridgewood_sus
con_2}</div>
                <div
className="symbol_suscon_mill_1">{this.state.symbol_suscon_mill_1}</
div>
                <div
className="symbol_suscon_mill_2">{this.state.symbol_suscon_mill_2}</
div>
                <div
className="symbol_mill_westSecaucus_1">{this.state.symbol_mill_westSec
aucus_1}</div>
                <div
className="symbol_mill_westSecaucus_2">{this.state.symbol_mill_westSec
aucus_2}</div>
                <div
className="symbol_westSecaucus_laurel_1">{this.state.symbol_westSecauc
us_laurel_1}</div>
                <div
className="symbol_westSecaucus_laurel_2">{this.state.symbol_westSecauc
us_laurel_2}</div>
                <div
```

```
className="symbol_laurel_westEnd_4">{this.state.symbol_laurel_westEnd_
3}</div>
                <div
className="symbol_laurel_westEnd_3">{this.state.symbol_laurel_westEnd_
1}</div>
                <div
className="symbol_laurel_westEnd_1">{this.state.symbol_laurel_westEnd_
2}</div>
                <div
className="symbol_laurel_westEnd_2">{this.state.symbol_laurel_westEnd_
4}</div>

                {/* First Row */}
                {/* West End to Laurel */}
                <div className="m_westEnd_laurel_1"
style={{background: this.state.westEnd_laurel_2}}></div>
                <div className="m_westEnd_laurel_2"
style={{background: this.state.westEnd_laurel_4}}></div>
                <div className="m_westEnd_laurel_3"
style={{background: this.state.westEnd_laurel_1}}></div>
                <div className="m_westEnd_laurel_4"
style={{background: this.state.westEnd_laurel_3}}></div>

                {/* Laurel to West Secaucus */}
                <div className="m_laurel_westSecaucus_1"
style={{background: this.state.laurel_westSecaucus_1}}></div>
                <div className="m_laurel_westSecaucus_2"
style={{background: this.state.laurel_westSecaucus_2}}></div>

                {/* West Secaucus To Mill */}
                <div className="m_westSecaucus_mill_1"
style={{background: this.state.mill_westSecaucus_1}}></div>
                <div className="m_westSecaucus_mill_2"
style={{background: this.state.mill_westSecaucus_2}}></div>

                {/* Mill to Suscon */}
                <div className="m_suscon_mill_1" style={{background:
this.state.suscon_mill_1}}></div>
                <div className="m_suscon_mill_2" style={{background:
this.state.suscon_mill_2}}></div>

                {/* Suscon to Ridgewood Junction */}
                <div className="m_ridgewood_suscon_1"
style={{background: this.state.ridgewood_suscon_1}}></div>
                <div className="m_ridgewood_suscon_2"
style={{background: this.state.ridgewood_suscon_2}}></div>

                {/* Ridgewood Junction to Screen */}
                <div className="m_screen_ridgewood_3"
style={{background: this.state.wc_ridgewood_3}}></div>
```

```
                <div className="m_screen_ridgewood_1"
style={{background: this.state.wc_ridgewood_1}}></div>
                <div className="m_screen_ridgewood_2"
style={{background: this.state.wc_ridgewood_2}}></div>


                {/* Second Row */}
                {/* Screen to WC */}
                <div className="m_wc_screen_3" style={{background:
this.state.wc_ridgewood_3}}></div>
                <div className="m_wc_screen_1" style={{background:
this.state.wc_ridgewood_1}}></div>
                <div className="m_wc_screen_2" style={{background:
this.state.wc_ridgewood_2}}></div>

                <div className="m_wc_yard " style={{background:
this.state.wc_yard}}></div>

                {/* WC to SF */}
                <div className="m_sf_wc_1" style={{background:
this.state.sf_wc_1}}></div>
                <div className="m_sf_wc_2" style={{background:
this.state.sf_wc_2}}></div>

                {/* SF to Hilburn */}
                <div className="m_hilburn_sf" style={{background:
this.state.hilburn_sf}}></div>

                {/* Hilburn Yard */}
                <div className="m_hilburn_yard_west"
style={{background: this.state.hilburn_yard_west}}></div>
                <div className="m_hilburn_yard_east"
style={{background: this.state.hilburn_yard_east}}></div>

                {/* SF to Sterling [Track 1]*/}
                <div className="m_sterling_sf_1" style={{background:
this.state.sterling_sf}}></div>

                {/* Hilburn to Sterling [Track 2] */}
                <div className="m_sterling_hilburn_2"
style={{background: this.state.sterling_hilburn}}></div>
            </div>
        );
    }
    // ---- END render() ----
}

// Export the tracks to be drawn on the screen
export default MainLineTracks;
```

```
/**
 * @file Mill.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Mill Interlocking
 *
 * Extends the React Component Class and is the UI part of the Mill
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS Style Sheet
import '../../../css/Main_Line/mill.css';

// Import Images
// Switch Images
// Images for a 135 Crossover
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

// Images for a 225 Crossover
import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
```

```
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Mill Interlocking
 *
 * This class is a JSX React Component for the Mill Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Mill extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
```

```
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_225,
        sw_3_src: CX_135,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_e: Empty,
        tail_1_w: Empty,
        tail_2_e: Empty,
        tail_2_w: Empty,
        // Information For Interlocking Routes
        occupied_trk_1: this.props.status.occupied_trk_1,
        occupied_trk_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_trk_1,
        route_2: this.props.status.routed_trk_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,

            route_1: nextProps.status.routed_trk_1,
            route_2: nextProps.status.routed_trk_2,
            occupied_trk_1: nextProps.status.occupied_trk_1,
            occupied_trk_2: nextProps.status.occupied_trk_2,
            routes: nextProps.status.routes
        });
```

```
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="mill_title">MILL</div>
                <div className="mill_milepost">MP 11.1</div>
                {/* East Side Tail Tracks */}
                <div className="mill_1_east" style={{background:
this.state.tail_1_w}}></div>
                <div className="mill_2_east" style={{background:
this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="mill_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="mill_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* West Side Tail Tracks */}
                <div className="mill_1_west" style={{background:
this.state.tail_1_e}}></div>
                <div className="mill_2_west" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="mill_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="mill_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
                <div className="mill_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="mill_sig_4w"
```

```
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * set_route_drawings()
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or
not.
     */
    set_route_drawing() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Set Track Colors
        // If each track has a route
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        // If each track is occupied
        if (this.state.occupied_trk_1) {
            color_1 = Red;
        }
        if (this.state.occupied_trk_2) {
            color_2 = Red;
        }

        // Loop Through All The Routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_suscon_mill" ||
this.state.routes[i] === "E_1_1__|__1_mill_westSecaucus") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                // If the Route Is Occupied
                if (this.state.occupied_trk_1) {
                    // Switches
                    // Track #2 is Routed
                    if (this.state.route_2) {
```

```
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    // Track #2 is Occupied
                    else if (this.state.occupied_trk_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Track #2 is Routed
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    // Track #2 is Occupied
                    else if (this.state.occupied_trk_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_suscon_mill") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
```

```
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|
__2_suscon_mill" || this.state.routes[i] === "E_2_2__|
__2_mill_westSecaucus") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switches
                    // Track #1 is Routed
                    if (this.state.route_1) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    // Track #1 is Occupied
                    else if (this.state.occupied_trk_1) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Bottom;
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w = SIG_W_Stop;
                    this.state.sig_4e = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Track #1 is Routed
                    if (this.state.route_1) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    // Track #1 is Occupied
                    else if (this.state.occupied_trk_1) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
```

```
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_1_src = CX_225_Lined_Bottom;
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_suscon_mill") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_suscon_mill") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_trk_1) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
```

```
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|
__1_mill_westSecaucus") {
                // Tail Tracks
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__1_suscon_mill") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_1_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
```

```
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_mill_westSecaucus") {
                // Tail Tracks
                this.state.tail_2_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----
```

```
    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_225_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_225;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_135_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = CX_135;
        }
    }
    // ---- END set_switch_image() ----


    /**
     * reset_drawings()
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_e = Empty;
        this.state.tail_1_w = Empty;
        this.state.tail_2_e = Empty;
```

```
        this.state.tail_2_w = Empty;

        this.state.sig_2e_src = SIG_E;
        this.state.sig_2w_src = SIG_W;
        this.state.sig_4e_src = SIG_E;
        this.state.sig_4w_src = SIG_W;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Mill;
```

```
/**
 * @file RidgewoodJunction.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Ridgewood Junction
Interlocking
 *
 * Extends the React Component Class and is the UI part of the
Ridgewood Junction Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

 // Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/ridgewood_jct.css';

// Import Images
// Switch Images
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
```

```
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Ridgewood Junction
Interlocking
 *
 * This class is a JSX React Component for the Ridgewood Junction
Interlocking, this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
```

```
or if the route is occupied
 */
class RidgewoodJunction extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        sw_5: this.props.status.sw_5,
        sw_7: this.props.status.sw_7,
        sw_9: this.props.status.sw_9,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_135,
        sw_3_src: CX_135,
        sw_5_src: CX_225,
        sw_7_src: CX_225,
        sw_9_src: SW_U_E,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_6w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        sig_6e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_3_w: Empty,
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_1_center: Empty,
        tail_3_center: Empty,
        tail_m_1_e: Empty,
        tail_m_2_e: Empty,
        tail_b_1_e: Empty,
        tial_b_2_e: Empty,
        // Information For Interlocking Routes
        routes: this.props.status.routes,
        routed_1: this.props.status.routed_trk_1,
        routed_2: this.props.status.routed_trk_2,
```

```
        routed_3: this.props.status.routed_trk_3,
        occupied_track_1: this.props.status.occupied_trk_1,
        occupied_track_2: this.props.status.occupied_trk_2,
        occupied_track_3: this.props.status.occupied_trk_3
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            sw_5: nextProps.status.sw_5,
            sw_7: nextProps.status.sw_7,
            sw_9: nextProps.status.sw_9,

            routed_1: nextProps.status.routed_trk_1,
            routed_2: nextProps.status.routed_trk_2,
            routed_3: nextProps.status.routed_trk_3,
            occupied_track_1: nextProps.status.occupied_trk_1,
            occupied_track_2: nextProps.status.occupied_trk_2,
            occupied_track_3: nextProps.status.occupied_trk_3,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
```

```
        return (
            <div>
                {/* Tags */}
                <div className="ridgewood_title">RIDGEWOOD JUNCTION</
div>
                <div className="ridgewood_milepost">MP 20.3</div>
                {/* West Side Tail Tracks */}
                <div className="m_ridgewood_3_west"
style={{background: this.state.tail_3_w}}></div>
                <div className="m_ridgewood_1_west"
style={{background: this.state.tail_1_w}}></div>
                <div className="m_ridgewood_2_west"
style={{background: this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="ridgewood_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="ridgewood_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="ridgewood_5"
onClick={this.props.throw_sw_5}><img src={this.state.sw_5_src}/></div>
                <div className="ridgewood_7"
onClick={this.props.throw_sw_7}><img src={this.state.sw_7_src}/></div>
                <div className="ridgewood_9"
onClick={this.props.throw_sw_9}><img src={this.state.sw_9_src}/></div>
                {/* Center Tail Tracks */}
                <div className="m_ridgewood_3_center"
style={{background: this.state.tail_3_center}}></div>
                <div className="m_ridgewood_1_center"
style={{background: this.state.tail_1_center}}></div>
                {/* East Side Tail Tracks */}
                <div className="m_ridgewood_1_east"
style={{background: this.state.tail_m_1_e}}></div>
                <div className="m_ridgewood_2_east"
style={{background: this.state.tail_m_2_e}}></div>
                <div className="b_ridgewood_1_Diag"
style={{background: this.state.tail_b_1_e}}></div>
                <div className="b_ridgewood_1" style={{background:
this.state.tail_b_1_e}}></div>
                <div className="b_ridgewood_2" style={{background:
this.state.tail_b_2_e}}></div>
                {/* Signals */}
                <div className="ridgewood_sig_6w"
onClick={this.props.click_sig_6w}><img src={this.state.sig_6w_src}/></
div>
                <div className="ridgewood_sig_2w-2"
onClick={this.props.click_sig_2w_2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="ridgewood_sig_2w-1"
onClick={this.props.click_sig_2w_1}><img src={this.state.sig_2w1_src}/
></div>
```

```
                <div className="ridgewood_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="ridgewood_sig_6e"
onClick={this.props.click_sig_6e}><img src={this.state.sig_6e_src}/></
div>
                <div className="ridgewood_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="ridgewood_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawing() {
        let color_1 = Empty;
        let color_2 = Empty;
        let color_3 = Empty;
        if (this.state.routed_1) {
            color_1 = Green;
        }
        if (this.state.routed_2) {
            color_2 = Green;
        }
        if (this.state.routed_3) {
            color_3 = Green;
        }
        if (this.state.occupied_track_1) {
            color_1 = Red;
        }
        if (this.state.occupied_track_2) {
            color_2 = Red;
        }
        if (this.state.occupied_track_3) {
            color_3 = Red;
        }

        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_wc_ridgewood" ||
```

```
this.state.routes[i] === "E_1_1__|__1_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_m_1_e = color_1;
                this.state.tail_1_center = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_track_1) {
                    // Switch Images
                    this.state.sw_9_src = SW_U_E_Occupied;

                    if (this.state.routed_3) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                        this.state.sw_7_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                    }

                    if (this.state.routed_2) {
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_3_src = CX_135_Occupied_Both;
                        this.state.sw_5_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Top;
                        this.state.sw_5_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                else {
                    // Switch Images
                    this.state.sw_9_src = SW_U_E_Lined;

                    if (this.state.routed_3) {
```

```
                        this.state.sw_1_src = CX_135_Lined_Both;
                        this.state.sw_7_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }

                    if (this.state.routed_2) {
                        this.state.sw_3_src = CX_135_Lined_Both;
                        this.state.sw_5_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_3_src = CX_135_Lined_Top;
                        this.state.sw_5_src = CX_225_Lined_Top;
                    }

                    if (this.state.routes[i] === "W_1_1__|
__1_wc_ridgewood") {
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_4_1__|
__1_wc_ridgewood" || this.state.routes[i] === "E_1_4__|
__2_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_b_2_e = color_1;
                this.state.tail_1_center = color_1;
                this.state.tail_1_w = color_1;
```

```
                if (this.state.occupied_track_1) {
                    // Switch Images
                    this.state.sw_9_src = SW_U_E_R_Occupied;

                    if (this.state.routed_3) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                        this.state.sw_7_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                    }

                    if (this.state.routed_2) {
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_3_src = CX_135_Occupied_Both;
                        this.state.sw_5_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Top;
                        this.state.sw_5_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                else {
                    // Switch Images
                    this.state.sw_9_src = SW_U_E_R_Lined;

                    if (this.state.routed_3) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                        this.state.sw_7_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
```

```
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }

                    if (this.state.routed_2) {
                        this.state.sw_3_src = CX_135_Lined_Both;
                        this.state.sw_5_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_3_src = CX_135_Lined_Top;
                        this.state.sw_5_src = CX_225_Lined_Top;
                    }

                    if (this.state.routes[i] === "W_4_1__|
__1_wc_ridgewood") {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|
__2_wc_ridgewood" || this.state.routes[i] === "E_2_2__|
__2_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_m_2_e = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_track_2) {
                    if (this.state.routed_1) {
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
```

```
                    else if (this.state.occupied_track_1) {
                        this.state.sw_3_src = CX_135_Occupied_Both;
                        this.state.sw_5_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                        this.state.sw_5_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    if (this.state.routed_1) {
                        this.state.sw_3_src = CX_135_Lined_Both;
                        this.state.sw_5_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_1) {
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                        this.state.sw_5_src = CX_225_Lined_Bottom;
                    }

                    if (this.state.routes[i] === "W_2_2__|
__2_wc_ridgewood") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_3__|
__3_wc_ridgewood" || this.state.routes[i] === "E_3_3__|
__1_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_b_1_e = color_3;
                this.state.tail_3_center = color_3;
                this.state.tail_3_w = color_3;

                if (this.state.occupied_track_3) {
                    if (this.state.routes.includes("W_2_1__|
```

```
__1_wc_ridgewood")) {
                        if (this.state.routed_2) {
                            this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                        }
                        else {
                            this.state.sw_1_src =
CX_135_Occupied_Both;
                        }
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Top;
                    }
                    this.state.sw_7_src = CX_225_Occupied_Top;

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    if (this.state.routes.includes("W_2_1__|
__1_wc_ridgewood")) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Top;
                    }
                    this.state.sw_7_src = CX_225_Lined_Top;

                    if (this.state.routes[i] === "W_3_3__|
__3_wc_ridgewood") {
                        this.state.sig_6w_src = SIG_W_Clear;
                        this.state.sig_6e_src = SIG_E_Stop;
                    }
                    else {
                        this.state.sig_6w_src = SIG_W_Stop;
                        this.state.sig_6e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_m_1_e = color_1;
                this.state.tail_1_center = color_1;
                this.state.tail_2_w = color_1;

                if (this.state.occupied_track_1) {

                }
```

```
                else {
                    if (this.state.routed_3) {
                        this.state.sw_7_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }

                    this.state.sw_9_src = SW_U_E_Lined;
                    this.state.sw_5_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Clear;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|
__1_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_m_1_e = color_2;
                this.state.tail_1_center = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_track_2) {
                    if (this.state.routed_3) {
                        this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                    }

                    this.state.sw_9_src = SW_U_E_Occupied;
                    this.state.sw_5_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
```

```
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    if (this.state.routed_3) {
                        this.state.sw_7_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }

                    this.state.sw_9_src = SW_U_E_Lined;
                    this.state.sw_5_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "E_2_4__|
__2_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_b_2_e = color_2;
                this.state.tail_1_center = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_track_2) {
                    if (this.state.routed_3) {
                        this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                    }

                    this.state.sw_9_src = SW_U_E_R_Occupied;
                    this.state.sw_5_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;
```

```
                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    if (this.state.routed_3) {
                        this.state.sw_7_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }

                    this.state.sw_9_src = SW_U_E_R_Lined;
                    this.state.sw_5_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_4_2__|
__2_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_b_2_e = color_1;
                this.state.tail_1_center = color_1;
                this.state.tail_2_w = color_1;

                if (this.state.occupied_track_1) {
                    if (this.state.routed_3) {
                        this.state.sw_7_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src = CX_225_Occupied_Both;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Occupied_Bottom;
                    }
```

```
                    this.state.sw_9_src = SW_U_E_R_Occupied;
                    this.state.sw_5_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    if (this.state.routed_3) {
                        this.state.sw_7_src = CX_225_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_7_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_7_src = CX_225_Lined_Bottom;
                    }

                    this.state.sw_9_src = SW_U_E_R_Lined;
                    this.state.sw_5_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_1_3__|
__3_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_m_1_e = color_1;
                this.state.tail_1_center = color_1;
                this.state.tail_3_w = color_1;

                if (this.state.occupied_track_1) {
                    this.state.sw_9_src = SW_U_E_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    if (this.state.routed_2) {
                        this.state.sw_5_src =
```

```
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    this.state.sw_9_src = SW_U_E_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    if (this.state.routed_2) {
                        this.state.sw_5_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Clear;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_1__|
__1_ridgewood_suscon") {
```

```
                // Tail Tracks
                this.state.tail_m_1_e = color_3;
                this.state.tail_1_center = color_3;
                this.state.tail_3_w = color_3;

                if (this.state.occupied_track_3) {
                    this.state.sw_9_src = SW_U_E_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    if (this.state.routed_2) {
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    this.state.sw_9_src = SW_U_E_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    if (this.state.routed_2) {
                        this.state.sw_5_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
```

```
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_3_1__|
__1_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_b_1_e = color_3;
                this.state.tail_1_w = color_3;

                if (this.state.occupied_track_3) {
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    if (this.state.routed_2) {
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    if (this.state.routed_2) {
                        this.state.sw_5_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
```

```
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_6w_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_3__|
__1_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_b_1_e = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_track_1) {
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_1_src = CX_135_Occupied_Bottom;

                    if (this.state.routed_2) {
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
```

```
                }
                else {
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_1_src = CX_135_Lined_Bottom;

                    if (this.state.routed_2) {
                        this.state.sw_5_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_3_2__|
__2_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_b_1_e = color_3;
                this.state.tail_2_w = color_3;

                if (this.state.occupied_track_3) {
                    // Switches
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_5_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
```

```
                    // Switches
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_5_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_6w_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_3__|
__1_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_b_1_e = color_2;
                this.state.tail_2_w = color_2;

                if (this.state.occupied_track_2) {
                    // Switches
                    this.state.sw_7_src = CX_225_R_Occupied;
                    this.state.sw_5_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_7_src = CX_225_R_Lined;
                    this.state.sw_5_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signals
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                    this.state.sig_6e_src = SIG_E_Stop;
```

```
                }
            }
            else if (this.state.routes[i] === "W_4_3__|
__3_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_b_2_e = color_1;
                this.state.tail_1_center = color_1;
                this.state.tail_3_w = color_1;

                if (this.state.occupied_track_1) {
                    // Switches
                    this.state.sw_9_src = SW_U_E_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    if (this.state.routed_2) {
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_9_src = SW_U_E_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    if (this.state.routed_2) {
                        this.state.sw_5_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
```

```
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_4__|
__2_ridgewood_bt") {
                // Tail Tracks
                this.state.tail_b_2_e = color_3;
                this.state.tail_1_center = color_3;
                this.state.tail_3_w = color_3;

                if (this.state.occupied_track_3) {
                    // Switches
                    this.state.sw_9_src = SW_U_E_R_Occupied;
                    this.state.sw_7_src = CX_225_Occupied_Bottom;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    if (this.state.routed_2) {
                        this.state.sw_5_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
```

```
                }
                else {
                    // Switches
                    this.state.sw_9_src = SW_U_E_R_Lined;
                    this.state.sw_7_src = CX_225_Lined_Bottom;
                    this.state.sw_1_src = CX_135_R_Lined;

                    if (this.state.routed_2) {
                        this.state.sw_5_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_2) {
                        this.state.sw_5_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else {
                        this.state.sw_5_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_2_3__|
__3_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_m_2_e = color_2;
                this.state.tail_3_w = color_2;

                if (this.state.occupied_track_2) {
                    this.state.sw_5_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
```

```
                else {
                    this.state.sw_5_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_2__|
__2_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_m_2_e = color_3;
                this.state.tail_3_w = color_3;

                if (this.state.occupied_track_2) {
                    this.state.sw_5_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;
                    this.state.sw_1_src = CX_135_R_Occupied;

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Stop;
                }
                else {
                    this.state.sw_5_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;
                    this.state.sw_1_src = CX_135_R_Lined;

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_6w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_6e_src = SIG_E_Clear;
                }
            }
```

```
            else if (this.state.routes[i] === "W_2_1__|
__1_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_m_2_e = color_2;
                this.state.tail_1_w = color_2;

                if (this.state.occupied_track_2) {
                    this.state.sw_5_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    if (this.state.routed_3) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                else {
                    this.state.sw_5_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    if (this.state.routed_3) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
```

```
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_m_2_e = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_track_1) {
                    this.state.sw_5_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    if (this.state.routed_3) {
                        this.state.sw_1_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src = CX_135_Occupied_Both;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                else {
                    this.state.sw_5_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    if (this.state.routed_3) {
                        this.state.sw_1_src = CX_135_Lined_Both;
                    }
                    else if (this.state.occupied_track_3) {
                        this.state.sw_1_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    else {
                        this.state.sw_1_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
```

```
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_135_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_135;
        }

        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_135_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_3_src = CX_135;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_5) {
            this.state.sw_5_src = CX_225_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_5_src = CX_225;
        }

        // Set SW #5
        // SW #5 Reversed
```

```
        if (this.state.sw_7) {
            this.state.sw_7_src = CX_225_R;
        }
        // SW #5 Normal
        else {
            this.state.sw_7_src = CX_225;
        }

        // Set SW #7
        // SW #7 Reversed
        if (this.state.sw_9) {
            this.state.sw_9_src = SW_U_E_R;
        }
        // SW #7 Normal
        else {
            this.state.sw_9_src = SW_U_E;
        }
    }
    // ---- END set_switch_image() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_center = Empty;
        this.state.tail_3_center = Empty;
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_3_w = Empty;
        this.state.tail_m_1_e = Empty;
        this.state.tail_m_2_e = Empty;
        this.state.tail_b_2_e = Empty;
        this.state.tail_b_1_e = Empty;

        this.state.sig_6w_src = SIG_W;
        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_6e_src = SIG_E;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
```

```
}

// Export the interlocking to be drawn on the screen
export default RidgewoodJunction;
```

```
/**
 * @file Hilburn.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for SF Interlocking
 *
 * Extends the React Component Class and is the UI part of the SF
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/sf.css';

// Import Images
// Switch Images
import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

import SW_D_W from '../../../../public/images/SW_D_W.png';
import SW_D_W_Lined from '../../../../public/images/SW_D_W_Lined.png';
import SW_D_W_Occupied from '../../../../public/images/
SW_D_W_Occupied.png';
import SW_D_W_R from '../../../../public/images/SW_D_W_R.png';
import SW_D_W_R_Lined from '../../../../public/images/
SW_D_W_R_Lined.png';
```

```
import SW_D_W_R_Occupied from '../../../../public/images/
SW_D_W_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the SF Interlocking
 *
 * This class is a JSX React Component for the SF Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class SF extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_D_W,
        sw_3_src: CX_225,
        // Image File for the signals - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
```

```
        tail_yard: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        // Colors for tail tracks - Will change depending on route
        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e1_src: SIG_E,
        sig_4e2_src: SIG_E,
        // Information For Interlocking Routes
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_trk_1,
        route_2: this.props.status.routed_trk_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_trk_1,
            route_2: nextProps.status.routed_trk_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
```

```
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="sf_title">SF</div>
                <div className="sf_milepost">MP 30.5</div>
                {/* West Side Tail Tracks */}
                <div className="sf_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="sf_2_west" style={{background:
this.state.tail_2_w}}></div>
                <div className="sf_yard" style={{background:
this.state.tail_yard}}></div>
                {/* Switches */}
                <div className="sf_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="sf_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="sf_1_center_west" style={{background:
this.state.tail_1_e}}></div>
                <div className="sf_2_center_west" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="sf_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="sf_sig_4e-1"
onClick={this.props.click_sig_4e_1}><img src={this.state.sig_4e1_src}/
></div>
                <div className="sf_sig_4e-2"
onClick={this.props.click_sig_4e_2}><img src={this.state.sig_4e2_src}/
></div>
                <div className="sf_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="sf_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
```

```
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of tracks depending on if interlocking is
occupied or not
        let color_1 = Empty;
        let color_2 = Empty;
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_sterling_sf" ||
this.state.routes[i] === "E_1_1__|__1_sf_wc") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    // Track #2 Is Routed
                    if (this.state.route_2) {
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Track #2 Is Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Occupied_Top;
                    }

                    // Signals
```

```
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Track #2 Is Routed
                    if (this.state.route_2) {
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    // Track #2 Is Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_sterling_sf") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|__2_hilburn_sf"
|| this.state.routes[i] === "E_2_2__|__2_sf_wc") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_1_src = SW_D_W_Occupied;

                    // Crossovers that could change depending on Track
#2
                    // Track #2 Is Routed
                    if (this.state.route_1) {
                        this.state.sw_3_src =
```

```
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Track #2 Is Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e1_src = SIG_E_Stop;
                    this.state.sig_4e2_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_1_src = SW_D_W_Lined;

                    // Crossovers that could change depending on Track
#2
                    // Track #2 Is Routed
                    if (this.state.route_1) {
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    // Track #2 Is Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_hilburn_sf") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e1_src = SIG_E_Stop;
                        this.state.sig_4e2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e1_src = SIG_E_Clear;
                        this.state.sig_4e2_src = SIG_E_Stop;
                    }
```

```
                }
            }
            else if (this.state.routes[i] === "W_2_3__|
__3_yardHilburn_sf" || this.state.routes[i] === "E_3_2__|__2_sf_wc") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_yard = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_1_src = SW_D_W_R_Occupied;

                    // Crossovers that could change based off of Track
#1
                    // Track #1 Routed
                    if (this.state.route_1) {
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Track #1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_3_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e1_src = SIG_E_Stop;
                    this.state.sig_4e2_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_D_W_R_Lined;

                    // Crossovers that could change based off of Track
#1
                    // Track #1 Routed
                    if (this.state.route_1) {
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    // Track #1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
```

```
                    // Nothing Track #1
                    else {
                        this.state.sw_3_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_3__|
__3_yardHilburn_sf") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e1_src = SIG_E_Stop;
                        this.state.sig_4e2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e1_src = SIG_E_Stop;
                        this.state.sig_4e2_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_hilburn_sf") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_D_W_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e1_src = SIG_E_Stop;
                    this.state.sig_4e2_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = SW_D_W_Lined;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Clear;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
```

```
                    this.state.sig_4e1_src = SIG_E_Stop;
                    this.state.sig_4e2_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|__1_sf_wc") {
                // Tail Tracks
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_D_W_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e1_src = SIG_E_Stop;
                    this.state.sig_4e2_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = SW_D_W_Lined;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e1_src = SIG_E_Clear;
                    this.state.sig_4e2_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_1_3__|
__2_yardHilburn_sf") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_yard = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_D_W_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
```

```
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_4e1_src = SIG_E_Stop;
            this.state.sig_4e2_src = SIG_E_Stop;
        }
        // The Route Is NOT Occupied
        else {
            // Switches
            this.state.sw_3_src = CX_225_R_Lined;
            this.state.sw_1_src = SW_D_W_R_Lined;

            // Signals
            this.state.sig_2w_src = SIG_W_Clear;
            this.state.sig_2e_src = SIG_E_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_4e1_src = SIG_E_Stop;
            this.state.sig_4e2_src = SIG_E_Stop;
        }
    }
    else if (this.state.routes[i] === "E_3_1__|__1_sf_wc") {
        // Tail Tracks
        this.state.tail_1_e = color_2;
        this.state.tail_yard = color_2;

        // The Route Is Occupied
        if (this.state.occupied_1) {
            // Switches
            this.state.sw_3_src = CX_225_R_Occupied;
            this.state.sw_1_src = SW_D_W_R_Occupied;

            // Signals
            this.state.sig_2w_src = SIG_W_Stop;
            this.state.sig_2e_src = SIG_E_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_4e1_src = SIG_E_Stop;
            this.state.sig_4e2_src = SIG_E_Stop;
        }
        // The Route Is NOT Occupied
        else {
            // Switches
            this.state.sw_3_src = CX_225_R_Lined;
            this.state.sw_1_src = SW_D_W_R_Lined;

            // Signals
            this.state.sig_2w_src = SIG_W_Stop;
            this.state.sig_2e_src = SIG_E_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_4e1_src = SIG_E_Stop;
            this.state.sig_4e2_src = SIG_E_Clear;
        }
    }
```

```
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_D_W_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_D_W;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_225_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = CX_225;
        }
    }
    // ---- END set_switch_image() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
```

```
        this.state.tail_2_w = Empty;
        this.state.tail_yard = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;

        this.state.sig_2w_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e1_src = SIG_E;
        this.state.sig_4e2_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default SF;
```

```
/**
 * @file Mill.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Mill Interlocking
 *
 * Extends the React Component Class and is the UI part of the Mill
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/suscon.css';

// Import Images
// Switch Images
// Images for a 135 Crossover
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

// Images for a 225 Crossover
import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
```

```
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Suscon Interlocking
 *
 * This class is a JSX React Component for the Suscon Interlocking,
this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Suscon extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
```

```
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        sw_1_src: CX_225,
        sw_3_src: CX_135,

        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,

        tail_1_e: Empty,
        tail_1_w: Empty,
        tail_2_e: Empty,
        tail_2_w: Empty,

        occupied_trk_1: this.props.status.occupied_trk_1,
        occupied_trk_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_trk_1,
        route_2: this.props.status.routed_trk_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            occupied_trk_1: nextProps.status.occupied_trk_1,
            occupied_trk_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_trk_1,
            route_2: nextProps.status.routed_trk_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
```

```
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="suscon_title">SUSCON</div>
                <div className="suscon_milepost">MP 17.5</div>

                {/* West Side Tracks */}
                <div className="suscon_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="suscon_2_west" style={{background:
this.state.tail_2_w}}></div>

                {/* Switches */}
                <div className="suscon_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="suscon_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>

                {/* East Side Tracks */}
                <div className="suscon_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="suscon_2_east" style={{background:
this.state.tail_2_e}}></div>

                {/* Signals */}
                <div className="suscon_sig_2w"
onClick={this.props.click_sig_2w} id="suscon_2w"><img
id="suscon_2w_image" src={this.state.sig_2w_src}/></div>
                <div className="suscon_sig_4w"
onClick={this.props.click_sig_4w} id="suscon_4w"><img
id="suscon_4w_image" src={this.state.sig_4w_src}/></div>
                <div className="suscon_sig_2e"
onClick={this.props.click_sig_2e} id="suscon_2e"><img
id="suscon_2e_image" src={this.state.sig_2e_src}/></div>
                <div className="suscon_sig_4e"
```

```
onClick={this.props.click_sig_4e} id="suscon_4e"><img
id="suscon_4e_image" src={this.state.sig_4e_src}/></div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * set_route_drawings()
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or
not.
     */
    set_route_drawing() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Set Track Colors
        // If each track has a route
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        // If each track is occupied
        if (this.state.occupied_trk_1) {
            color_1 = Red;
        }
        if (this.state.occupied_trk_2) {
            color_2 = Red;
        }

        // Loop through all the Routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|
__1_ridgewood_suscon" || this.state.routes[i] === "E_1_1__|
__1_suscon_mill") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_trk_1) {
                    // Routed Track #2
                    if (this.state.route_2) {
```

```
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    // Occupied Track #2
                    else if (this.state.occupied_trk_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                        this.state.sw_3_src = CX_135_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Routed Track #2
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    // Occupied Track #2
                    else if (this.state.occupied_trk_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                        this.state.sw_3_src = CX_135_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_ridgewood_suscon") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
```

```
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|
__2_ridgewood_suscon" || this.state.routes[i] === "E_2_2__|
__2_suscon_mill") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_2_w = color_2;

                // If The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Routed Track #1
                    if (this.state.route_1) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        this.state.sw_3_src =
CX_135_Lined_Top_Occupied_Bottom;
                    }
                    // Occupied Track #1
                    else if (this.state.occupied_trk_1) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                        this.state.sw_3_src = CX_135_Occupied_Both;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Bottom;
                        this.state.sw_3_src = CX_135_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w = SIG_W_Stop;
                    this.state.sig_4e = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Routed Track #1
                    if (this.state.route_1) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                        this.state.sw_3_src = CX_135_Lined_Both;
                    }
                    // Occupied Track #1
                    else if (this.state.occupied_trk_1) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        this.state.sw_3_src =
CX_135_Occupied_Top_Lined_Bottom;
                    }
                    // Nothing Track #1
```

```
                    else {
                        this.state.sw_1_src = CX_225_Lined_Bottom;
                        this.state.sw_3_src = CX_135_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_ridgewood_suscon") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_trk_1) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
```

```
            else if (this.state.routes[i] === "E_2_1__|
__1_suscon_mill") {
                // Tail Tracks
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Occupied;
                    this.state.sw_3_src = CX_135_Occupied_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_R_Lined;
                    this.state.sw_3_src = CX_135_Lined_Bottom;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__1_ridgewood_suscon") {
                // Tail Tracks
                this.state.tail_2_e = color_2;
                this.state.tail_1_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
```

```
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_suscon_mill") {
                // Tail Tracks
                this.state.tail_2_e = color_1;
                this.state.tail_1_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_trk_2) {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Occupied_Bottom;
                    this.state.sw_3_src = CX_135_R_Occupied;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_1_src = CX_225_Lined_Bottom;
                    this.state.sw_3_src = CX_135_R_Lined;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
```

```
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_225_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_225;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_135_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = CX_135;
        }
    }
    // ---- END set_switch_image() ----


    /**
     * reset_drawings()
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_e = Empty;
        this.state.tail_1_w = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_2_w = Empty;

        this.state.sig_2e_src = SIG_E;
```

```
        this.state.sig_2w_src = SIG_W;
        this.state.sig_4e_src = SIG_E;
        this.state.sig_4w_src = SIG_W;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Suscon;
```

```
/**
 * @file Hilburn.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Hilburn Interlocking
 *
 * Extends the React Component Class and is the UI part of the Hilburn
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/wc.css';

// Import Images
// Switch Images
import CX_135 from '../../../../public/images/CX_135.png';
import CX_135_Lined_Top from '../../../../public/images/
CX_135_Lined_Top.png';
import CX_135_Lined_Bottom from '../../../../public/images/
CX_135_Lined_Bottom.png';
import CX_135_Lined_Both from '../../../../public/images/
CX_135_Lined_Both.png';
import CX_135_R from '../../../../public/images/CX_135_R.png';
import CX_135_R_Lined from '../../../../public/images/
CX_135_R_Lined.png';
import CX_135_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_135_Lined_Top_Occupied_Bottom.png';
import CX_135_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_135_Occupied_Top_Lined_Bottom.png';
import CX_135_Occupied_Top from '../../../../public/images/
CX_135_Occupied_Top.png';
import CX_135_Occupied_Bottom from '../../../../public/images/
CX_135_Occupied_Bottom.png';
import CX_135_Occupied_Both from '../../../../public/images/
CX_135_Occupied_Both.png';
import CX_135_R_Occupied from '../../../../public/images/
CX_135_R_Occupied.png';

import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
```

```
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

import SW_U_W from '../../../../public/images/SW_U_W.png';
import SW_U_W_Lined from '../../../../public/images/SW_U_W_Lined.png';
import SW_U_W_Occupied from '../../../../public/images/
SW_U_W_Occupied.png';
import SW_U_W_R from '../../../../public/images/SW_U_W_R.png';
import SW_U_W_R_Lined from '../../../../public/images/
SW_U_W_R_Lined.png';
import SW_U_W_R_Occupied from '../../../../public/images/
SW_U_W_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';
```

```
/**
 * The React JSX Component Class for the WC Interlocking
 *
 * This class is a JSX React Component for the WC Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class WC extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        sw_5: this.props.status.sw_5,
        sw_7: this.props.status.sw_7,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_225,
        sw_3_src: SW_U_W,
        sw_5_src: CX_135,
        sw_7_src: SW_U_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_yard: Empty,
        tail_2_center: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        tail_3_e: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e1_src: SIG_E,
        sig_2e2_src: SIG_E,
        sig_4e_src: SIG_E,
        // Information For Interlocking Routes
```

```
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_trk_1,
        route_2: this.props.status.routed_trk_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            sw_5: nextProps.status.sw_5,
            sw_7: nextProps.status.sw_7,
            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_trk_1,
            route_2: nextProps.status.routed_trk_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
```

```
                <div className="wc_title">WC</div>
                <div className="wc_milepost">MP 23.6</div>
                {/* West Side Tail Tracks */}
                <div className="wc_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="wc_2_west" style={{background:
this.state.tail_2_w}}></div>
                <div className="wc_yard" style={{background:
this.state.tail_yard}}></div>
                {/* Switches */}
                <div className="wc_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                <div className="wc_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="wc_SW_5"
onClick={this.props.throw_sw_5}><img src={this.state.sw_5_src}/></div>
                <div className="wc_SW_7"
onClick={this.props.throw_sw_7}><img src={this.state.sw_7_src}/></div>
                {/* Center Tail Tracks */}
                <div className="wc_2_center" style={{background:
this.state.tail_2_center}}></div>
                {/* East Side Tail Tracks */}
                <div className="wc_3_east" style={{background:
this.state.tail_3_e}}></div>
                <div className="wc_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="wc_2_east" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="wc_sig_2e-2"
onClick={this.props.click_sig_2e_2}><img src={this.state.sig_2e2_src}/
></div>
                <div className="wc_sig_2e-1"
onClick={this.props.click_sig_2e_1}><img src={this.state.sig_2e1_src}/
></div>
                <div className="wc_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
                <div className="wc_sig_2w-2"
onClick={this.props.click_sig_2w_2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="wc_sig_2w-1"
onClick={this.props.click_sig_2w_1}><img src={this.state.sig_2w1_src}/
></div>
                <div className="wc_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
            </div>
        );
    }
```

```
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }

        // Loop Through All The Routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_sf_wc" ||
this.state.routes[i] === "E_1_1__|__1_wc_ridgewood") {
                // Tail Tracks
                this.state.tail_1_e = color_1;
                this.state.tail_1_w = color_1;

                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_7_src = SW_U_E_Occupied;
                    this.state.sw_3_src = SW_U_W_Occupied;

                    // Crossovers that could change based off of Track
#2 Status
                    if (this.state.routes.includes("W_2_2__|
__2_sf_wc") || this.state.routes.includes("E_2_2__|__2_wc_ridgewood"))
{
                        // Track #2 Is Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_5_src =
```

```
CX_135_Occupied_Both;
                            this.state.sw_1_src =
CX_225_Occupied_Bottom;
                        }
                        // Track #2 Routed
                        else if (this.state.route_2) {
                            this.state.sw_5_src =
CX_135_Occupied_Top_Lined_Bottom;
                            this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                        }
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_5_src = CX_135_Occupied_Top;
                        this.state.sw_1_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                }
                else {
                    // Switches
                    this.state.sw_7_src = SW_U_E_Lined;
                    this.state.sw_3_src = SW_U_W_Lined;

                    // Crossovers that could change based off of Track
#2 Status
                    if (this.state.routes.includes("W_2_2__|
__2_sf_wc") || this.state.routes.includes("E_2_2__|__2_wc_ridgewood"))
{
                        // Track #2 Occupied
                        if (this.state.occupied_2) {
                            this.state.sw_5_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                        }
                        // Track #2 Routed
                        else if (this.state.route_2) {
                            this.state.sw_5_src = CX_135_Lined_Both;
                            this.state.sw_1_src = CX_225_Lined_Both;
                        }
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_5_src = CX_135_Lined_Top;
```

```
                        this.state.sw_1_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|__1_sf_wc")
{
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e1_src = SIG_E_Stop;
                        this.state.sig_2e2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e1_src = SIG_E_Clear;
                        this.state.sig_2e2_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|__2_sf_wc" ||
this.state.routes[i] === "E_2_2__|__2_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_2_e = color_2;
                this.state.tail_2_center = color_2;
                this.state.tail_2_w = color_2;

                // If The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    // Crossovers that could change based of the state
of Track #1
                    if (this.state.routes.includes("W_1_1__|
__1_sf_wc") || this.state.routes.includes("E_1_1__|__1_wc_ridgewood")
||
                        this.state.routes.includes("W_3_1__|
__1_sf_wc") || this.state.routes.includes("E_1_3__|__3_wc_ridgewood"))
{
                        if (this.state.occupied_1) {
                            this.state.sw_5_src =
CX_135_Occupied_Both;
                            this.state.sw_1_src =
CX_225_Occupied_Both;
                        }
                        else if (this.state.route_1) {
                            this.state.sw_5_src =
CX_135_Lined_Top_Occupied_Bottom;
                            this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
```

```
                    }
                }
                else if (this.state.routes.includes("W_3_3__|
__0_yard_wc") || this.state.routes.includes("E_3_3__|
__3_wc_ridgewood") ||
                    this.state.routes.includes("W_1_3__|
__0_yard_wc") || this.state.routes.includes("E_3_1__|
__1_wc_ridgewood")) {
                    if (this.state.occupied_1) {
                        this.state.sw_5_src =
CX_135_Occupied_Both;
                        this.state.sw_1_src =
CX_225_Occupied_Bottom;
                    }
                    else if (this.state.route_1) {
                        this.state.sw_5_src =
CX_135_Lined_Top_Occupied_Bottom;
                        this.state.sw_1_src =
CX_225_Occupied_Bottom;
                    }
                }
                // Nothing Track #1
                else {
                    this.state.sw_5_src = CX_135_Occupied_Bottom;
                    this.state.sw_1_src = CX_225_Occupied_Bottom;
                }

                // Signals
                this.state.sig_4w_src = SIG_W_Stop;
                this.state.sig_4e_src = SIG_E_Stop;
            }
            // The Route Is NOT Occupied
            else {
                // Switches
                // Crossovers that could change based of the state
of Track #1
                if (this.state.routes.includes("W_1_1__|
__1_sf_wc") || this.state.routes.includes("E_1_1__|__1_wc_ridgewood")
||
                    this.state.routes.includes("W_3_1__|
__1_sf_wc") || this.state.routes.includes("E_1_3__|__3_wc_ridgewood"))
{
                    if (this.state.occupied_1) {
                        this.state.sw_5_src =
CX_135_Occupied_Top_Lined_Bottom;
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    else if (this.state.route_1) {
                        this.state.sw_5_src = CX_135_Lined_Both;
```

```
                            this.state.sw_1_src = CX_225_Lined_Both;
                        }
                    }
                    else if (this.state.routes.includes("W_3_3__|
__0_yard_wc") || this.state.routes.includes("E_3_3__|
__3_wc_ridgewood") ||
                            this.state.routes.includes("W_1_3__|
__0_yard_wc") || this.state.routes.includes("E_3_1__|
__1_wc_ridgewood")) {
                            if (this.state.occupied_1) {
                                this.state.sw_5_src =
CX_135_Occupied_Top_Lined_Bottom;
                                this.state.sw_1_src = CX_225_Lined_Bottom;
                            }
                            else if (this.state.route_1) {
                                this.state.sw_5_src = CX_135_Lined_Both;
                                this.state.sw_1_src = CX_225_Lined_Bottom;
                            }
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_5_src = CX_135_Lined_Bottom;
                        this.state.sw_1_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|__2_sf_wc")
{
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_1__|__1_sf_wc" ||
this.state.routes[i] === "E_1_3__|__3_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_1;
                this.state.tail_1_w = color_1;

                // If The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Occupied;
                    this.state.sw_5_src = CX_135_Occupied_Top;
```

```
                    this.state.sw_3_src = SW_U_W_Occupied;
                    this.state.sw_1_src = CX_225_Occupied_Top;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Lined;
                    this.state.sw_5_src = CX_135_Lined_Top;
                    this.state.sw_3_src = SW_U_W_Lined;
                    this.state.sw_1_src = CX_225_Lined_Top;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_1__|__1_sf_wc")
{
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2e1_src = SIG_E_Stop;
                        this.state.sig_2e2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2e1_src = SIG_E_Clear;
                        this.state.sig_2e2_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_3__|__0_yard_wc" ||
this.state.routes[i] === "E_3_3__|__3_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_1;
                this.state.tail_yard = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Occupied;
                    this.state.sw_5_src = CX_135_Occupied_Top;
                    this.state.sw_3_src = SW_U_W_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
```

```
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Lined;
                    this.state.sw_5_src = CX_135_Lined_Top;
                    this.state.sw_3_src = SW_U_W_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_3__|
__0_yard_wc") {
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2e1_src = SIG_E_Stop;
                        this.state.sig_2e2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2e1_src = SIG_E_Stop;
                        this.state.sig_2e2_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_2__|__2_sf_wc") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Occupied;
                    this.state.sw_5_src = CX_135_Occupied_Top;
                    this.state.sw_3_src = SW_U_W_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
```

```
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Lined;
                    this.state.sw_5_src = CX_135_Lined_Top;
                    this.state.sw_3_src = SW_U_W_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Clear;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_3__|
__3_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_3_e = color_2;
                this.state.tail_2_w = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Occupied;
                    this.state.sw_5_src = CX_135_Occupied_Top;
                    this.state.sw_3_src = SW_U_W_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_7_src = SW_U_E_R_Lined;
                    this.state.sw_5_src = CX_135_Lined_Top;
                    this.state.sw_3_src = SW_U_W_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
```

```
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_1_2__|__2_sf_wc") {
                // Set Tail Track Colors
                this.state.tail_1_e = color_1;
                this.state.tail_2_w = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_7_src = SW_U_E_Occupied;
                    this.state.sw_5_src = CX_135_Occupied_Top;
                    this.state.sw_3_src = SW_U_W_Occupied;
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_7_src = SW_U_E_Lined;
                    this.state.sw_5_src = CX_135_Lined_Top;
                    this.state.sw_3_src = SW_U_W_Lined;
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_1__|
__1_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_1_e = color_2;
                this.state.tail_2_w = color_2;
```

```
        // The Route Is Occupied
        if (this.state.occupied_2) {
            // Switches
            this.state.sw_7_src = SW_U_E_Occupied;
            this.state.sw_5_src = CX_135_Occupied_Top;
            this.state.sw_3_src = SW_U_W_Occupied;
            this.state.sw_1_src = CX_225_R_Occupied;

            // Signals
            this.state.sig_2w2_src = SIG_W_Stop;
            this.state.sig_2w1_src = SIG_W_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e1_src = SIG_E_Stop;
            this.state.sig_2e2_src = SIG_E_Stop;
            this.state.sig_4e_src = SIG_E_Stop;
        }
        // The Route Is NOT Occupied
        else {
            // Switches
            this.state.sw_7_src = SW_U_E_Lined;
            this.state.sw_5_src = CX_135_Lined_Top;
            this.state.sw_3_src = SW_U_W_Lined;
            this.state.sw_1_src = CX_225_R_Lined;

            // Signals
            this.state.sig_2w2_src = SIG_W_Stop;
            this.state.sig_2w1_src = SIG_W_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e1_src = SIG_E_Stop;
            this.state.sig_2e2_src = SIG_E_Stop;
            this.state.sig_4e_src = SIG_E_Clear;
        }
    }
    else if (this.state.routes[i] === "W_2_1__|__1_sf_wc") {
        // Set Tail Track Colors
        this.state.tail_2_e = color_2;
        this.state.tail_1_w = color_2;

        // If The Route Is Occupied
        if (this.state.occupied_2) {
            // Switches
            this.state.sw_5_src = CX_135_R_Occupied;
            this.state.sw_3_src = SW_U_W_Occupied;
            this.state.sw_1_src = CX_225_Occupied_Top;

            // Signals
            this.state.sig_2w2_src = SIG_W_Stop;
            this.state.sig_2w1_src = SIG_W_Stop;
            this.state.sig_4w_src = SIG_W_Stop;
            this.state.sig_2e1_src = SIG_E_Stop;
```

```
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // If The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Lined;
                    this.state.sw_3_src = SW_U_W_Lined;
                    this.state.sw_1_src = CX_225_Lined_Top;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_2_e = color_1;
                this.state.tail_1_w = color_1;

                // If The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Occupied;
                    this.state.sw_3_src = SW_U_W_Occupied;
                    this.state.sw_1_src = CX_225_Occupied_Top;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // If The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Lined;
                    this.state.sw_3_src = SW_U_W_Lined;
                    this.state.sw_1_src = CX_225_Lined_Top;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
```

```
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Clear;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "W_2_3__|__0_yard_wc") {
                // Set Tail Track Colors
                this.state.tail_2_e = color_2;
                this.state.tail_yard = color_2;

                // If The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Occupied;
                    this.state.sw_3_src = SW_U_W_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Lined;
                    this.state.sw_3_src = SW_U_W_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_3_2__|
__2_wc_ridgewood") {
                // Set Tail Track Colors
                this.state.tail_2_e = color_1;
                this.state.tail_yard = color_1;

                // If The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Occupied;
```

```
                    this.state.sw_3_src = SW_U_W_R_Occupied;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_5_src = CX_135_R_Lined;
                    this.state.sw_3_src = SW_U_W_R_Lined;

                    // Signals
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e1_src = SIG_E_Stop;
                    this.state.sig_2e2_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_225_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = CX_225;
        }
```

```
        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = SW_U_W_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = SW_U_W;
        }

        // Set SW #5
        // SW #5 Reversed
        if (this.state.sw_5) {
            this.state.sw_5_src = CX_135_R;
        }
        // SW #5 Normal
        else {
            this.state.sw_5_src = CX_135;
        }

        // Set SW #7
        // SW #7 Reversed
        if (this.state.sw_7) {
            this.state.sw_7_src = SW_U_E_R;
        }
        // SW #7 Normal
        else {
            this.state.sw_7_src = SW_U_E;
        }
    }
    // ---- END set_switch_image() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_yard = Empty;
        this.state.tail_2_center = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
```

```
        this.state.tail_3_e = Empty;

        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e1_src = SIG_E;
        this.state.sig_2e2_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default WC;
```

```
/**
 * @file WestSecaucus.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for West Secaucus
Interlocking
 *
 * Extends the React Component Class and is the UI part of the West
Secaucus Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Main_Line/west_secaucus.css';

// Import Images
// Switch Images
import SW_D_E from '../../../../public/images/SW_D_E.png';
import SW_D_E_Lined from '../../../../public/images/SW_D_E_Lined.png';
import SW_D_E_Occupied from '../../../../public/images/
SW_D_E_Occupied.png'
import SW_D_E_R from '../../../../public/images/SW_D_E_R.png';
import SW_D_E_R_Lined from '../../../../public/images/
SW_D_E_R_Lined.png';
import SW_D_E_R_Occupied from '../../../../public/images/
SW_D_E_R_Occupied.png';

import SW_D_W from '../../../../public/images/SW_D_W.png';
import SW_D_W_Lined from '../../../../public/images/SW_D_W_Lined.png';
import SW_D_W_Occupied from '../../../../public/images/
SW_D_W_Occupied.png'
import SW_D_W_R from '../../../../public/images/SW_D_W_R.png';
import SW_D_W_R_Lined from '../../../../public/images/
SW_D_W_R_Lined.png';
import SW_D_W_R_Occupied from '../../../../public/images/
SW_D_W_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
```

```
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';


/**
 * The React JSX Component Class for the West Secaucus Interlocking
 *
 * This class is a JSX React Component for the West Secaucus
Interlocking, this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class WestSecaucus extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_D_W,
        sw_3_src: SW_D_E,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_e: Empty,
        tail_1_w: Empty,
        tail_2_e: Empty,
        tail_2_w: Empty,
        // Information For Interlocking Routes
        routes: this.props.status.routes,
        occupied: this.props.status.occupied
    };
```

```
    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps) {
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            sig_2w_src: SIG_W,
            sig_4w_src: SIG_W,
            sig_2e_src: SIG_E,
            sig_4e_src: SIG_E,
            tail_1_e: Empty,
            tail_1_w: Empty,
            tail_center: Empty,
            tail_2_e: Empty,
            tail_2_w: Empty,
            routes: nextProps.status.routes,
            occupied: nextProps.status.occupied
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="westSecaucus_title">WEST SECAUCUS</
div>
                <div className="westSecaucus_milepost">MP 5.0</div>
                {/* East Side Tail Tracks */}
                <div className="m_westSecaucus_1_east"
style={{background: this.state.tail_1_e}}></div>
```

```
                <div className="m_westSecaucus_2_east"
style={{background: this.state.tail_2_e}}></div>
                {/* Switches */}
                <div className="westSecaucus_SW_1"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="m_westSecaucus_bridge"
style={{background: this.state.tail_center}}></div>
                <div className="westSecaucus_SW_3"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* West Side Tail Tracks */}
                <div className="m_westSecaucus_1_west"
style={{background: this.state.tail_1_w}}></div>
                <div className="m_westSecaucus_2_west"
style={{background: this.state.tail_2_w}}></div>
                {/* Signals */}
                <div className="westSecaucus_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="westSecaucus_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
                <div className="westSecaucus_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="westSecaucus_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawing = () => {
        // Loop through all the Routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|
__1_mill_westSecaucus" || this.state.routes[i] === "E_1_1__|
__2_westSecaucus_laurel") {
                // The Route Is Occupied
                if (this.state.occupied) {
                    // Set Tail Tracks Color
                    this.state.tail_1_e = Occupied;
```

```
                    this.state.tail_center = Occupied;
                    this.state.tail_1_w = Occupied;

                    // Switches
                    this.state.sw_1_src = SW_D_W_Occupied;
                    this.state.sw_3_src = SW_D_E_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Set Tail Track Colors
                    this.state.tail_1_e = Lined;
                    this.state.tail_center = Lined;
                    this.state.tail_1_w = Lined;

                    // Switches
                    this.state.sw_1_src = SW_D_W_Lined;
                    this.state.sw_3_src = SW_D_E_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_mill_westSecaucus") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_mill_westSecaucus" || this.state.routes[i] === "E_2_1__|
__2_westSecaucus_laurel") {
                // The Route Is Occupied
                if (this.state.occupied) {
                    // Set Tail Track Colors
                    this.state.tail_1_e = Occupied;
                    this.state.tail_center = Occupied;
```

```
                    this.state.tail_2_w = Occupied;

                    // Switches
                    this.state.sw_1_src = SW_D_W_R_Occupied;
                    this.state.sw_3_src = SW_D_E_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                else {
                    // Set Tail Track Colors
                    this.state.tail_1_e = Lined;
                    this.state.tail_center = Lined;
                    this.state.tail_2_w = Lined;

                    // Switches
                    this.state.sw_1_src = SW_D_W_R_Lined;
                    this.state.sw_3_src = SW_D_E_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_2__|
__2_mill_westSecaucus") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__1_mill_westSecaucus" || this.state.routes[i] === "E_1_2__|
__4_westSecaucus_laurel") {
                // The Route Is Occupied
                if (this.state.occupied) {
                    // Set Tail Track Colors
                    this.state.tail_2_e = Occupied;
                    this.state.tail_center = Occupied;
                    this.state.tail_1_w = Occupied;
```

```
                    // Switches
                    this.state.sw_1_src = SW_D_W_Occupied;
                    this.state.sw_3_src = SW_D_E_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Set Tail Track Colors
                    this.state.tail_2_e = Lined;
                    this.state.tail_center = Lined;
                    this.state.tail_1_w = Lined;

                    // Switches
                    this.state.sw_1_src = SW_D_W_Lined;
                    this.state.sw_3_src = SW_D_E_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_1__|
__1_mill_westSecaucus") {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|
__2_mill_westSecaucus" || this.state.routes[i] === "E_2_2__|
__4_westSecaucus_laurel") {
                // The Route Is Occupied
                if (this.state.occupied) {
                    // Set Tail Track Colors
                    this.state.tail_2_e = Occupied;
                    this.state.tail_center = Occupied;
                    this.state.tail_2_w = Occupied;

                    // Switches
```

```
                    this.state.sw_1_src = SW_D_W_R_Occupied;
                    this.state.sw_3_src = SW_D_E_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Set Tail Track Colors
                    this.state.tail_2_e = Lined;
                    this.state.tail_center = Lined;
                    this.state.tail_2_w = Lined;

                    // Switches
                    this.state.sw_1_src = SW_D_W_R_Lined;
                    this.state.sw_3_src = SW_D_E_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_mill_westSecaucus") {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
```

```
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img = () => {
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_D_W_R;
        }
        else {
            this.state.sw_1_src = SW_D_W;
        }

        if (this.state.sw_3) {
            this.state.sw_3_src = SW_D_E_R;
        }
        else {
            this.state.sw_3_src = SW_D_E;
        }
    }
    // ---- END set_switch_image() ----
}

// Export the interlocking to be drawn on the screen
export default WestSecaucus;
```

```
/**
 * @file BC.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for BC Interlocking
 *
 * Extends the React Component Class and is the UI part of the BC
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/bc.css';

// Import Images
// Switch Images
import SW_U_W from '../../../../public/images/SW_U_W_R.png';
import SW_U_W_Lined from '../../../../public/images/
SW_U_W_R_Lined.png';
import SW_U_W_Occupied from '../../../../public/images/
SW_U_W_R_Occupied.png';
import SW_U_W_R from '../../../../public/images/SW_U_W.png';
import SW_U_W_R_Lined from '../../../../public/images/
SW_U_W_Lined.png';
import SW_U_W_R_Occupied from '../../../../public/images/
SW_U_W_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the BC Interlocking
 *
 * This class is a JSX React Component for the BC Interlocking, this
will control all the UI for the comonent,
```

```
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class BC extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_U_W,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_e: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
```

```
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="bc_title">CP BC</div>
                <div className="bc_milepost">MP 86.7SR</div>
                {/* West Side Tail Tracks */}
                <div className="bc_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="bc_2_west" style={{background:
this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="bc_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="bc_east" style={{background:
this.state.tail_e}}></div>
                {/* Signals */}
                <div className="bc_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="bc_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
                <div className="bc_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----
```

```
    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawing() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_port_bc" ||
this.state.routes[i] ===
            "E_1_1__|__1_bc_ov") {
                // Tail Tracks
                this.state.tail_e = color;
                this.state.tail_1_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_port_bc") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4e_src = SIG_E_Stop;
```

```
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|__2_pa_bc" ||
this.state.routes[i] === "E_2_1__|__1_bc_ov") {
                // Tail Tracks
                this.state.tail_e = color;
                this.state.tail_2_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_2__|__2_pa_bc")
{
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----
```

```
    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_U_W_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_U_W;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_e = Empty;

        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
        this.state.sig_2w_src = SIG_W;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default BC;
```

```
/**
 * @file CentralValley.jsx
 * @author Joey Damico
 * @date July 16, 2019
 * @summary React JSX Component Class that is for CP Central Valley
 *
 * Extends the React Component Class and is the UI part of CP Central
Valley,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/centralValley.css';

// Import Images
// Switch Images
import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Central Valley Interlocking
 *
 * This class is a JSX React Component for the Central Valley
Interlocking, this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
```

```
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class CentralValley extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_21: this.props.status.sw_21,
        // Image File for the switch - Will change depending on route
        sw_21_src: SW_U_E,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_1w_src: SIG_W,
        sig_1e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_21: nextProps.status.sw_21,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----
```

```
    /**
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Title Text */}
                <div className="valley_title">CP CENTRAL VALLEY</div>
                <div className="valley_milepost">MP 47.8JS</div>

                {/* West Side Tail Tracks */}
                <div className="valley_west" style={{background:
this.state.tail_w}}></div>

                {/* Switches */}
                <div className="valley_SW_21"
onClick={this.props.throw_sw_21}><img src={this.state.sw_21_src}/></
div>

                {/* East Side Tail Tracks */}
                <div className="valley_2_east" style={{background:
this.state.tail_2_e}}></div>
                <div className="valley_1_east" style={{background:
this.state.tail_1_e}}></div>

                {/* Signal */}
                {/* West */}
                <div className="valley_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="valley_sig_1w"
onClick={this.props.click_sig_1w}><img src={this.state.sig_1w_src}/></
div>
                {/* East */}
                <div className="valley_sig_1e"
onClick={this.props.click_sig_1e}><img src={this.state.sig_1e_src}/></
div>
            </div>
```

```
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }
        for (let i = 0; i < this.state.routes.length; i++) {
            // Routes with Track 1 on both the West and East sides
            if (this.state.routes[i] === "W_1_1__|__1_hudson_valley"
|| this.state.routes[i] === "E_1_1__|__1_valley_harriman") {
                // Tail Tracks
                this.state.tail_1_e = color;
                this.state.tail_w = color;

                // Drawing if the interlocking is occupied
                if (this.state.occupied) {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_Occupied;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_1w_src = SIG_W_Stop;
                    this.state.sig_1e_src = SIG_E_Stop;
                }
                // Routing is not occupied
                else {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_Lined;

                    // Signal Images
                    // West Bound
                    if (this.state.routes[i] === "W_1_1__|
__1_hudson_valley") {
                        this.state.sig_2w_src = SIG_W_Stop;
```

```
                        this.state.sig_1w_src = SIG_W_Clear;
                        this.state.sig_1e_src = SIG_E_Stop;
                    }
                    // East Bound
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_1w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Clear;
                    }
                }
            }
            // Routes With Track 2 on West Side and Track 1 on East
Side
            else if (this.state.routes[i] === "W_2_1__|
__1_hudson_valley" || this.state.routes[i] === "E_1_2__|
__2_valley_harriman") {
                // Tail Tracks
                this.state.tail_2_e = color;
                this.state.tail_w = color;

                // Drawing if the interlocking is occupied
                if (this.state.occupied) {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_R_Occupied;

                    // Signal Images
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_1w_src = SIG_W_Stop;
                    this.state.sig_1e_src = SIG_E_Stop;
                }
                // Routing that is not occupied
                else {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_R_Lined;

                    // Signal Images
                    // West Bound Route
                    if (this.state.routes[i] === "W_2_1__|
__1_hudson_valley") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_1w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Stop;
                    }
                    // East Bound Route
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_1w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Clear;
                    }
                }
```

```
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Switch #21
        // Reversed
        if (this.state.sw_21) {
            this.state.sw_21_src = SW_U_E_R;
        }
        // Normal
        else {
            this.state.sw_21_src = SW_U_E;
        }
    }
    // ---- END set_switch_image() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.sig_2w_src = SIG_W;
        this.state.sig_1w_src = SIG_W;
        this.state.sig_1e_src = SIG_E;

        this.state.tail_w = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
    }
    //---- END reset_drawings() ----
}
```

```
// Export the interlocking to be drawn on the screen
export default CentralValley;
```

```
/**
 * @file Hall.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Hall Interlocking
 *
 * Extends the React Component Class and is the UI part of the Hall
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/hall.css';

// Import Images
// Switch Images
import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';
```

```
// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Hall Interlocking
 *
 * This class is a JSX React Component for the Hall Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Hall extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        // Image File for the switch - Will change depending on route
        sw_1_src: CX_225,
        // Colors for tail tracks - Will change depending on route
        tail_yard: Empty,
        tail_west: Empty,
        tail_2_east: Empty,
        tail_1_east: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Information For Interlocking Routes
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_trk_1,
        route_2: this.props.status.routed_trk_2,
```

```
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_trk_1,
            route_2: nextProps.status.routed_trk_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="hall_title">CP HALL</div>
                <div className="hall_milepost">MP 64.7JS</div>
                {/* West Side Tail Tracks */}
                <div className="hall_yard" style={{background:
this.state.tail_yard}}></div>
                <div className="hall_west" style={{background:
this.state.tail_west}}></div>
```

```
                {/* Switches */}
                <div className="hall_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="hall_2_east" style={{background:
this.state.tail_2_east}}></div>
                <div className="hall_1_east" style={{background:
this.state.tail_1_east}}></div>
                {/* Signals */}
                <div className="hall_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="hall_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="hall_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
                <div className="hall_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
```

```
            color_2 = Red;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_howells_hall" ||
this.state.routes[i] === "E_1_1__|__1_hall_hudson") {
                // Tail Tracks
                this.state.tail_1_east = color_1;
                this.state.tail_west = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    // Crossovers that could change based off of Track
#2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Crossovers that could change based off of Track
#2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                    }
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Nothing Track #2
                    else {
```

```
                        this.state.sw_1_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_howells_hall") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|__2_yard_hall"
|| this.state.routes[i] === "E_2_2__|__2_hall_hudson") {
                // Tail Tracks
                this.state.tail_2_east = color_2;
                this.state.tail_yard = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    // Crossovers that could change based off of Track
#1
                    // Track #1 Routed
                    if (this.state.route_1) {
                        this.state.sw_1_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Track #1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_1_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Crossovers that could change based off of Track
```

```
#1
                    // Track #1 Routed
                    if (this.state.route_1) {
                        this.state.sw_1_src = CX_225_Lined_Both;
                    }
                    // Track #1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_1_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_1_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_yard_hall") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__1_howells_hall") {
                // Tail Tracks
                this.state.tail_2_east = color_2;
                this.state.tail_west = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Lined;
```

```
                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Clear;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_1_2__|
__2_hall_hudson") {
                // Tail Tracks
                this.state.tail_2_east = color_1;
                this.state.tail_west = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = CX_225_R_Lined;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Clear;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
```

```
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        if (this.state.sw_1) {
            this.state.sw_1_src = CX_225_R;
        }
        else {
            this.state.sw_1_src = CX_225;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_east = Empty;
        this.state.tail_2_east = Empty;
        this.state.tail_west = Empty;
        this.state.tail_yard = Empty;

        this.state.sig_2w_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Hall;
```

```
/**
 * @file Harriman.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Harriman
Interlocking
 *
 * Extends the React Component Class and is the UI part of the
Harriman Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/harriman.css';

// Import Images
// Switch Images
import SW_U_W from '../../../../public/images/SW_U_W.png';
import SW_U_W_Lined from '../../../../public/images/SW_U_W_Lined.png';
import SW_U_W_Occupied from '../../../../public/images/
SW_U_W_Occupied.png';
import SW_U_W_R from '../../../../public/images/SW_U_W_R.png';
import SW_U_W_R_Lined from '../../../../public/images/
SW_U_W_R_Lined.png';
import SW_U_W_R_Occupied from '../../../../public/images/
SW_U_W_R_Occupied.png';
import SW_D_W from '../../../../public/images/SW_D_W.png';
import SW_D_W_Lined from '../../../../public/images/SW_D_W_Lined.png';
import SW_D_W_Occupied from '../../../../public/images/
SW_D_W_Occupied.png';
import SW_D_W_R from '../../../../public/images/SW_D_W_R.png';
import SW_D_W_R_Lined from '../../../../public/images/
SW_D_W_R_Lined.png';
import SW_D_W_R_Occupied from '../../../../public/images/
SW_D_W_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
```

```
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Harriman Interlocking
 *
 * This class is a JSX React Component for the Harriman Interlocking,
this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Harriman extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_21: this.props.sw_21,
        sw_32: this.props.sw_32,
        // Image File for the switch - Will change depending on route
        sw_21_src: SW_U_W,
        sw_32_src: SW_D_W,
        // Image File for the signals - Will change depending on route
        sig_1w_src: SIG_W,
        sig_1e_src: SIG_E,
        sig_2e_src: SIG_E,
        sig_3e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_ind: Empty,
        tail_e: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
```

```
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_21: nextProps.status.sw_21,
            sw_32: nextProps.status.sw_32,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="harriman_title">CP HARRIMAN</div>
                <div className="harriman_milepost">MP 45.0JS</div>
                {/* West Side Tail Tracks */}
                <div className="harriman_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="harriman_2_west" style={{background:
this.state.tail_2_w}}></div>
                <div className="harriman_industrial"
style={{background: this.state.tail_ind}}></div>
                {/* Switches */}
                <div className="harriman_SW_21"
onClick={this.props.throw_sw_21}><img src={this.state.sw_21_src}/></
div>
```

```
                <div className="harriman_SW_32"
onClick={this.props.throw_sw_32}><img src={this.state.sw_32_src}/></
div>
                {/* East Side Tail Tracks */}
                <div className="harriman_1_east" style={{background:
this.state.tail_e}}></div>
                {/* Signals */}
                <div className="harriman_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="harriman_sig_1e"
onClick={this.props.click_sig_1e}><img src={this.state.sig_1e_src}/></
div>
                <div className="harriman_sig_3e"
onClick={this.props.click_sig_3e}><img src={this.state.sig_3e_src}/></
div>
                <div className="harriman_sig_1w"
onClick={this.props.click_sig_1w}><img src={this.state.sig_1w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_valley_harriman"
|| this.state.routes[i] === "E_1_1__|__1_harriman_sterling") {
                // Tail Tracks
                this.state.tail_1_w = color;
                this.state.tail_e = color;

                // The Route Is Occupied
```

```
                if (this.state.occupied) {
                    // Switch Images
                    this.state.sw_21_src = SW_U_W_Occupied;
                    this.state.sw_32_src = SW_D_W_Occupied;

                    // Signal Images
                    this.state.sig_1w_src = SIG_W_Stop;
                    this.state.sig_1e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_3e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_21_src = SW_U_W_Lined;
                    this.state.sw_32_src = SW_D_W_Lined;

                    // Signal Images
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_valley_harriman") {
                        this.state.sig_1w_src = SIG_W_Clear;
                        this.state.sig_1e_src = SIG_E_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_3e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_1w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_3e_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_valley_harriman" || this.state.routes[i] === "E_2_1__|
__1_harriman_sterling") {
                // Tail Tracks
                this.state.tail_2_w = color;
                this.state.tail_e = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switch Images
                    this.state.sw_21_src = SW_U_W_R_Occupied;
                    this.state.sw_32_src = SW_D_W_Occupied;

                    // Signal Images
                    this.state.sig_1w_src = SIG_W_Stop;
```

```
                    this.state.sig_1e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_3e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_21_src = SW_U_W_R_Lined;
                    this.state.sw_32_src = SW_D_W_Lined;

                    // Signal Images
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_2__|
__2_valley_harriman") {
                        this.state.sig_1w_src = SIG_W_Clear;
                        this.state.sig_1e_src = SIG_E_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_3e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_1w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_3e_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_3__|
__3_industrial_harriman" || this.state.routes[i] === "E_3_1__|
__1_harriman_sterling") {
                // Tail Tracks
                this.state.tail_ind = color;
                this.state.tail_e = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switch Images
                    this.state.sw_32_src = SW_D_W_R_Occupied;

                    // Signals
                    this.state.sig_1w_src = SIG_W_Stop;
                    this.state.sig_1e_src = SIG_E_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_3e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switch Images
                    this.state.sw_32_src = SW_D_W_R_Lined;
```

```
                    // Signal Images
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_3__|
__3_industrial_harriman") {
                        this.state.sig_1w_src = SIG_W_Clear;
                        this.state.sig_1e_src = SIG_E_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_3e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_1w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_3e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #21
        // SW #21 Reversed
        if (this.state.sw_21) {
            this.state.sw_21_src = SW_U_W_R;
        }
        // SW #21 Normal
        else {
            this.state.sw_21_src = SW_U_W;
        }

        // Set SW #32
        // SW #32 Reversed
        if (this.state.sw_32) {
            this.state.sw_32_src = SW_D_W_R;
        }
```

```
        // SW #32 Normal
        else {
            this.state.sw_32_src = SW_D_W;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.sig_1w_src = SIG_W;
        this.state.sig_1e_src = SIG_E;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_3e_src = SIG_E;

        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_ind = Empty;
        this.state.tail_e = Empty;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Harriman;
```

```
/**
 * @file Howells.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Howells Interlocking
 *
 * Extends the React Component Class and is the UI part of the Howells
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/howells.css';

// Import Images
// Switch Images
import SW_U_W from '../../../../public/images/SW_U_W.png';
import SW_U_W_Lined from '../../../../public/images/SW_U_W_Lined.png';
import SW_U_W_Occupied from '../../../../public/images/
SW_U_W_Occupied.png';
import SW_U_W_R from '../../../../public/images/SW_U_W_R.png';
import SW_U_W_R_Lined from '../../../../public/images/
SW_U_W_R_Lined.png';
import SW_U_W_R_Occupied from '../../../../public/images/
SW_U_W_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Howells Interlocking
 *
 * This class is a JSX React Component for the Howells Interlocking,
this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
```

```
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Howells extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_3_src: SW_U_W,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_e: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_2es_src: SIG_E,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_3: nextProps.status.sw_3,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
```

```
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="howells_title">CP HOWELLS</div>
                <div className="howells_milepost">MP 69.1SR</div>
                {/* West Side Tail Tracks */}
                <div className="howells_2_west" style={{background:
this.state.tail_2_w}}></div>
                <div className="howells_1_west" style={{background:
this.state.tail_1_w}}></div>
                {/* Switches */}
                <div className="howells_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="howells_east" style={{background:
this.state.tail_e}}></div>
                {/* Signals */}
                <div className="howells_sig_2es"
onClick={this.props.click_sig_2es}><img src={this.state.sig_2es_src}/
></div>
                <div className="howells_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="howells_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----
```

```
    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_ov_howells" ||
this.state.routes[i] === "E_1_1__|__1_howells_hall") {
                // Tail Tracks
                this.state.tail_e = color;
                this.state.tail_1_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_3_src = SW_U_W_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = SW_U_W_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_ov_howells") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                    // East Bound Signals
```

```
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|__2_ov_howells"
|| this.state.routes[i] === "E_2_1__|__1_howells_hall") {
                // Tail Tracks
                this.state.tail_e = color;
                this.state.tail_2_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_3_src = SW_U_W_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2es_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = SW_U_W_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_2__|
__2_ov_howells") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
```

```
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = SW_U_W_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = SW_U_W;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_e = Empty;

        this.state.sig_2w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_2es_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Howells;
```

```
/**
 * @file HudsonJunction.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Hudson Junction
Interlocking
 *
 * Extends the React Component Class and is the UI part of the Hudson
Junction Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/hudsonJunction.css';

// Import Images
// Switch Images
import SW_D_E from '../../../../public/images/SW_D_E.png';
import SW_D_E_Lined from '../../../../public/images/SW_D_E_Lined.png';
import SW_D_E_Occupied from '../../../../public/images/
SW_D_E_Occupied.png';
import SW_D_E_R from '../../../../public/images/SW_D_E_R.png';
import SW_D_E_R_Lined from '../../../../public/images/
SW_D_E_R_Lined.png';
import SW_D_E_R_Occupied from '../../../../public/images/
SW_D_E_R_Occupied.png';
import SW_U_W from '../../../../public/images/SW_U_W.png';
import SW_U_W_Lined from '../../../../public/images/SW_U_W_Lined.png';
import SW_U_W_Occupied from '../../../../public/images/
SW_U_W_Occupied.png';
import SW_U_W_R from '../../../../public/images/SW_U_W_R.png';
import SW_U_W_R_Lined from '../../../../public/images/
SW_U_W_R_Lined.png';
import SW_U_W_R_Occupied from '../../../../public/images/
SW_U_W_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
```

```
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Hudson Junction Interlocking
 *
 * This class is a JSX React Component for the Hudson Junction
Interlocking, this will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class HudsonJunction extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_U_W,
        sw_3_src: SW_D_E,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_2ws_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_2es_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
        tail_e: Empty,
        tail_nysw: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
```

```
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="hudson_title">CP HUDSON JUNCTION</div>
                <div className="hudson_milepost">MP 63.4JS</div>
                {/* West Side Tail Tracks */}
                <div className="hudson_2_west" style={{background:
this.state.tail_2_w}}></div>
                <div className="hudson_1_west" style={{background:
this.state.tail_1_w}}></div>
                {/* Switches */}
                <div className="hudson_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="hudson_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
```

```
                <div className="hudson_east" style={{background:
this.state.tail_e}}></div>
                <div className="hudson_nysw" style={{background:
this.state.tail_nysw}}></div>
                {/* Signals */}
                <div className="hudson_sig_2es"
onClick={this.props.click_sig_2es}><img src={this.state.sig_2es_src}/
></div>
                <div className="hudson_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="hudson_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="hudson_sig_2ws"
onClick={this.props.click_sig_2ws}><img src={this.state.sig_2ws_src}/
></div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_hall_hudson" ||
this.state.routes[i] === "E_1_1__|__1_hudson_valley") {
                // Tail Tracks
                this.state.tail_1_w = color;
                this.state.tail_e = color;

                // The Route Is Occupied
```

```
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Occupied;
                    this.state.sw_3_src = SW_D_E_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2es_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Lined;
                    this.state.sw_3_src = SW_D_E_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_hall_hudson") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_hall_hudson" || this.state.routes[i] === "E_2_1__|
__1_hudson_valley") {
                // Set Tail Track Colors
                this.state.tail_2_w = color;
                this.state.tail_e = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Occupied;
                    this.state.sw_3_src = SW_D_E_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
```

```
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2es_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Lined;
                    this.state.sw_3_src = SW_D_E_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_2__|
__2_hall_hudson") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_1__|
__1_hall_hudson" || this.state.routes[i] === "E_1_3__|
__1_hudson_nysw") {
                // Set Tail Track Colors
                this.state.tail_1_w = color;
                this.state.tail_nysw = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Occupied;
                    this.state.sw_3_src = SW_D_E_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2es_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
```

```
                    this.state.sw_1_src = SW_U_W_Lined;
                    this.state.sw_3_src = SW_D_E_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_1__|
__1_hall_hudson") {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_2__|
__2_hall_hudson" || this.state.routes[i] === "E_2_3__|
__1_hudson_nysw") {
                // Set Tail Track Colors
                this.state.tail_2_w = color;
                this.state.tail_nysw = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Occupied;
                    this.state.sw_3_src = SW_D_E_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2es_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Lined;
                    this.state.sw_3_src = SW_D_E_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_2__|
__2_hall_hudson") {
```

```
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2es_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_U_W_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_U_W;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = SW_D_E_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = SW_D_E;
        }
    }
```

```
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_e = Empty;
        this.state.tail_nysw = Empty;

        this.state.sig_2w_src = SIG_W;
        this.state.sig_2ws_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_2es_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default HudsonJunction;
```

```
/**
 * @file OV.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for OV Interlocking
 *
 * Extends the React Component Class and is the UI part of the OV
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/ov.css';

// Import Images
// Switch Images
import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the OV Interlocking
 *
 * This class is a JSX React Component for the OV Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
```

```
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class OV extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_U_E,
        // Colors for tail tracks - Will change depending on route
        tail_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_2ws_src: SIG_W,
        sig_2e_src: SIG_E,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
```

```
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="ov_title">CP OV</div>
                <div className="ov_milepost">MP 75.0SR</div>
                {/* West Side Tail Tracks */}
                <div className="ov_west" style={{background:
this.state.tail_w}}></div>
                {/* Switches */}
                <div className="ov_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="ov_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="ov_2_east" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="ov_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="ov_sig_2ws"
onClick={this.props.click_sig_2ws}><img src={this.state.sig_2ws_src}/
></div>
                <div className="ov_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----
```

```
    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawing() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_bc_ov" ||
this.state.routes[i] === "E_1_1__|__1_ov_howells") {
                // Tail Tracks
                this.state.tail_1_e = color;
                this.state.tail_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_E_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_E_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|__1_bc_ov")
{
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
```

```
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_1__|__1_bc_ov" ||
this.state.routes[i] === "E_1_2__|__2_ov_howells") {
                // Tail Tracks
                this.state.tail_2_e = color;
                this.state.tail_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_E_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_E_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_1__|__1_bc_ov")
{
                        this.state.sig_2ws_src = SIG_W_Clear;
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
```

```
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_U_E_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_U_E;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_w = Empty;

        this.state.sig_2w_src = SIG_W;
        this.state.sig_2ws_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default OV;
```

```
/**
 * @file PA.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for PA Interlocking
 *
 * Extends the React Component Class and is the UI part of the PA
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/pa.css';

// Import Images
// Switch Images
import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

import CX_225 from '../../../../public/images/CX_225.png';
import CX_225_Lined_Top from '../../../../public/images/
CX_225_Lined_Top.png';
import CX_225_Lined_Bottom from '../../../../public/images/
CX_225_Lined_Bottom.png';
import CX_225_Lined_Both from '../../../../public/images/
CX_225_Lined_Both.png';
import CX_225_R from '../../../../public/images/CX_225_R.png';
import CX_225_R_Lined from '../../../../public/images/
CX_225_R_Lined.png';
import CX_225_Lined_Top_Occupied_Bottom from '../../../../public/
images/CX_225_Lined_Top_Occupied_Bottom.png';
import CX_225_Occupied_Top_Lined_Bottom from '../../../../public/
images/CX_225_Occupied_Top_Lined_Bottom.png';
import CX_225_Occupied_Top from '../../../../public/images/
CX_225_Occupied_Top.png';
import CX_225_Occupied_Bottom from '../../../../public/images/
CX_225_Occupied_Bottom.png';
import CX_225_Occupied_Both from '../../../../public/images/
CX_225_Occupied_Both.png';
```

```
import CX_225_R_Occupied from '../../../../public/images/
CX_225_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the PA Interlocking
 *
 * This class is a JSX React Component for the PA Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class PA extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_U_E,
        sw_3_src: CX_225,
        // Colors for tail tracks - Will change depending on route
        tail_1_w: Empty,
        tail_2_w: Empty,
```

```
        tail_1_e: Empty,
        tail_2_e: Empty,
        tail_yard: Empty,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_4w_src: SIG_W,
        sig_2e_src: SIG_E,
        sig_4e_src: SIG_E,
        // Information For Interlocking Routes
        occupied_1: this.props.status.occupied_trk_1,
        occupied_2: this.props.status.occupied_trk_2,
        route_1: this.props.status.routed_trk_1,
        route_2: this.props.status.routed_trk_2,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            occupied_1: nextProps.status.occupied_trk_1,
            occupied_2: nextProps.status.occupied_trk_2,
            route_1: nextProps.status.routed_trk_1,
            route_2: nextProps.status.routed_trk_2,
            routes: nextProps.status.routes
        });
    }
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
```

```
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="pa_title">CP PA</div>
                <div className="pa_milepost">MP 87.9SR</div>
                {/* West Side Tail Tracks */}
                <div className="pa_1_west" style={{background:
this.state.tail_1_w}}></div>
                <div className="pa_2_west" style={{background:
this.state.tail_2_w}}></div>
                {/* Switches */}
                <div className="pa_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="pa_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="pa_yard" style={{background:
this.state.tail_yard}}></div>
                <div className="pa_1_east" style={{background:
this.state.tail_1_e}}></div>
                <div className="pa_2_east" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="pa_sig_2w-2"
onClick={this.props.click_sig_2w_2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="pa_sig_2w-1"
onClick={this.props.click_sig_2w_1}><img src={this.state.sig_2w1_src}/
></div>
                <div className="pa_sig_4w"
onClick={this.props.click_sig_4w}><img src={this.state.sig_4w_src}/></
div>
                <div className="pa_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
                <div className="pa_sig_4e"
onClick={this.props.click_sig_4e}><img src={this.state.sig_4e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
```

```
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        let color_1 = Empty;
        let color_2 = Empty;

        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        if (this.state.route_1) {
            color_1 = Green;
        }
        if (this.state.route_2) {
            color_2 = Green;
        }
        if (this.state.occupied_1) {
            color_1 = Red;
        }
        if (this.state.occupied_2) {
            color_2 = Red;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_sparrow_pa" ||
this.state.routes[i] === "E_1_1__|__1_pa_port") {
                // Tail Tracks
                this.state.tail_1_w = color_1;
                this.state.tail_1_e = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_1_src = SW_U_E_Occupied;

                    // Crossovers that could change based off of Track
#2 state
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #2
```

```
                    else {
                        this.state.sw_3_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_E_Lined;

                    // Crossovers that could change based off of Track
#2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_sparrow_pa") {
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_2__|__2_sparrow_pa"
|| this.state.routes[i] === "E_2_2__|__2_pa_bc") {
                // Tail Tracks
                this.state.tail_2_w = color_2;
```

```
                this.state.tail_2_e = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    // Crossovers that could change based off of Track
#1
                    // Track #1 Routed
                    if (this.state.route_1) {
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Track #1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_3_src = CX_225_Occupied_Bottom;
                    }

                    // Signals
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    // Crossovers that could change based off of Track
#1
                    // Track #1 Routed
                    if (this.state.route_1) {
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
                    // Track #1 Occupied
                    else if (this.state.occupied_1) {
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Nothing Track #1
                    else {
                        this.state.sw_3_src = CX_225_Lined_Bottom;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_2_2__|
__2_sparrow_pa") {
                        this.state.sig_4w_src = SIG_W_Clear;
                        this.state.sig_4e_src = SIG_E_Stop;
```

```
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_4w_src = SIG_W_Stop;
                        this.state.sig_4e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_1__|__1_sparrow_pa"
|| this.state.routes[i] === "E_1_3__|__0_portYard_west") {
                // Tail Tracks
                this.state.tail_1_w = color_1;
                this.state.tail_yard = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_1_src = SW_U_E_R_Occupied;

                    // Crossovers that could change based of Track #2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_3_src =
CX_225_Occupied_Top_Lined_Bottom;
                    }
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_3_src = CX_225_Occupied_Both;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Occupied_Top;
                    }

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_E_R_Lined;

                    // Crossovers that could change based off of Track
#2
                    // Track #2 Routed
                    if (this.state.route_2) {
                        this.state.sw_3_src = CX_225_Lined_Both;
                    }
```

```
                    // Track #2 Occupied
                    else if (this.state.occupied_2) {
                        this.state.sw_3_src =
CX_225_Lined_Top_Occupied_Bottom;
                    }
                    // Nothing Track #2
                    else {
                        this.state.sw_3_src = CX_225_Lined_Top;
                    }

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_3_1__|
__1_sparrow_pa") {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2e_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2e_src = SIG_E_Clear;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_2__|
__2_sparrow_pa") {
                // Tail Tracks
                this.state.tail_2_w = color_1;
                this.state.tail_yard = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_U_E_R_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = SW_U_E_R_Lined;
```

```
                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Clear;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] === "E_2_3__|
__0_portYard_west") {
                // Tail Tracks
                this.state.tail_2_w = color_2;
                this.state.tail_yard = color_2;

                // The Route Is Occupied
                if (this.state.occupied_1) {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_U_E_R_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = SW_U_E_R_Lined;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
            else if (this.state.routes[i] === "W_1_2__|
__2_sparrow_pa") {
                // Tail Tracks
                this.state.tail_2_w = color_1;
                this.state.tail_1_e = color_1;

                // The Route Is Occupied
                if (this.state.occupied_1) {
```

```
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_U_E_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = SW_U_E_Lined;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Clear;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
            }
            else if (this.state.routes[i] ===  "E_2_1__|__1_pa_port")
{
                // Tail Tracks
                this.state.tail_2_w = color_2;
                this.state.tail_1_e = color_2;

                // The Route Is Occupied
                if (this.state.occupied_2) {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Occupied;
                    this.state.sw_1_src = SW_U_E_Occupied;

                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = CX_225_R_Lined;
                    this.state.sw_1_src = SW_U_E_Lined;
```

```
                    // Signals
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_4w_src = SIG_W_Stop;
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_4e_src = SIG_E_Clear;
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_U_E_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_U_E;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = CX_225_R;
        }
        // SW #3 Normal
        else {
            this.state.sw_3_src = CX_225;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
```

```
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_1_w = Empty;
        this.state.tail_2_w = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_yard = Empty;

        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_4w_src = SIG_W;
        this.state.sig_2e_src = SIG_E;
        this.state.sig_4e_src = SIG_E;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default PA;
```

```
/**
 * @file PA.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for PA Interlocking
 *
 * Extends the React Component Class and is the UI part of the PA
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/port.css';

// Import Images
// Switch Images
import SW_U_W from '../../../../public/images/SW_U_W.png';
import SW_U_W_Lined from '../../../../public/images/SW_U_W_Lined.png';
import SW_U_W_Occupied from '../../../../public/images/
SW_U_W_Occupied.png';
import SW_U_W_R from '../../../../public/images/SW_U_W_R.png';
import SW_U_W_R_Lined from '../../../../public/images/
SW_U_W_R_Lined.png';
import SW_U_W_R_Occupied from '../../../../public/images/
SW_U_W_R_Occupied.png';
// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the PA Interlocking
 *
 * This class is a JSX React Component for the PA Interlocking, this
will control all the UI for the comonent,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
```

```
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Port extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_U_W,
        // Colors for tail tracks - Will change depending on route
        tail_yard: Empty,
        tail_west: Empty,
        tail_east: Empty,
        // Image File for the signals - Will change depending on route
        sig_2e_1_src: SIG_E,
        sig_2e_2_src: SIG_E,
        sig_2w_src: SIG_W,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }
```

```
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawing();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="port_title">CP PORT</div>
                <div className="port_milepost">MP 87.5SR</div>
                {/* West Side Tail Tracks */}
                <div className="port_yard" style={{background:
this.state.tail_yard}}></div>
                <div className="port_west" style={{background:
this.state.tail_west}}></div>
                {/* Switches */}
                <div className="port_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="port_east" style={{background:
this.state.tail_east}}></div>
                {/* Signals */}
                <div className="port_sig_2e-2"
onClick={this.props.click_sig_2e_2}><img
src={this.state.sig_2e_2_src}/></div>
                <div className="port_sig_2e-1"
onClick={this.props.click_sig_2e_1}><img
src={this.state.sig_2e_1_src}/></div>
                <div className="port_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
```

```
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawing() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_pa_port" ||
this.state.routes[i] === "E_1_1__|__1_port_bc") {
                // Tail Tracks
                this.state.tail_east = color;
                this.state.tail_west = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_1_src = SIG_E_Stop;
                    this.state.sig_2e_2_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_1__|
__1_pa_port") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_1_src = SIG_E_Stop;
                        this.state.sig_2e_2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
```

```
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_1_src = SIG_E_Clear;
                        this.state.sig_2e_2_src = SIG_E_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_1_3__|
__3_yardEast_port" || this.state.routes[i] === "E_3_1__|__1_port_bc")
{
                // Tail Tracks
                this.state.tail_east = color;
                this.state.tail_yard = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Occupied;

                    // Signals
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_2e_1_src = SIG_E_Stop;
                    this.state.sig_2e_2_src = SIG_E_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_W_R_Lined;

                    // Signals
                    // West Bound Signals
                    if (this.state.routes[i] === "W_1_3__|
__3_yardEast_port") {
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_2e_1_src = SIG_E_Stop;
                        this.state.sig_2e_2_src = SIG_E_Stop;
                    }
                    // East Bound Signals
                    else {
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_2e_1_src = SIG_E_Stop;
                        this.state.sig_2e_2_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
```

```
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_U_W_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_U_W;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.tail_yard = Empty;
        this.state.tail_west = Empty;
        this.state.tail_east = Empty;

        this.state.sig_2e_1_src = SIG_E;
        this.state.sig_2e_2_src = SIG_E;
        this.state.sig_2w_src = SIG_W;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Port;
```

```
/**
 * @file SouthernTierTracks.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for The Tracks of the
Southern Tier
 *
 * Extends the React Component Class and is the UI part of the
Southern Tier Tracks,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/southernTier.css';


/**
 * The React JSX Component Class for the Tracks in the Southern Tier
portion
 *
 * This class is a JSX React Component for the Southern Tier Tracks,
this will control all the UI for the comonent,
 * showing what blocks are occupied by a train
 */
class SouthernTierTracks extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the tracks that is
required to display the routes
     * correctly
     */
    state = {
        // Train Symbols
        symbol_bingo_sparrow: this.props.symbols.symbol_bingo_sparrow,
        symbol_sparrow_pa_1: this.props.symbols.symbol_sparrow_pa_1,
        symbol_sparrow_pa_2: this.props.symbols.symbol_sparrow_pa_2,
        symbol_pa_port_1: this.props.symbols.symbol_pa_port_1,
        symbol_port_bc_1: this.props.symbols.symbol_port_bc_1,
        symbol_pa_bc_2: this.props.symbols.symbol_pa_bc_2,
        symbol_port_yardEast: this.props.symbols.symbol_port_yardEast,
        symbol_bc_ov: this.props.symbols.symbol_bc_ov,
        symbol_ov_howells_1: this.props.symbols.symbol_ov_howells_1,
        symbol_ov_howells_2: this.props.symbols.symbol_ov_howells_2,
```

```
        // Second Row
        symbol_howells_hall: this.props.symbols.symbol_howells_hall,
        symbol_hall_yard: this.props.symbols.symbol_hall_yard,
        symbol_hall_hudson_1: this.props.symbols.symbol_hall_hudson_1,
        symbol_hall_hudson_2: this.props.symbols.symbol_hall_hudson_2,
        symbol_hudson_valley: this.props.symbols.symbol_hudson_valley,
        symbol_hudson_nysw: this.props.symbols.symbol_hudson_nysw,
        symbol_valley_harriman_1:
this.props.symbols.symbol_valley_harriman_1,
        symbol_valley_harriman_2:
this.props.symbols.symbol_valley_harriman_2,
        // Third Row
        symbol_harriman_sterling:
this.props.symbols.symbol_harriman_sterling,
        symbol_harriman_industrial:
this.props.symbols.symbol_harriman_industrial,


        // Blocks
        harriman_sterling_1:
this.props.blocks.block_harriman_sterling_1,

        valley_harriman_1: this.props.blocks.block_valley_harriman_1,
        valley_harriman_2: this.props.blocks.block_valley_harriman_2,
        harriman_industrial:
this.props.blocks.block_harriman_industrial,

        hudson_valley_1: this.props.blocks.block_hudson_valley_1,
        hudson_nysw: this.props.blocks.block_hudson_nysw,

        hall_hudson_1: this.props.blocks.block_hall_hudson_1,
        hall_hudson_2: this.props.blocks.block_hall_hudson_2,
        hall_yard: this.props.blocks.block_hall_yard,

        howells_hall_1: this.props.blocks.block_howells_hall_1,

        ov_howells_1: this.props.blocks.block_ov_howells_1,
        ov_howells_2: this.props.blocks.block_ov_howells_2,

        bc_ov_1: this.props.blocks.block_bc_ov_1,

        port_bc_1: this.props.blocks.block_port_bc_1,
        pa_port_1: this.props.blocks.block_pa_port_1,
        pa_bc_2: this.props.blocks.block_pa_bc_2,
        port_yard_west: this.props.blocks.block_port_yard_west,
        port_yard_east: this.props.blocks.block_port_yard_east,

        buckleys_west: this.props.blocks.block_buckleys_west,
        buckleys_east: this.props.blocks.block_buckleys_east,
```

```
        sparrow_pa_1: this.props.blocks.block_sparrow_pa_1,
        sparrow_pa_2: this.props.blocks.block_sparrow_pa_2,
        sparrow_cripple: this.props.blocks.block_sparrow_cripple,

        bingo_sparrow: this.props.blocks.block_bingo_sparrow
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            // Train Symbols
            symbol_bingo_sparrow:
nextProps.symbols.symbol_bingo_sparrow,
            symbol_sparrow_pa_1:
nextProps.symbols.symbol_sparrow_pa_1,
            symbol_sparrow_pa_2:
nextProps.symbols.symbol_sparrow_pa_2,
            symbol_pa_port_1: nextProps.symbols.symbol_pa_port_1,
            symbol_port_bc_1: nextProps.symbols.symbol_port_bc_1,
            symbol_pa_bc_2: nextProps.symbols.symbol_pa_bc_2,
            symbol_port_yardEast:
nextProps.symbols.symbol_port_yardEast,
            symbol_bc_ov: nextProps.symbols.symbol_bc_ov,
            symbol_ov_howells_1:
nextProps.symbols.symbol_ov_howells_1,
            symbol_ov_howells_2:
nextProps.symbols.symbol_ov_howells_2,
            // Second Row
            symbol_howells_hall:
nextProps.symbols.symbol_howells_hall,
            symbol_hall_yard: nextProps.symbols.symbol_hall_yard,
            symbol_hall_hudson_1:
nextProps.symbols.symbol_hall_hudson_1,
            symbol_hall_hudson_2:
nextProps.symbols.symbol_hall_hudson_2,
            symbol_hudson_valley:
nextProps.symbols.symbol_hudson_valley,
            symbol_hudson_nysw: nextProps.symbols.symbol_hudson_nysw,
            symbol_valley_harriman_1:
nextProps.symbols.symbol_valley_harriman_1,
            symbol_valley_harriman_2:
nextProps.symbols.symbol_valley_harriman_2,
```

```
            // Third Row
            symbol_harriman_sterling:
nextProps.symbols.symbol_harriman_sterling,
            symbol_harriman_industrial:
nextProps.symbols.symbol_harriman_industrial,


            // Blocks
            harriman_sterling_1:
nextProps.blocks.block_harriman_sterling_1,

            valley_harriman_1:
nextProps.blocks.block_valley_harriman_1,
            valley_harriman_2:
nextProps.blocks.block_valley_harriman_2,
            harriman_industrial:
nextProps.blocks.block_harriman_industrial,

            hudson_valley_1: nextProps.blocks.block_hudson_valley_1,
            hudson_nysw: nextProps.blocks.block_hudson_nysw,

            hall_hudson_1: nextProps.blocks.block_hall_hudson_1,
            hall_hudson_2: nextProps.blocks.block_hall_hudson_2,
            hall_yard: nextProps.blocks.block_hall_yard,

            howells_hall_1: nextProps.blocks.block_howells_hall_1,

            ov_howells_1: nextProps.blocks.block_ov_howells_1,
            ov_howells_2: nextProps.blocks.block_ov_howells_2,

            bc_ov_1: nextProps.blocks.block_bc_ov_1,

            port_bc_1: nextProps.blocks.block_port_bc_1,
            pa_port_1: nextProps.blocks.block_pa_port_1,
            pa_bc_2: nextProps.blocks.block_pa_bc_2,
            port_yard_west: nextProps.blocks.block_port_yard_west,
            port_yard_east: nextProps.blocks.block_port_yard_east,

            buckleys_west: nextProps.blocks.block_buckleys_west,
            buckleys_east: nextProps.blocks.block_buckleys_east,

            sparrow_pa_1: nextProps.blocks.block_sparrow_pa_1,
            sparrow_pa_2: nextProps.blocks.block_sparrow_pa_2,
            sparrow_cripple: nextProps.blocks.block_sparrow_cripple,

            bingo_sparrow: nextProps.blocks.block_bingo_sparrow
        });
    }
    // ---- END componentWillReceiveProps() ----
```

```
    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {

        return (
            <div>
                {/* Tags */}
                <div className="port_jervis_tag_1">Port Jervis Yard</
div>
                <div className="crippple_tag">Cripple</div>
                <div className="hall_yard_tag">Campbell Hall Yard</
div>
                <div className="hudson_nysw_tag">NYS&W RR</div>
                <div className="harriman_int_tag">Harriman</div>
                <div className="harriman_int_tag_2">Industrial</div>

                {/* Train Symbols */}
                {/* First Row */}
                <div
className="symbol_bingo_sparrow">{this.state.symbol_bingo_sparrow}</
div>
                <div
className="symbol_sparrow_pa_1">{this.state.symbol_sparrow_pa_1}</div>
                <div
className="symbol_sparrow_pa_2">{this.state.symbol_sparrow_pa_2}</div>
                <div
className="symbol_port_yardEast">{this.state.symbol_port_yardEast}</
div>
                <div
className="symbol_pa_port_1">{this.state.symbol_pa_port_1}</div>
                <div
className="symbol_port_bc_1">{this.state.symbol_port_bc_1}</div>
                <div
className="symbol_pa_bc_2">{this.state.symbol_pa_bc_2}</div>
                <div
className="symbol_bc_ov">{this.state.symbol_bc_ov}</div>
                <div
className="symbol_ov_howells_1">{this.state.symbol_ov_howells_1}</div>
                <div
className="symbol_ov_howells_2">{this.state.symbol_ov_howells_2}</div>
                {/* Second Row */}
                <div
className="symbol_howells_hall">{this.state.symbol_howells_hall}</div>
                <div
className="symbol_hall_yard">{this.state.symbol_hall_yard}</div>
                <div
className="symbol_hall_hudson_1">{this.state.symbol_hall_hudson_1}</
```

```
div>
                <div
className="symbol_hall_hudson_2">{this.state.symbol_hall_hudson_2}</
div>
                <div
className="symbol_hudson_valley">{this.state.symbol_hudson_valley}</
div>
                <div
className="symbol_hudson_nysw">{this.state.symbol_hudson_nysw}</div>
                <div
className="symbol_valley_harriman_1">{this.state.symbol_valley_harrima
n_1}</div>
                <div
className="symbol_valley_harriman_2">{this.state.symbol_valley_harrima
n_2}</div>
                {/* Third Row */}
                <div
className="symbol_harriman_sterling">{this.state.symbol_harriman_sterl
ing}</div>
                <div
className="symbol_harriman_industrial">{this.state.symbol_harriman_ind
ustrial}</div>

                {/* First Line */}
                {/* Sterling to Harriman */}
                <div className="s_sterling_harriman"
style={{background: this.state.harriman_sterling_1}}></div>

                {/* Harriman to Screen */}
                <div className="s_screen_harriman_1"
style={{background: this.state.valley_harriman_1}}></div>
                <div className="s_screen_harriman_2"
style={{background: this.state.valley_harriman_2}}></div>
                <div className="s_harriman_industrial"
style={{background: this.state.harriman_industrial}}></div>


                {/* Second Line */}
                {/* Screen to Central Valley */}
                <div className="s_central_valley_screen_2"
style={{background: this.state.valley_harriman_2}}></div>
                <div className="s_central_valley_screen_1"
style={{background: this.state.valley_harriman_1}}></div>

                {/* Central Valley to Hudson Junction */}
                <div className="s_hudson_valley" style={{background:
this.state.hudson_valley_1}}></div>

                {/* NYS&W Hudson Junction Lead */}
                <div className="s_hudson_nysw" style={{background:
```

```
this.state.hudson_nysw}}></div>

                {/* Hudson Junction to Hall */}
                <div className="s_hall_hudson_2" style={{background:
this.state.hall_hudson_2}}></div>
                <div className="s_hall_hudson_1" style={{background:
this.state.hall_hudson_1}}></div>

                {/* Howells to Hall */}
                <div className="s_howells_hall" style={{background:
this.state.howells_hall_1}}></div>

                {/* Hall Yard Lead */}
                <div className="s_hall_yard" style={{background:
this.state.hall_yard}}></div>

                {/* Howells to Screen */}
                <div className="s_screen_howells_2"
style={{background: this.state.ov_howells_2}}></div>
                <div className="s_screen_howells_1"
style={{background: this.state.ov_howells_1}}></div>


                {/* Third Line */}
                {/* Screen to OV */}
                <div className="s_ov_screen_2" style={{background:
this.state.ov_howells_2}}></div>
                <div className="s_ov_screen_1" style={{background:
this.state.ov_howells_1}}></div>

                {/* OV to BC */}
                <div className="s_ov_bc" style={{background:
this.state.bc_ov_1}}></div>

                {/* BC to Port Jervis */}
                <div className="s_bc_port_1" style={{background:
this.state.port_bc_1}}></div>

                {/* Port Jervis to PA */}
                <div className="s_port_pa_1" style={{background:
this.state.pa_port_1}}></div>
                <div className="s_bc_pa_2" style={{background:
this.state.pa_bc_2}}></div>

                {/* Port Jervis Yard */}
                <div className="s_port_yard_west" style={{background:
this.state.port_yard_west}}></div>
                <div className="s_port_yard_east" style={{background:
this.state.port_yard_east}}></div>
```

```
                {/* PA to Sparrow */}
                <div className="s_sparrow_pa_1" style={{background:
this.state.sparrow_pa_1}}></div>
                <div className="s_sparrow_pa_2" style={{background:
this.state.sparrow_pa_2}}></div>

                {/* Sparrow Cripple */}
                <div className="s_sparrow_cripple" style={{background:
this.state.sparrow_cripple}}></div>

                {/* Sparrow to Screen */}
                <div className="s_screen_sparrow" style={{background:
this.state.bingo_sparrow}}></div>
            </div>
        );
    }
    // ---- END render() ----
}

// Export the tracks to be drawn on the screen
export default SouthernTierTracks;
```

```
/**
 * @file Sparrow.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Sparrow Interlocking
 *
 * Extends the React Component Class and is the UI part of the Sparrow
Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */

// Import React Component
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/sparrow.css';

// Import Images
// Switch Images
import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';
import SW_D_E from '../../../../public/images/SW_D_E.png';
import SW_D_E_Lined from '../../../../public/images/SW_D_E_Lined.png';
import SW_D_E_Occupied from '../../../../public/images/
SW_D_E_Occupied.png';
import SW_D_E_R from '../../../../public/images/SW_D_E_R.png';
import SW_D_E_R_Lined from '../../../../public/images/
SW_D_E_R_Lined.png';
import SW_D_E_R_Occupied from '../../../../public/images/
SW_D_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
```

```
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Sparrow Interlocking
 *
 * This class is a JSX React Component for the Sparrow Interlocking,
this will control all the UI for the component,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Sparrow extends Component {
    /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_1: this.props.status.sw_1,
        sw_3: this.props.status.sw_3,
        // Image File for the switch - Will change depending on route
        sw_1_src: SW_U_E,
        sw_3_src: SW_D_E,
        // Image File for the signals - Will change depending on route
        sig_2w1_src: SIG_W,
        sig_2w2_src: SIG_W,
        sig_2w3_src: SIG_W,
        sig_2e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        tail_cripple: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
```

```
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_1: nextProps.status.sw_1,
            sw_3: nextProps.status.sw_3,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawing();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="sparrow_title">CP SPARROW</div>
                <div className="sparrow_milepost">MP 89.9SR</div>
                {/* West Side Tail Tracks */}
                <div className="sparrow_west" style={{background:
this.state.tail_w}}></div>
                {/* Switches */}
                <div className="sparrow_SW_3"
onClick={this.props.throw_sw_3}><img src={this.state.sw_3_src}/></div>
                <div className="sparrow_SW_1"
onClick={this.props.throw_sw_1}><img src={this.state.sw_1_src}/></div>
                {/* East Side Tail Tracks */}
                <div className="sparrow_cripple" style={{background:
this.state.tail_cripple}}></div>
                <div className="sparrow_1_east" style={{background:
this.state.tail_1_e}}></div>
```

```
                <div className="sparrow_2_east" style={{background:
this.state.tail_2_e}}></div>
                {/* Signals */}
                <div className="sparrow_sig_2w-2"
onClick={this.props.click_sig_2w_2}><img src={this.state.sig_2w2_src}/
></div>
                <div className="sparrow_sig_2w-1"
onClick={this.props.click_sig_2w_1}><img src={this.state.sig_2w1_src}/
></div>
                <div className="sparrow_sig_2w-3"
onClick={this.props.click_sig_2w_3}><img src={this.state.sig_2w3_src}/
></div>
                <div className="sparrow_sig_2e"
onClick={this.props.click_sig_2e}><img src={this.state.sig_2e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----

    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }

        // Loop through all the routes
        for (let i = 0; i < this.state.routes.length; i++) {
            if (this.state.routes[i] === "W_1_1__|__1_bingo_sparrow"
|| this.state.routes[i] === "E_1_1__|__1_sparrow_pa") {
                // Tail Tracks
                this.state.tail_1_e = color;
                this.state.tail_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
```

```
                    this.state.sw_1_src = SW_U_E_Occupied;
                    this.state.sw_3_src = SW_D_E_Occupied;

                    // Signals
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_E_Lined;
                    this.state.sw_3_src = SW_D_E_Lined;

                    // Signals
                    // West Bound
                    if (this.state.routes[i] === "W_1_1__|
__1_bingo_sparrow") {
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2w1_src = SIG_W_Clear;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                    }
                    // East Bound
                    else {
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_2_1__|
__1_bingo_sparrow" || this.state.routes[i] === "E_1_2__|
__2_sparrow_pa") {
                // Tail Tracks
                this.state.tail_2_e = color;
                this.state.tail_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_3_src = SW_D_E_R_Occupied;

                    // Signals
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
```

```
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_3_src = SW_D_E_R_Lined;

                    // Signals
                    // West Bound
                    if (this.state.routes[i] === "W_2_1__|
__1_bingo_sparrow") {
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Clear;
                    }
                    // East Bound
                    else {
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                    }
                }
            }
            else if (this.state.routes[i] === "W_3_1__|
__1_bingo_sparrow" || this.state.routes[i] === "E_1_3__|
__0_sparrow_cripple") {
                // Tail Tracks
                this.state.tail_cripple = color;
                this.state.tail_w = color;

                // The Route Is Occupied
                if (this.state.occupied) {
                    // Switches
                    this.state.sw_1_src = SW_U_E_R_Occupied;
                    this.state.sw_3_src = SW_D_E_Occupied;

                    // Signals
                    this.state.sig_2e_src = SIG_E_Stop;
                    this.state.sig_2w1_src = SIG_W_Stop;
                    this.state.sig_2w2_src = SIG_W_Stop;
                    this.state.sig_2w3_src = SIG_W_Stop;
                }
                // The Route Is NOT Occupied
                else {
                    // Switches
                    this.state.sw_1_src = SW_U_E_R_Lined;
                    this.state.sw_3_src = SW_D_E_Lined;

                    // Signals
```

```
                    // West Bound
                    if (this.state.routes[i] === "W_3_1__|
__1_bingo_sparrow") {
                        this.state.sig_2e_src = SIG_E_Stop;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Clear;
                        this.state.sig_2w3_src = SIG_W_Stop;
                    }
                    // East Bound
                    else {
                        this.state.sig_2e_src = SIG_E_Clear;
                        this.state.sig_2w1_src = SIG_W_Stop;
                        this.state.sig_2w2_src = SIG_W_Stop;
                        this.state.sig_2w3_src = SIG_W_Stop;
                    }
                }
            }
        }
    }
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_1) {
            this.state.sw_1_src = SW_U_E_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_1_src = SW_U_E;
        }

        // Set SW #3
        // SW #3 Reversed
        if (this.state.sw_3) {
            this.state.sw_3_src = SW_D_E_R;
        }
        // SW #3 Normal
        else {
```

```
            this.state.sw_3_src = SW_D_E;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawing() {
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
        this.state.tail_cripple = Empty;
        this.state.tail_w = Empty;

        this.state.sig_2e_src = SIG_E;
        this.state.sig_2w1_src = SIG_W;
        this.state.sig_2w2_src = SIG_W;
        this.state.sig_2w3_src = SIG_W;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Sparrow;
```

```
/**
 * @file Sterling.jsx
 * @author Joey Damico
 * @date September 25, 2019
 * @summary React JSX Component Class that is for Sterling
Interlocking
 *
 * Extends the React Component Class and is the UI part of the
Sterling Interlocking,
 * this class controls all the drawings of routes, and also gives a
visual reprenstation
 * of that status of the interlocking
 */
import React, { Component } from 'react';
// Import CSS style sheet
import '../../../css/Southern_Tier_Line/sterling.css';

// Import Images
// Switch Images
import SW_U_E from '../../../../public/images/SW_U_E.png';
import SW_U_E_Lined from '../../../../public/images/SW_U_E_Lined.png';
import SW_U_E_Occupied from '../../../../public/images/
SW_U_E_Occupied.png';
import SW_U_E_R from '../../../../public/images/SW_U_E_R.png';
import SW_U_E_R_Lined from '../../../../public/images/
SW_U_E_R_Lined.png';
import SW_U_E_R_Occupied from '../../../../public/images/
SW_U_E_R_Occupied.png';

// Signal Images
import SIG_W from '../../../../public/images/SIG_W.png';
import SIG_W_Clear from '../../../../public/images/SIG_W_Clear.png';
import SIG_W_Stop from '../../../../public/images/SIG_W_Stop.png';
import SIG_E from '../../../../public/images/SIG_E.png';
import SIG_E_Clear from '../../../../public/images/SIG_E_Clear.png';
import SIG_E_Stop from '../../../../public/images/SIG_E_Stop.png';

// Color Constants For Drawing Routes
const Empty = '#999999';
const Green = '#75fa4c';
const Red = '#eb3323';


/**
 * The React JSX Component Class for the Hilburn Interlocking
 *
 * This class is a JSX React Component for the Hilburn Interlocking,
this will control all the UI for the component,
 * and the click events that will pass reference between the backend
and the user. This also controls drawing the
```

```
 * route drawings to show if a route(s) is setup in the interlocking
or if the route is occupied
 */
class Sterling extends Component {
     /**
     * State
     * @summary Object that holds the state or status information for
the component
     *
     * This object holds all the information for the interlocking that
is required to display the routes
     * correctly
     *
     * Anything that has "this.props." is passed down from the CTC
interlocking class
     */
    state = {
        // Switch Status
        sw_21: this.props.status.sw_21,
        // Image File for the switch - Will change depending on route
        sw_21_src: SW_U_E,
        // Image File for the signals - Will change depending on route
        sig_2w_src: SIG_W,
        sig_2ws_src: SIG_W,
        sig_1e_src: SIG_E,
        // Colors for tail tracks - Will change depending on route
        tail_w: Empty,
        tail_1_e: Empty,
        tail_2_e: Empty,
        // Information For Interlocking Routes
        occupied: this.props.status.occupied,
        routes: this.props.status.routes
    };

    /**
     * componentWillReceiveProps()
     * @summary Function that updates the state of the component
     *
     * The data that is being changed is passed down from the CTC
classes in the simulation backend
     *
     * @param nextProps, the new data to set the component state too
     */
    componentWillReceiveProps(nextProps){
        this.setState({
            sw_21: nextProps.status.sw_21,
            occupied: nextProps.status.occupied,
            routes: nextProps.status.routes
        });
    }
```

```
    // ---- END componentWillReceiveProps() ----

    /**
     * render()
     * @summary standard React function that draws the interlocking to
the screen
     */
    render() {
        // Clear all the drawings from the interlocking so if a train
clears the route is gone
        this.reset_drawings();
        // Set the switch images based off the state of each crossover
        this.set_switch_img();
        // Draw all the current routes in the interlocking
        this.set_route_drawings();

        // Returns the HTML to draw the interlocking and it's current
state to the screen
        return (
            <div>
                {/* Tags */}
                <div className="sterling_title">CP STERLING</div>
                <div className="sterling_milepost">MP 34.5JS</div>
                {/* West Side Tail Tracks */}
                <div className="sterling_west" style={{background:
this.state.tail_w}}></div>
                {/* Switches */}
                <div className="sterling_SW_21"
onClick={this.props.throw_sw_21}><img src={this.state.sw_21_src}/></
div>
                {/* East Side Tail Tracks */}
                <div className="sterling_1_east" style={{background:
this.state.tail_2_e}}></div>
                <div className="sterling_2_east" style={{background:
this.state.tail_1_e}}></div>
                {/* Signals */}
                <div className="sterling_sig_2ws"
onClick={this.props.click_sig_2ws}><img src={this.state.sig_2ws_src}/
></div>
                <div className="sterling_sig_2w"
onClick={this.props.click_sig_2w}><img src={this.state.sig_2w_src}/></
div>
                <div className="sterling_sig_1e"
onClick={this.props.click_sig_1e}><img src={this.state.sig_1e_src}/></
div>
            </div>
        );
    }
    // ---- END render() ----
```

```
    /**
     * @summary Sets the drawing for the route through the
interlocking
     *
     * Function takes what routes are currently set in the
Interlocking class and displays that route in the UI, the drawing
     * will change depending on if the interlocking is occupied or not
     */
    set_route_drawings() {
        // Setting the color of the tracks depending on if the
interlocking in occupied or not
        let color = null;
        if (this.state.occupied) {
            color = Red;
        }
        else {
            color = Green;
        }
        for (let i = 0; i < this.state.routes.length; i++) {
            // Routes with Track 1 on both the West and East sides
            if (this.state.routes[i] === "W_1_1__|
__1_harriman_sterling" || this.state.routes[i] === "E_1_2__|
__2_sterling_hilburn") {
                // Tail Tracks
                this.state.tail_1_e = color;
                this.state.tail_w = color;

                // Drawing if the interlocking is occupied
                if (this.state.occupied) {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_Occupied;

                    // Signal Images
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_1e_src = SIG_E_Stop;
                }
                // Routing is not occupied
                else {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_Lined;

                    // Signal Images
                    // West Bound
                    if (this.state.routes[i] === "W_1_1__|
__1_harriman_sterling") {
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2w_src = SIG_W_Clear;
                        this.state.sig_1e_src = SIG_E_Stop;
                    }
```

```
                    // East Bound
                    else {
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Clear;
                    }
                }
            }
            // Routes With Track 2 on West Side and Track 1 on East
Side
            else if (this.state.routes[i] === "W_2_1__|
__1_harriman_sterling" || this.state.routes[i] === "E_1_1__|
__1_sterling_sf" ) {
                // Tail Tracks
                this.state.tail_2_e = color;
                this.state.tail_w = color;

                // Drawing if the interlocking is occupied
                if (this.state.occupied) {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_R_Occupied;

                    // Signal Images
                    this.state.sig_2ws_src = SIG_W_Stop;
                    this.state.sig_2w_src = SIG_W_Stop;
                    this.state.sig_1e_src = SIG_E_Stop;
                }
                // Routing that is not occupied
                else {
                    // Switch Image
                    this.state.sw_21_src = SW_U_E_R_Lined;

                    // Signal Images
                    // West Bound Route
                    if (this.state.routes[i] === "W_2_1__|
__1_harriman_sterling") {
                        this.state.sig_2ws_src = SIG_W_Clear;
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Stop;
                    }
                    // East Bound Route
                    else {
                        this.state.sig_2ws_src = SIG_W_Stop;
                        this.state.sig_2w_src = SIG_W_Stop;
                        this.state.sig_1e_src = SIG_E_Clear;
                    }
                }
            }
        }
    }
```

```
    // ---- END set_route_drawings() ----

    /**
     * set_switch_img()
     * @summary Changes image sources for the switches, depending on
switch status
     *
     * This function uses the data passed in through status from the
CTC classes and
     * shows if the switches are reversed or not on the screen, by
changing the image
     * source files, to the correct .png file respectivly
     */
    set_switch_img() {
        // Set SW #1
        // SW #1 Reversed
        if (this.state.sw_21) {
            this.state.sw_21_src = SW_U_E_R;
        }
        // SW #1 Normal
        else {
            this.state.sw_21_src = SW_U_E;
        }
    }
    // ---- END set_switch_img() ----

    /**
     * @summary Function to reset the signal images and track colors
     *
     * This function is need, because if the player was to remove a
route,
     * or when the train clears the interlocking nothing will clear
the route
     * the is displaying on the screen, even if it's gone in the
backend
     */
    reset_drawings() {
        this.state.sig_2w_src = SIG_W;
        this.state.sig_2ws_src = SIG_W;
        this.state.sig_1e_src = SIG_E;
        this.state.tail_w = Empty;
        this.state.tail_1_e = Empty;
        this.state.tail_2_e = Empty;
    }
    //---- END reset_drawings() ----
}

// Export the interlocking to be drawn on the screen
export default Sterling;
```

```
/**
 * @file bergenCounty.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Bergen County Tracks
 *
 * Styles the divs tracks and other tags for the Bergen County Tracks
 */

@charset "UTF-8";
    /* Tags */
    .bt_nysw_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 689px;
        left: 775px;
        font-size: 16px;
        color: #52fff6;
    }

    .hx_line_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 674px;
        left: 1380px;
        font-size: 16px;
        color: #52fff6;
    }
    /* END Tags */

    /* Symbols*/
    .symbol_ridgewood_bt_1 {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 710px;
        left: 430px;
        font-size: 15px;
        font-weight: 700;
        color: #eb3323
    }

    .symbol_ridgewood_bt_2 {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 742px;
        left: 455px;
```

```
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_bt_pascack_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 710px;
    left: 780px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_bt_pascack_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 742px;
    left: 780px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_bt_nysw {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 678px;
    left: 710px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_pascack_hx_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 710px;
    left: 1070px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_pascack_hx_2 {
```

```
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 742px;
    left: 1070px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_hx_laurel_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 710px;
    left: 1280px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_hx_laurel_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 742px;
    left: 1235px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_hx_croxton_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 647px;
    left: 1315px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_hx_croxton_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 678px;
    left: 1315px;
    font-size: 15px;
```

```
    font-weight: 700;
    color: #eb3323
}
/* END Symbols */

/* Tracks */
.b_croxton_1 {
    position: absolute;
    top: 664px;
    left: 1313px;
    height: 7px;
    width: 60px;
    background: #999999;
}

.b_croxton_2 {
    position: absolute;
    top: 695px;
    left: 1313px;
    height: 7px;
    width: 60px;
    background: #999999;
}

.b_laurel_hx_1_west {
    position: absolute;
    top: 726px;
    left: 1275px;
    height: 7px;
    width: 20px;
    background: #999999;
}

.b_laurel_hx_1_diag {
    position: absolute;
    top: 788px;
    left: 1290px;
    height: 7px;
    width: 92px;
    background: #999999;
    transform:
        translateY(-31px)
        translateX(-11px)
        rotate(43deg);
}

.b_laurel_hx_1_east {
    position: absolute;
    top: 788px;
    left: 1356px;
```

```
    height: 7px;
    width: 20px;
    background: #999999;
}

.b_laurel_hx_2_west {
    position: absolute;
    top: 757px;
    left: 1237px;
    height: 7px;
    width: 30px;
    background: #999999;
}

.b_laurel_hx_2_diag {
    position: absolute;
    top: 819px;
    left: 1255px;
    height: 7px;
    width: 94px;
    background: #999999;
    transform:
        translateY(-31px)
        translateX(-3px)
        rotate(43deg);
}

.b_laurel_hx_2_east {
    position: absolute;
    top: 819px;
    left: 1331px;
    height: 7px;
    width: 20px;
    background: #999999;
}

.b_hx_pascack_1 {
    position: absolute;
    top: 726px;
    left: 1059px;
    height: 7px;
    width: 80px;
    background: #999999;
}

.b_hx_pascack_2 {
    position: absolute;
    top: 757px;
    left: 1059px;
    height: 7px;
```

```
    width: 80px;
    background: #999999;
}

.b_pascack_bt_1 {
    position: absolute;
    top: 726px;
    left: 709px;
    height: 7px;
    width: 214px;
    background: #999999;
}

.b_pascack_bt_2 {
    position: absolute;
    top: 757px;
    left: 709px;
    height: 7px;
    width: 214px;
    background: #999999;
}

.b_bt_ridgewood_1_east {
    position: absolute;
    top: 726px;
    left: 450px;
    height: 7px;
    width: 85px;
    background: #999999;
}

.b_bt_ridgewood_1_diag {
    position: absolute;
    top: 788px;
    left: 365px;
    height: 7px;
    width: 104px;
    background: #999999;
    transform:
        translateY(-31px)
        translateX(-6px)
        rotate(-37.5deg);
}

.b_bt_ridgewood_1_west {
    position: absolute;
    top: 788px;
    left: 347px;
    height: 7px;
    width: 25px;
```

```
    background: #999999;
}

.b_bt_ridgewood_2_east {
    position: absolute;
    top: 757px;
    left: 475px;
    height: 7px;
    width: 60px;
    background: #999999;
}

.b_bt_ridgewood_2_diag {
    position: absolute;
    top: 819px;
    left: 390px;
    height: 7px;
    width: 104px;
    background: #999999;
    transform:
        translateY(-31px)
        translateX(-6px)
        rotate(-37.5deg);
}

.b_bt_ridgewood_2_west {
    position: absolute;
    top: 819px;
    left: 347px;
    height: 7px;
    width: 50px;
    background: #999999;
}

.b_nysw {
    position: absolute;
    top: 695px;
    left: 709px;
    height: 7px;
    width: 60px;
    background: #999999;
}

.b_pascack_1 {
    position: absolute;
    top: 660px;
    left: 995px;
    height: 7px;
    width: 60px;
    background: #999999;
```

```
}

.b_pascack_2 {
    position: absolute;
    top: 629px;
    left: 995px;
    height: 7px;
    width: 60px;
    background: #999999;
}
```

```
/**
 * @file bt.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the BT Interlocking
 *
 * Styles the divs tracks and other tags for the BT Interlocking
 */

@charset "UTF-8";

     /* Texts */
     .bt_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 650px;
        left: 612px;
        font-size: 18px;
        color: #14cc00;
    }

    .bt_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 670px;
        left: 600px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .bt_SW_1 {
        position: absolute;
        z-index: 2;
        top: 726px;
        left: 565px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .bt_SW_3 {
        position: absolute;
        z-index: 2;
        top: 726px;
        left: 603px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}

.bt_SW_5 {
    position: absolute;
    z-index: 2;
    top: 695px;
    left: 641px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.bt_1_west {
    position: absolute;
    top: 726px;
    left: 540px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.bt_2_west {
    position: absolute;
    top: 757px;
    left: 540px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.bt_1_east {
    position: absolute;
    top: 726px;
    left: 679px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.bt_2_east {
    position: absolute;
    top: 757px;
    left: 641px;
    height: 7px;
    width: 63px;
    background: #999999;
```

```
}

.bt_3_east {
    position: absolute;
    top: 695px;
    left: 679px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.bt_sig_2w-2 {
    position: absolute;
    top: 677px;
    left: 682px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.bt_sig_2w-1 {
    position: absolute;
    top: 708px;
    left: 682px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.bt_sig_4w {
    position: absolute;
    top: 739px;
    left: 682px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.bt_sig_2e {
    position: absolute;
    top: 730px;
    left: 544px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.bt_sig_4e {
```

```
    position: absolute;
    top: 761px;
    left: 544px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file hx.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the HX Interlocking
 *
 * Styles the divs tracks and other tags for the HX Interlocking
 */

@charset "UTF-8";

    /* Texts */
    .hx_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 620px;
        left: 1207px;
        font-size: 18px;
        color: #14cc00;
    }

    .hx_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 640px;
        left: 1200px;
        font-size: 14px;
        color: white;
    }
    /* END Text */


    /* Switches */
    .hx_SW_1 {
        position: absolute;
        z-index: 2;
        top: 726px;
        left: 1169px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .hx_SW_3 {
        position: absolute;
        z-index: 2;
        top: 695px;
        left: 1207px;
```

```
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.hx_SW_5 {
    position: absolute;
    z-index: 2;
    top: 664px;
    left: 1245px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */


/* Tail Tracks */
.hx_1_west {
    position: absolute;
    top: 726px;
    left: 1144px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hx_2_west {
    position: absolute;
    top: 757px;
    left: 1144px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hx_1_east {
    position: absolute;
    top: 726px;
    left: 1245px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hx_2_east {
    position: absolute;
    top: 757px;
    left: 1207px;
    height: 7px;
```

```
    width: 25px;
    background: #999999;
}

.hx_croxton_1 {
    position: absolute;
    top: 664px;
    left: 1283px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hx_croxton_2 {
    position: absolute;
    top: 695px;
    left: 1283px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */


/* Signals */
.hx_sig_2w-3 {
    position: absolute;
    top: 646px;
    left: 1286px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hx_sig_2w-2 {
    position: absolute;
    top: 677px;
    left: 1286px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hx_sig_2w-1 {
    position: absolute;
    top: 708px;
    left: 1248px;
    height:8px;
    width: 19px;
    cursor: pointer;
```

```
}

.hx_sig_4w {
    position: absolute;
    top: 739px;
    left: 1210px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hx_sig_2e {
    position: absolute;
    top: 730px;
    left: 1148px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hx_sig_4e {
    position: absolute;
    top: 761px;
    left: 1148px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file pascack_jct.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Pascack Junction Interlocking
 *
 * Styles the divs tracks and other tags for the Pascack Junction
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .pascack_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 670px;
        left: 948px;
        font-size: 18px;
        color: #14cc00;
    }

    .pascack_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 690px;
        left: 970px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .pascack_SW_1 {
        position: absolute;
        z-index: 2;
        top: 726px;
        left: 953px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .pascack_SW_3 {
        position: absolute;
        z-index: 2;
        top: 726px;
        left: 991px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}
/* END Switches */


/* Tail Tracks */
.pascack_1_west {
    position: absolute;
    top: 726px;
    left: 928px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.pascack_2_west {
    position: absolute;
    top: 757px;
    left: 928px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.pascack_1_east {
    position: absolute;
    top: 726px;
    left: 1029px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.pascack_2_east {
    position: absolute;
    top: 757px;
    left: 1029px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */


/* Signals */
.pascack_sig_2w {
    position: absolute;
    top: 708px;
    left: 1033px;
```

```
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pascack_sig_4w {
    position: absolute;
    top: 739px;
    left: 1033px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pascack_sig_2e {
    position: absolute;
    top: 730px;
    left: 932px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pascack_sig_4e {
    position: absolute;
    top: 761px;
    left: 932px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file hilburn.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Hilburn Interlocking
 *
 * Styles the divs tracks and other tags for the Hilburn Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .hilburn_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 455px;
        left: 699px;
        font-size: 18px;
        color: #14cc00;
    }

    .hilburn_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 475px;
        left: 715px;
        font-size: 14px;
        color: white;
    }
    /* END Text */


    /* Switches */
    .hilburn_SW_1 {
        position: absolute;
        z-index: 2;
        top: 540px;
        left: 719px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches */


    /* Tail Tracks */
    .hilburn_west {
        position: absolute;
        top: 540px;
```

```
    left: 694px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hilburn_east {
    position: absolute;
    top: 540px;
    left: 757px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hilburn_yard {
    position: absolute;
    top: 571px;
    left: 757px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */


/* Signals */
.hilburn_sig_2w-1 {
    position: absolute;
    top: 522px;
    left: 759px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hilburn_sig_2w-2 {
    position: absolute;
    top: 553px;
    left: 759px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hilburn_sig_2e {
    position: absolute;
    top: 545px;
    left: 696px;
    height:8px;
```

```
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file laurel.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Laurel Interlocking
 *
 * Styles the divs tracks and other tags for the Laurel Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .laurel_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 745px;
        left: 1466px;
        font-size: 18px;
        color: #14cc00;
    }

    .laurel_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 765px;
        left: 1480px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Tail Tracks */
    .m_laurel_3_east {
        position: absolute;
        top: 788px;
        left: 1558px;
        height: 7px;
        width: 25px;
        background: #999999;
    }

    .m_laurel_1_east {
        position: absolute;
        top: 819px;
        left: 1558px;
        height: 7px;
        width: 25px;
        background: #999999;
    }
```

```
.m_laurel_2_east {
    position: absolute;
    top: 850px;
    left: 1558px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.m_laurel_4_east {
    position: absolute;
    top: 881px;
    left: 1558px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.m_laurel_3_center {
    position: absolute;
    top: 788px;
    left: 1444px;
    height: 7px;
    width: 76px;
    background: #999999;
}

.b_laurel_3_west {
    position: absolute;
    top: 788px;
    left: 1381px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.b_laurel_2_west {
    position: absolute;
    top: 819px;
    left: 1356px;
    height: 7px;
    width: 50px;
    background: #999999;
}

.m_laurel_2_west {
    position: absolute;
    top: 850px;
    left: 1406px;
```

```
    height: 7px;
    width: 38px;
    background: #999999;
}

.m_laurel_4_west {
    position: absolute;
    top: 881px;
    left: 1406px;
    height: 7px;
    width: 114px;
    background: #999999;
}
/* END Tail Tracks */

/* Switches */
.laurel_SW_1 {
    position: absolute;
    z-index: 2;
    top: 819px;
    left: 1444px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_SW_3 {
    position: absolute;
    z-index: 2;
    top: 788px;
    left: 1406px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_SW_7 {
    position: absolute;
    z-index: 2;
    top: 819px;
    left: 1482px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_SW_9 {
    position: absolute;
    z-index: 2;
    top: 819px;
```

```
    left: 1520px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_SW_11 {
    position: absolute;
    z-index: 2;
    top: 788px;
    left: 1520px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_SW_13 {
    position: absolute;
    z-index: 2;
    top: 850px;
    left: 1520px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}


/* Signals */
.laurel_sig_10w {
    position: absolute;
    top: 770px;
    left: 1557px;
    height: 10px;
    width: 24px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_sig_2w {
    position: absolute;
    top: 801px;
    left: 1557px;
    height: 10px;
    width: 24px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
```

```
.laurel_sig_4w {
    position: absolute;
    top: 832px;
    left: 1557px;
    height: 10px;
    width: 24px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_sig_8w {
    position: absolute;
    top: 863px;
    left: 1557px;
    height: 10px;
    width: 24px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.laurel_sig_6e {
    position: absolute;
    top: 795px;
    left: 1385px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.laurel_sig_12e {
    position: absolute;
    top: 823px;
    left: 1362px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.laurel_sig_4e {
    position: absolute;
    top: 854px;
    left: 1410px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.laurel_sig_8e {
```

```
    position: absolute;
    top: 885px;
    left: 1410px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file mailLine.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Main Line Tracks
 *
 * Styles the divs tracks and other tags for the Main Line Section of
the pannel
 */

@charset "UTF-8";
    /* Tags */
    .wc_yard_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 470px;
        left: 1225px;
        font-size: 16px;
        color: #52fff6;
    }

    .hilburn_yard_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 575px;
        left: 828px;
        font-size: 16px;
        color: #52fff6;
    }
    /*Second Row*/
    /* END Tags */

    /* Symbols */
    /* First Row */
    .symbol_sterling_hilburn_2 {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 524px;
        left: 610px;
        font-size: 15px;
        font-weight: 700;
        color: #eb3323
    }

    .symbol_hilburn_sf_2 {
        position: absolute;
        overflow: hidden;
```

```
    white-space: nowrap;
    top: 524px;
    left: 830px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_hilburn_yardWest {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 555px;
    left: 785px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_hilburn_yardEast {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 555px;
    left: 900px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_sterling_sf_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 493px;
    left: 750px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_sf_wc_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 524px;
    left: 1180px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
```

```
}

.symbol_sf_wc_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 493px;
    left: 1180px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_wc_ridgewood_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 524px;
    left: 1560px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_wc_ridgewood_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 493px;
    left: 1560px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_wc_ridgewood_3 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 462px;
    left: 1560px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_wc_yard {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
```

```
    top: 462px;
    left: 1330px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}
/* Second Row */
.symbol_ridgewood_suscon_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 834px;
    left: 400px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_ridgewood_suscon_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 865px;
    left: 400px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_suscon_mill_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 834px;
    left: 720px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_suscon_mill_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 865px;
    left: 720px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}
```

```
.symbol_mill_westSecaucus_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 834px;
    left: 1050px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_mill_westSecaucus_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 865px;
    left: 1050px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_westSecaucus_laurel_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 834px;
    left: 1340px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_westSecaucus_laurel_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 865px;
    left: 1340px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_laurel_westEnd_4 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 772px;
```

```
    left: 1590px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_laurel_westEnd_3 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 803px;
    left: 1590px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_laurel_westEnd_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 834px;
    left: 1590px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_laurel_westEnd_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 865px;
    left: 1590px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}
/* END Symbols */


/* Tracks */
.m_sterling_hilburn_2 {
    position: absolute;
    top: 540px;
    left: 599px;
    height: 7px;
    width: 90px;
    background: #999999;
}
```

```
.m_sterling_sf_1 {
    position: absolute;
    top: 509px;
    left: 599px;
    height: 7px;
    width: 353px;
    background: #999999;
}

.m_hilburn_sf {
    position: absolute;
    top: 540px;
    left: 787px;
    height: 7px;
    width: 165px;
    background: #999999;
}

.m_hilburn_yard_east {
    position: absolute;
    top: 571px;
    left: 918px;
    height: 7px;
    width: 35px;
    background: #999999;
}

.m_hilburn_yard_west {
    position: absolute;
    top: 571px;
    left: 787px;
    height: 7px;
    width: 35px;
    background: #999999;
}

.m_sf_wc_1 {
    position: absolute;
    top: 509px;
    left: 1089px;
    height: 7px;
    width: 246px;
    background: #999999;
}

.m_sf_wc_2 {
    position: absolute;
    top: 540px;
    left: 1089px;
```

```
    height: 7px;
    width: 246px;
    background: #999999;
}

.m_wc_yard {
    position: absolute;
    top: 478px;
    left: 1330px;
    height: 7px;
    width: 43px;
    background: #999999;
}

.m_wc_screen_3 {
    position: absolute;
    top: 478px;
    left: 1547px;
    height: 7px;
    width: 125px;
    background: #999999;
}

.m_wc_screen_1 {
    position: absolute;
    top: 509px;
    left: 1547px;
    height: 7px;
    width: 125px;
    background: #999999;
}

.m_wc_screen_2 {
    position: absolute;
    top: 540px;
    left: 1547px;
    height: 7px;
    width: 125px;
    background: #999999;
}

.m_screen_ridgewood_3 {
    position: absolute;
    top: 819px;
    left: 5px;
    height: 7px;
    width: 41px;
    background: #999999;
}
```

```
.m_screen_ridgewood_1 {
    position: absolute;
    top: 850px;
    left: 5px;
    height: 7px;
    width: 41px;
    background: #999999;
}

.m_screen_ridgewood_2 {
    position: absolute;
    top: 881px;
    left: 5px;
    height: 7px;
    width: 41px;
    background: #999999;
}

.m_ridgewood_suscon_1 {
    position: absolute;
    top: 850px;
    left: 347px;
    height: 7px;
    width: 165px;
    background: #999999;
}

.m_ridgewood_suscon_2 {
    position: absolute;
    top: 881px;
    left: 347px;
    height: 7px;
    width: 165px;
    background: #999999;
}

.m_suscon_mill_1 {
    position: absolute;
    top: 850px;
    left: 648px;
    height: 7px;
    width: 216px;
    background: #999999;
}

.m_suscon_mill_2 {
    position: absolute;
    top: 881px;
    left: 648px;
    height: 7px;
```

```
    width: 216px;
    background: #999999;
}

.m_westSecaucus_mill_1 {
    position: absolute;
    top: 850px;
    left: 1000px;
    height: 7px;
    width: 180px;
    background: #999999;
}

.m_westSecaucus_mill_2 {
    position: absolute;
    top: 881px;
    left: 1000px;
    height: 7px;
    width: 180px;
    background: #999999;
}

.m_laurel_westSecaucus_1 {
    position: absolute;
    top: 850px;
    left: 1336px;
    height: 7px;
    width: 65px;
    background: #999999;
}

.m_laurel_westSecaucus_2 {
    position: absolute;
    top: 881px;
    left: 1336px;
    height: 7px;
    width: 65px;
    background: #999999;
}

.m_westEnd_laurel_1 {
    position: absolute;
    top: 850px;
    left: 1588px;
    height: 7px;
    width: 88px;
    background: #999999;
}

.m_westEnd_laurel_2 {
```

```
    position: absolute;
    top: 881px;
    left: 1588px;
    height: 7px;
    width: 88px;
    background: #999999;
}

.m_westEnd_laurel_3 {
    position: absolute;
    top: 819px;
    left: 1588px;
    height: 7px;
    width: 88px;
    background: #999999;
}

.m_westEnd_laurel_4 {
    position: absolute;
    top: 788px;
    left: 1588px;
    height: 7px;
    width: 88px;
    background: #999999;
}
/* END Tracks */
```

```
/**
 * @file mill.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Mill Interlocking
 *
 * Styles the divs tracks and other tags for the Mill Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .mill_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 805px;
        left: 910px;
        font-size: 18px;
        color: #14cc00;
    }

    .mill_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 825px;
        left: 910px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .mill_SW_3 {
        position: absolute;
        z-index: 2;
        top: 850px;
        left: 894px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .mill_SW_1 {
        position: absolute;
        z-index: 2;
        top: 850px;
        left: 932px;
        height: 38px;
        width: 38px;
```

```
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.mill_1_west {
    position: absolute;
    top: 850px;
    left: 970px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.mill_2_west {
    position: absolute;
    top: 881px;
    left: 970px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.mill_1_east {
    position: absolute;
    top: 850px;
    left: 869px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.mill_2_east {
    position: absolute;
    top: 881px;
    left: 869px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.mill_sig_2w {
    position: absolute;
    top: 832px;
    left: 975px;
    height: 8px;
    width: 19px;
    cursor: pointer;
```

```
}

.mill_sig_4w {
    position: absolute;
    top: 863px;
    left: 975px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.mill_sig_2e {
    position: absolute;
    top: 854px;
    left: 870px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.mill_sig_4e {
    position: absolute;
    top: 885px;
    left: 870px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file ridgewood_jct.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Ridgewood Junction Interlocking
 *
 * Styles the divs tracks and other tags for the Ridgewood Junction
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .ridgewood_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 745px;
        left: 85px;
        font-size: 18px;
        color: #14cc00;
    }

    .ridgewood_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 765px;
        left: 165px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Tail Tracks */
    .m_ridgewood_3_west {
        position: absolute;
        top: 819px;
        left: 51px;
        height: 7px;
        width: 25px;
        background: #999999;
    }

    .m_ridgewood_1_west {
        position: absolute;
        top: 850px;
        left: 51px;
        height: 7px;
        width: 25px;
        background: #999999;
```

```
}

.m_ridgewood_2_west {
    position: absolute;
    top: 881px;
    left: 51px;
    height: 7px;
    width: 63px;
    background: #999999;
}

     .b_ridgewood_1 {
    position: absolute;
    top: 788px;
    left: 266px;
    height: 7px;
    width: 76px;
    background: #999999;
}

.b_ridgewood_1_Diag {
    position: absolute;
    top: 789px;
    left: 220px;
    height: 7px;
    width: 55px;
    background: #999999;
    transform:
        translateY(15px)
        translateX(0px)
        rotate(-37deg);
}

     .b_ridgewood_2 {
    position: absolute;
    top: 819px;
    left: 304px;
    height: 7px;
    width: 38px;
    background: #999999;
}

.m_ridgewood_3_center {
    position: absolute;
    top: 819px;
    left: 114px;
    height: 7px;
    width: 76px;
    background: #999999;
}
```

```
     .m_ridgewood_1_center {
    position: absolute;
    top: 850px;
    left: 228px;
    height: 7px;
    width: 38px;
    background: #999999;
}

.m_ridgewood_1_east {
    position: absolute;
    top: 850px;
    left: 304px;
    height: 7px;
    width: 38px;
    background: #999999;
}

.m_ridgewood_2_east {
    position: absolute;
    top: 881px;
    left: 190px;
    height: 7px;
    width: 152px;
    background: #999999;
}
/* END Tail Tracks */

/* Switches */
     .ridgewood_1 {
    position: absolute;
    z-index: 2;
    top: 819px;
    left: 76px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.ridgewood_3 {
    position: absolute;
    z-index: 2;
    top: 850px;
    left: 114px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
```

```
.ridgewood_5 {
    position: absolute;
    z-index: 2;
    top: 850px;
    left: 152px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.ridgewood_7{
    position: absolute;
    z-index: 2;
    top: 819px;
    left: 190px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.ridgewood_9 {
    position: absolute;
    z-index: 2;
    top: 819px;
    left: 266px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Signals */
.ridgewood_sig_6w {
    position: absolute;
    top: 770px;
    left: 320px;
    height:38px;
    width: 19px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.ridgewood_sig_2w-2 {
    position: absolute;
    top: 801px;
    left: 320px;
    height:38px;
    width: 19px;
    height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}

.ridgewood_sig_2w-1 {
    position: absolute;
    top: 832px;
    left: 320px;
    height:38px;
    width: 19px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.ridgewood_sig_4w {
    position: absolute;
    top: 863px;
    left: 320px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.ridgewood_sig_6e {
    position: absolute;
    top: 823px;
    left: 55px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.ridgewood_sig_2e {
    position: absolute;
    top: 854px;
    left: 55px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.ridgewood_sig_4e {
    position: absolute;
    top: 885px;
    left: 55px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}
```

```
/* END Signals */
```

```
/**
 * @file sf.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the SF Interlocking
 *
 * Styles the divs tracks and other tags for the SF Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .sf_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 455px;
        left: 1012px;
        font-size: 18px;
        color: #14cc00;
    }

    .sf_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 475px;
        left: 1000px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .sf_SW_1 {
        position: absolute;
        z-index: 2;
        top: 540px;
        left: 983px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .sf_SW_3 {
        position: absolute;
        z-index: 2;
        top: 509px;
        left: 1021px;
        height: 38px;
        width: 38px;
```

```
    cursor: pointer;
}

.sf_SW_5 {
    position: absolute;
    z-index: 2;
    top: 509px;
    left: 1149px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.sf_SW_7 {
    position: absolute;
    z-index: 2;
    top: 509px;
    left: 1187px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.sf_1_west {
    position: absolute;
    top: 509px;
    left: 958px;
    height: 7px;
    width: 63px;
    background: #999999;
}

.sf_2_west {
    position: absolute;
    top: 540px;
    left: 958px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sf_yard {
    position: absolute;
    top: 571px;
    left: 958px;
    height: 7px;
    width: 25px;
    background: #999999;
```

```
}

.sf_1_center_west {
    position: absolute;
    top: 509px;
    left: 1059px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sf_2_center_west {
    position: absolute;
    top: 540px;
    left: 1059px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.sf_sig_2e {
    position: absolute;
    top: 514px;
    left: 961px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sf_sig_4e-1 {
    position: absolute;
    top: 545px;
    left: 961px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sf_sig_4e-2 {
    position: absolute;
    top: 576px;
    left: 961px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sf_sig_2w {
```

```
    position: absolute;
    top: 491px;
    left: 1064px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sf_sig_4w {
    position: absolute;
    top: 522px;
    left: 1064px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
```

```
/**
 * @file suscon.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Suscon Interlocking
 *
 * Styles the divs tracks and other tags for the Suscon Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .suscon_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 805px;
        left: 546px;
        font-size: 18px;
        color: #14cc00;
    }

    .suscon_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 825px;
        left: 559px;
        font-size: 14px;
        color: white;
    }
    /* END Text */


    /* Switches */
    .suscon_SW_3 {
        position: absolute;
        z-index: 2;
        top: 850px;
        left: 542px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .suscon_SW_1 {
        position: absolute;
        z-index: 2;
        top: 850px;
        left: 580px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.suscon_1_west {
    position: absolute;
    top: 850px;
    left: 517px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.suscon_2_west {
    position: absolute;
    top: 881px;
    left: 517px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.suscon_1_east {
    position: absolute;
    top: 850px;
    left: 618px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.suscon_2_east {
    position: absolute;
    top: 881px;
    left: 618px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.suscon_sig_2w {
    position: absolute;
    top: 832px;
    left: 620px;
    height: 8px;
    width: 19px;
```

```
    cursor: pointer;
}

.suscon_sig_2e {
    position: absolute;
    top: 854px;
    left: 520px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.suscon_sig_4w {
    position: absolute;
    top: 863px;
    left: 620px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.suscon_sig_4e {
    position: absolute;
    top: 885px;
    left: 520px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file wc.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the WC Interlocking
 *
 * Styles the divs tracks and other tags for the WC Interlocking
 */

@charset "UTF-8";

    /* Texts */
    .wc_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 430px;
        left: 1430px;
        font-size: 18px;
        color: #14cc00;
    }

    .wc_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 450px;
        left: 1420px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .wc_SW_1 {
        position: absolute;
        z-index: 2;
        top: 509px;
        left: 1365px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .wc_SW_3 {
        position: absolute;
        z-index: 2;
        top: 478px;
        left: 1403px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}

.wc_SW_5 {
    position: absolute;
    z-index: 2;
    top: 509px;
    left: 1441px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}

.wc_SW_7 {
    position: absolute;
    z-index: 2;
    top: 478px;
    left: 1479px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.wc_yard {
    position: absolute;
    top: 478px;
    left: 1378px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.wc_1_west {
    position: absolute;
    top: 509px;
    left: 1340px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.wc_2_west {
    position: absolute;
    top: 540px;
    left: 1340px;
    height: 7px;
    width: 25px;
```

```
    background: #999999;
}

.wc_2_center {
    position: absolute;
    top: 540px;
    left: 1403px;
    height: 7px;
    width: 38px;
    background: #999999;
}

.wc_3_east {
    position: absolute;
    top: 478px;
    left: 1517px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.wc_1_east {
    position: absolute;
    top: 509px;
    left: 1517px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.wc_2_east {
    position: absolute;
    top: 540px;
    left: 1479px;
    height: 7px;
    width: 63px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.wc_sig_2e-2 {
    position: absolute;
    top: 483px;
    left: 1381px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
```

```
.wc_sig_2e-1 {
    position: absolute;
    top: 514px;
    left: 1344px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.wc_sig_4e {
    position: absolute;
    top: 545px;
    left: 1344px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.wc_sig_2w-2 {
    position: absolute;
    top: 460px;
    left: 1520px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.wc_sig_2w-1 {
    position: absolute;
    top: 491px;
    left: 1520px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.wc_sig_4w {
    position: absolute;
    top: 522px;
    left: 1520px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* Signals */
```

```
/**
 * @file west_secaucus.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the West Secaucus Interlocking
 *
 * Styles the divs tracks and other tags for the West Secaucus
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .westSecaucus_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 805px;
        left: 1155px;
        font-size: 18px;
        color: #14cc00;
    }

    .westSecaucus_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 825px;
        left: 1235px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .westSecaucus_SW_1 {
        position: absolute;
        z-index: 2;
        top: 850px;
        left: 1268px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .m_westSecaucus_bridge {
        position: absolute;
        top: 850px;
        left: 1248px;
        height: 7px;
        width: 20px;
```

```
    background: #999999;
}

.westSecaucus_SW_3 {
    position: absolute;
    z-index: 2;
    top: 850px;
    left: 1210px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.m_westSecaucus_1_west {
    position: absolute;
    top: 850px;
    left: 1185px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.m_westSecaucus_2_west {
    position: absolute;
    top: 881px;
    left: 1185px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.m_westSecaucus_1_east {
    position: absolute;
    top: 850px;
    left: 1306px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.m_westSecaucus_2_east {
    position: absolute;
    top: 881px;
    left: 1306px;
    height: 7px;
    width: 25px;
    background: #999999;
}
```

```
/* END Tail Tracks */

/* SIGNALS */
.westSecaucus_sig_2w {
    position: absolute;
    top: 832px;
    left: 1307px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.westSecaucus_sig_4w {
    position: absolute;
    top: 863px;
    left: 1307px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.westSecaucus_sig_2e {
    position: absolute;
    top: 854px;
    left: 1190px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}

.westSecaucus_sig_4e {
    position: absolute;
    top: 885px;
    left: 1190px;
    height: 8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file bc.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP BC Interlocking
 *
 * Styles the divs tracks and other tags for the CP BC Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .bc_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 180px;
        left: 1008px;
        font-size: 18px;
        color: #14cc00;
    }

    .bc_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 200px;
        left: 1000px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .bc_SW_1 {
        position: absolute;
        z-index: 2;
        top: 230px;
        left: 1018px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches*/

    /* Tail Tracks */
    .bc_1_west {
        position: absolute;
        top: 230px;
        left: 993px;
        height: 7px;
```

```
    width: 25px;
    background: #999999;
}

.bc_2_west {
    position: absolute;
    top: 261px;
    left: 993px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.bc_east {
    position: absolute;
    top: 261px;
    left: 1056px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.bc_sig_2e {
    position: absolute;
    top: 235px;
    left: 995px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.bc_sig_4e {
    position: absolute;
    top: 266px;
    left: 995px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.bc_sig_2w {
    position: absolute;
    top: 243px;
    left: 1058px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
```

```
/* END Signals */
```

```
/**
 * @file centralValley.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Central Valley Interlocking
 *
 * Styles the divs tracks and other tags for the CP Central Valley
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .valley_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 320px;
        left: 1360px;
        font-size: 18px;
        color: #14cc00;
    }

    .valley_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 340px;
        left: 1420px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .valley_SW_21 {
        position: absolute;
        z-index: 2;
        top: 369px;
        left: 1432px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches */

    /* Tail Tracks */
    .valley_west {
        position: absolute;
        top: 400px;
        left: 1407px;
```

```
    height: 7px;
    width: 25px;
    background: #999999;
}

.valley_2_east {
    position: absolute;
    top: 369px;
    left: 1470px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.valley_1_east {
    position: absolute;
    top: 400px;
    left: 1470px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.valley_sig_1e {
    position: absolute;
    top: 405px;
    left: 1410px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.valley_sig_2w {
    position: absolute;
    top: 351px;
    left: 1472px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.valley_sig_1w {
    position: absolute;
    top: 382px;
    left: 1472px;
    height:8px;
    width: 19px;
    cursor: pointer;
```

```
}
/* END Signals */
```

```
/**
 * @file hall.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Hall Interlocking
 *
 * Styles the divs tracks and other tags for the CP Hall Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .hall_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 320px;
        left: 618px;
        font-size: 18px;
        color: #14cc00;
    }

    .hall_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 340px;
        left: 625px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .hall_SW_1 {
        position: absolute;
        z-index: 2;
        top: 369px;
        left: 634px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches */

    /* Tail Tracks */
    .hall_yard {
        position: absolute;
        top: 369px;
        left: 609px;
        height: 7px;
```

```
    width: 25px;
    background: #999999;
}

.hall_west {
    position: absolute;
    top: 400px;
    left: 609px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hall_2_east {
    position: absolute;
    top: 369px;
    left: 672px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hall_1_east {
    position: absolute;
    top: 400px;
    left: 672px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.hall_sig_4w {
    position: absolute;
    top: 351px;
    left: 675px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hall_sig_2w {
    position: absolute;
    top: 382px;
    left: 675px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
```

```
.hall_sig_4e {
    position: absolute;
    top: 374px;
    left: 611px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hall_sig_2e {
    position: absolute;
    top: 405px;
    left: 611px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file harriman.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Harriman Interlocking
 *
 * Styles the divs tracks and other tags for the CP Harriman
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .harriman_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 460px;
        left: 115px;
        font-size: 18px;
        color: #14cc00;
    }

    .harriman_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 480px;
        left: 148px;
        font-size: 14px;
        color: white;
    }
    /* END Text */


    /* Switches */
    .harriman_SW_21 {
        position: absolute;
        z-index: 2;
        top: 509px;
        left: 135px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .harriman_SW_32 {
        position: absolute;
        z-index: 2;
        top: 540px;
        left: 173px;
```

```
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */


/* Tail Tracks */
.harriman_1_west {
    position: absolute;
    top: 540px;
    left: 110px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.harriman_2_west {
    position: absolute;
    top: 509px;
    left: 110px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.harriman_industrial {
    position: absolute;
    top: 571px;
    left: 148px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.harriman_1_east {
    position: absolute;
    top: 540px;
    left: 211px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */


/* Signals */
.harriman_sig_2e {
    position: absolute;
    top: 514px;
```

```
    left: 112px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.harriman_sig_1e {
    position: absolute;
    top: 545px;
    left: 112px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.harriman_sig_3e {
    position: absolute;
    top: 576px;
    left: 152px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.harriman_sig_1w {
    position: absolute;
    top: 522px;
    left: 214px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
@charset "UTF-8";

    /* Texts */
    .howells_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 320px;
        left: 295px;
        font-size: 18px;
        color: #14cc00;
    }

    .howells_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 340px;
        left: 322px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .howells_SW_3 {
        position: absolute;
        z-index: 2;
        top: 369px;
        left: 335px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches */

    /* Tail Tracks */
    .howells_2_west {
        position: absolute;
        top: 369px;
        left: 310px;
        height: 7px;
        width: 25px;
        background: #999999;
    }

    .howells_1_west {
        position: absolute;
        top: 400px;
        left: 310px;
```

```
    height: 7px;
    width: 25px;
    background: #999999;
}

.howells_east {
    position: absolute;
    top: 400px;
    left: 373px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Signals */

/* Signals */
.howells_sig_2es {
    position: absolute;
    top: 374px;
    left: 313px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.howells_sig_2e {
    position: absolute;
    top: 405px;
    left: 313px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.howells_sig_2w {
    position: absolute;
    top: 382px;
    left: 376px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file hudsonJunction.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Hudson Junction Interlocking
 *
 * Styles the divs tracks and other tags for the CP Hudson Junction
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .hudson_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 320px;
        left: 835px;
        font-size: 18px;
        color: #14cc00;
    }

    .hudson_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 340px;
        left: 900px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .hudson_SW_1 {
        position: absolute;
        z-index: 2;
        top: 369px;
        left: 896px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .hudson_SW_3 {
        position: absolute;
        z-index: 2;
        top: 400px;
        left: 934px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.hudson_east {
    position: absolute;
    top: 400px;
    left: 972px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hudson_nysw {
    position: absolute;
    top: 431px;
    left: 972px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hudson_2_west {
    position: absolute;
    top: 369px;
    left: 871px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.hudson_1_west {
    position: absolute;
    top: 400px;
    left: 871px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.hudson_sig_2es {
    position: absolute;
    top: 374px;
    left: 873px;
    height:8px;
    width: 19px;
```

```
    cursor: pointer;
}

.hudson_sig_2e {
    position: absolute;
    top: 405px;
    left: 873px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hudson_sig_2w {
    position: absolute;
    top: 382px;
    left: 973px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.hudson_sig_2ws {
    position: absolute;
    top: 413px;
    left: 973px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file ov.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP OV Interlocking
 *
 * Styles the divs tracks and other tags for the CP OV Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .ov_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 180px;
        left: 1412px;
        font-size: 18px;
        color: #14cc00;
    }

    .ov_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 200px;
        left: 1405px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .ov_SW_1 {
        position: absolute;
        z-index: 2;
        top: 230px;
        left: 1432px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches */

    /* Tail Tracks */
    .ov_west {
        position: absolute;
        top: 261px;
        left: 1382px;
        height: 7px;
```

```
    width: 50px;
    background: #999999;
}

.ov_2_east {
    position: absolute;
    top: 230px;
    left: 1470px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.ov_1_east {
    position: absolute;
    top: 261px;
    left: 1470px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.ov_sig_2e {
    position: absolute;
    top: 266px;
    left: 1385px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.ov_sig_2ws {
    position: absolute;
    top: 212px;
    left: 1472px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.ov_sig_2w {
    position: absolute;
    top: 243px;
    left: 1472px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
```

```
/* END Signals */
```

```
/**
 * @file pa.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP PA Interlocking
 *
 * Styles the divs tracks and other tags for the CP PA Interlocking
 */

@charset "UTF-8";

    /* Texts */
    .pa_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 145px;
        left: 505px;
        font-size: 18px;
        color: #14cc00;
    }

    .pa_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 165px;
        left: 495px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .pa_SW_1 {
        position: absolute;
        z-index: 2;
        top: 199px;
        left: 547px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .pa_SW_3 {
        position: absolute;
        z-index: 2;
        top: 230px;
        left: 509px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}

.pa_SW_5 {
    position: absolute;
    z-index: 2;
    top: 261px;
    left: 471px;
    height: 38px;
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.pa_1_west {
    position: absolute;
    top: 230px;
    left: 446px;
    height: 7px;
    width: 63px;
    background: #999999;
}

.pa_2_west {
    position: absolute;
    top: 261px;
    left: 446px;
    height: 7px;
    width: 63px;
    background: #999999;
}

.pa_yard {
    position: absolute;
    top: 199px;
    left: 585px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.pa_1_east {
    position: absolute;
    top: 230px;
    left: 585px;
    height: 7px;
    width: 25px;
    background: #999999;
```

```
}

.pa_2_east {
    position: absolute;
    top: 261px;
    left: 547px;
    height: 7px;
    width: 63px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.pa_sig_2w-2 {
    position: absolute;
    top: 181px;
    left: 587px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pa_sig_2w-1 {
    position: absolute;
    top: 212px;
    left: 587px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pa_sig_4w {
    position: absolute;
    top: 243px;
    left: 587px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pa_sig_6w {
    position: absolute;
    top: 274px;
    left: 511px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pa_sig_2e {
```

```
    position: absolute;
    top: 235px;
    left: 450px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pa_sig_4e {
    position: absolute;
    top: 266px;
    left: 450px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.pa_sig_6e {
    position: absolute;
    top: 297px;
    left: 450px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file port.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Port Interlocking
 *
 * Styles the divs tracks and other tags for the CP Port Interlocking
 */

@charset "UTF-8";

    /* Texts */
    .port_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 145px;
        left: 752px;
        font-size: 18px;
        color: #14cc00;
    }

    .port_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 165px;
        left: 756px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .port_SW_1 {
        position: absolute;
        z-index: 2;
        top: 199px;
        left: 770px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches */

    /* Tail Tracks */
    .port_yard {
        position: absolute;
        top: 199px;
        left: 745px;
```

```
    height: 7px;
    width: 25px;
    background: #999999;
}

.port_west {
    position: absolute;
    top: 230px;
    left: 745px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.port_east {
    position: absolute;
    top: 230px;
    left: 808px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.port_sig_2w {
    position: absolute;
    top: 212px;
    left: 808px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.port_sig_2e-2 {
    position: absolute;
    top: 204px;
    left: 748px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.port_sig_2e-1 {
    position: absolute;
    top: 235px;
    left: 748px;
    height:8px;
    width: 19px;
    cursor: pointer;
```

```
}
/* END Signals */
```

```
/**
 * @file southernTier.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the Southern Tier Tracks
 *
 * Styles the divs tracks and other tags for the Southern Tier Section
of the pannel
 */
@charset "UTF-8";

    /* Tags */
    .port_jervis_tag_1 {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 150px;
        left: 625px;
        font-size: 16px;
        color: #52fff6;
    }

    .crippple_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 180px;
        left: 300px;
        font-size: 16px;
        color: #52fff6;
    }

    .hall_yard_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 362px;
        left: 420px;
        font-size: 16px;
        color: #52fff6;
    }

    .hudson_nysw_tag {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 425px;
        left: 1090px;
        font-size: 16px;
        color: #52fff6;
```

```
}

.harriman_int_tag {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 555px;
    left: 24px;
    font-size: 16px;
    color: #52fff6;
}

.harriman_int_tag_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 572px;
    left: 24px;
    font-size: 16px;
    color: #52fff6;
}
/* END Tags */

/* Train Symbols */
/* First Row */
.symbol_bingo_sparrow {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 212px;
    left: 20px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_sparrow_pa_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 212px;
    left: 320px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_sparrow_pa_2 {
    position: absolute;
    overflow: hidden;
```

```
    white-space: nowrap;
    top: 244px;
    left: 300px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_pa_port_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 212px;
    left: 660px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_port_yardEast {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 180px;
    left: 710px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_pa_bc_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 244px;
    left: 800px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_port_bc_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 212px;
    left: 895px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
```

```
}

.symbol_bc_ov {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 244px;
    left: 1210px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_ov_howells_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 244px;
    left: 1580px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_ov_howells_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 212px;
    left: 1580px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

/* Second Row*/
.symbol_howells_hall {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 382px;
    left: 490px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_hall_yard {
    position: absolute;
    overflow: hidden;
```

```
    white-space: nowrap;
    top: 352px;
    left: 565px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_hall_hudson_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 382px;
    left: 770px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_hall_hudson_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 352px;
    left: 770px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_hudson_nysw {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 415px;
    left: 1020px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_hudson_valley {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 382px;
    left: 1190px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
```

```
}

.symbol_valley_harriman_1 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 382px;
    left: 1580px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

.symbol_valley_harriman_2 {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 352px;
    left: 1580px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323;
}

/* Third Row */
.symbol_harriman_sterling {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 520px;
    left: 355px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}

.symbol_harriman_industrial {
    position: absolute;
    overflow: hidden;
    white-space: nowrap;
    top: 590px;
    left: 90px;
    font-size: 15px;
    font-weight: 700;
    color: #eb3323
}
/* END Train Symbols */


/* Tracks */
```

```
.s_ov_screen_2 {
    position: absolute;
    top: 230px;
    left: 1500px;
    height: 7px;
    width: 170px;
    background: #999999;
}

.s_ov_screen_1 {
    position: absolute;
    top: 261px;
    left: 1500px;
    height: 7px;
    width: 170px;
    background: #999999;
}

.s_ov_bc {
    position: absolute;
    top: 261px;
    left: 1086px;
    height: 7px;
    width: 291px;
    background: #999999;
}

.s_bc_port_1 {
    position: absolute;
    top: 230px;
    left: 838px;
    height: 7px;
    width: 150px;
    background: #999999;
}

.s_port_pa_1 {
    position: absolute;
    top: 230px;
    left: 615px;
    height: 7px;
    width: 125px;
    background: #999999;
}

.s_bc_pa_2 {
    position: absolute;
    top: 261px;
    left: 615px;
    height: 7px;
```

```
    width: 373px;
    background: #999999;
}

.s_port_yard_west {
    position: absolute;
    top: 199px;
    left: 615px;
    height: 7px;
    width: 30px;
    background: #999999;
}

.s_port_yard_east {
    position: absolute;
    top: 199px;
    left: 710px;
    height: 7px;
    width: 30px;
    background: #999999;
}

.s_sparrow_pa_1 {
    position: absolute;
    top: 230px;
    left: 241px;
    height: 7px;
    width: 200px;
    background: #999999;
}

.s_sparrow_pa_2 {
    position: absolute;
    top: 261px;
    left: 203px;
    height: 7px;
    width: 238px;
    background: #999999;
}

.s_sparrow_cripple {
    position: absolute;
    top: 199px;
    left: 241px;
    height: 7px;
    width: 50px;
    background: #999999;
}

.s_screen_sparrow {
```

```
    position: absolute;
    top: 230px;
    left: 5px;
    height: 7px;
    width: 100px;
    background: #999999;
}

.s_screen_harriman_1 {
    position: absolute;
    top: 540px;
    left: 5px;
    height: 7px;
    width: 100px;
    background: #999999;
}

.s_screen_harriman_2 {
    position: absolute;
    top: 509px;
    left: 5px;
    height: 7px;
    width: 100px;
    background: #999999;
}

.s_harriman_industrial {
    position: absolute;
    top: 571px;
    left: 93px;
    height: 7px;
    width: 50px;
    background: #999999;
}

.s_sterling_harriman {
    position: absolute;
    top: 540px;
    left: 241px;
    height: 7px;
    width: 260px;
    background: #999999;
}

.s_central_valley_screen_2 {
    position: absolute;
    top: 369px;
    left: 1500px;
    height: 7px;
    width: 170px;
```

```
    background: #999999;
}

.s_central_valley_screen_1 {
    position: absolute;
    top: 400px;
    left: 1500px;
    height: 7px;
    width: 170px;
    background: #999999;
}

.s_hudson_valley {
    position: absolute;
    top: 400px;
    left: 1002px;
    height: 7px;
    width: 400px;
    background: #999999;
}

.s_hudson_nysw {
    position: absolute;
    top: 431px;
    left: 1002px;
    height: 7px;
    width: 80px;
    background: #999999;
}

.s_hall_hudson_2 {
    position: absolute;
    top: 369px;
    left: 702px;
    height: 7px;
    width: 164px;
    background: #999999;
}

.s_hall_hudson_1 {
    position: absolute;
    top: 400px;
    left: 702px;
    height: 7px;
    width: 164px;
    background: #999999;
}

.s_howells_hall {
    position: absolute;
```

```
    top: 400px;
    left: 404px;
    height: 7px;
    width: 200px;
    background: #999999;
}

.s_hall_yard {
    position: absolute;
    top: 369px;
    left: 553px;
    height: 7px;
    width: 50px;
    background: #999999;
}

.s_screen_howells_2 {
    position: absolute;
    top: 369px;
    left: 5px;
    height: 7px;
    width: 300px;
    background: #999999;
}

.s_screen_howells_1 {
    position: absolute;
    top: 400px;
    left: 5px;
    height: 7px;
    width: 300px;
    background: #999999;
}
```

```
/**
 * @file sparrow.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Sparrow Interlocking
 *
 * Styles the divs tracks and other tags for the CP Sparrow
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .sparrow_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 145px;
        left: 120px;
        font-size: 18px;
        color: #14cc00;
    }

    .sparrow_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 165px;
        left: 145px;
        font-size: 14px;
        color: white;
    }
    /* END Text */

    /* Switches */
    .sparrow_SW_1 {
        position: absolute;
        z-index: 2;
        top: 199px;
        left: 173px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }

    .sparrow_SW_3 {
        position: absolute;
        z-index: 2;
        top: 230px;
        left: 135px;
        height: 38px;
```

```
    width: 38px;
    cursor: pointer;
}
/* END Switches */

/* Tail Tracks */
.sparrow_west {
    position: absolute;
    top: 230px;
    left: 110px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sparrow_cripple {
    position: absolute;
    top: 199px;
    left: 211px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sparrow_1_east {
    position: absolute;
    top: 230px;
    left: 211px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sparrow_2_east {
    position: absolute;
    top: 261px;
    left: 173px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */

/* Signals */
.sparrow_sig_2w-2 {
    position: absolute;
    top: 181px;
    left: 213px;
    height:8px;
    width: 19px;
```

```
    cursor: pointer;
}

.sparrow_sig_2w-1 {
    position: absolute;
    top: 212px;
    left: 213px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sparrow_sig_2w-3 {
    position: absolute;
    top: 243px;
    left: 176px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sparrow_sig_2e {
    position: absolute;
    top: 235px;
    left: 113px;
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

```
/**
 * @file sterling.css
 * @author Joey Damico
 * @date September 25, 2019
 * @brief CSS Stylesheet for the CP Sterling Interlocking
 *
 * Styles the divs tracks and other tags for the CP Sterling
Interlocking
 */

@charset "UTF-8";
    /* Texts */
    .sterling_title {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 455px;
        left: 490px;
        font-size: 18px;
        color: #14cc00;
    }

    .sterling_milepost {
        position: absolute;
        overflow: hidden;
        white-space: nowrap;
        top: 475px;
        left: 520px;
        font-size: 14px;
        color: white;
    }
    /* END Text */


    /* Switches */
    .sterling_SW_21 {
        position: absolute;
        z-index: 2;
        top: 509px;
        left: 531px;
        height: 38px;
        width: 38px;
        cursor: pointer;
    }
    /* END Switches*/


    /* Tail Tracks */
    .sterling_west {
        position: absolute;
```

```
    top: 540px;
    left: 506px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sterling_2_east {
    position: absolute;
    top: 540px;
    left: 569px;
    height: 7px;
    width: 25px;
    background: #999999;
}

.sterling_1_east {
    position: absolute;
    top: 509px;
    left: 569px;
    height: 7px;
    width: 25px;
    background: #999999;
}
/* END Tail Tracks */


/* Signals */
.sterling_sig_2ws {
    position: absolute;
    top: 491px;
    left: 572px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sterling_sig_2w {
    position: absolute;
    top: 522px;
    left: 572px;
    height:8px;
    width: 19px;
    cursor: pointer;
}

.sterling_sig_1e {
    position: absolute;
    top: 545px;
    left: 508px;
```

```
    height:8px;
    width: 19px;
    cursor: pointer;
}
/* END Signals */
```

Part IV: Summary & Conclusion

When everything was done, this project allowed me to learn and grown on many fronts, first was just on how people create web applications, and how they work. It gave me a much better understanding how the internet works. And now I find myself thinking about how each website I visit was designed and if I was tasked with creating a similar app, how would I create it. This I hope will be helpful when job hunting, since now I have some serious experience developing a web application, I'll be confident to talk about them on Interviews. Another thing this project taught me was time management. The spring semester was quite busy for me, so although I worked on it, I didn't go deep into the program until the summer. I had to make sure I kept on working on the project, even though It was easy for the time to get away from you. It happened a few times when I said I would take a week off for various reasons, and then in a blink of an eye it was 2 or 3 weeks later.

And the last major thing working on this project taught me is that it's okay to admit that you are going in the wrong direction, and completely start over. We reached the end of the semester, the program I had was nowhere near good enough and had a very poor design. It also wasn't using any sort of framework, which would make it very difficult to upload the project to a webserver. I got to the point where I didn't want to work on the program, because it was so poorly designed. So, I decided to completely scrap what I had and started again. Which was probably the best thing I could have done. Allowing me do start over again made the design flow a lot easier, and the code seemed to naturally write it's self once I had the correct design.

If I had to do this program all over again, I would tell myself to do more work earlier on, to let me do less work at the end of the time working on it. Sleep is good, and towards the end of this project I definitely didn't get enough of it.

Part X: Work Log

SEE NEXT PAGE

March 21, 2019:
    - Finished drawing out the layout of the pannel on paper
    (1.5 Hours)

    - Worked on creating the pannel in HTML & CSS
    (2 Hours)

August 20, 2019:
    - Started work to convert to a ReactJS app, started with tutorials and research.
    (2 Hours)

    - Tried to create first component, which would be a crossover button, which took
    4 hours to get an image in appear in ReactJS
    (4.5 Hours)

August 21, 2019:
    - Changed the componet which was going to be just a crossover button to the entire
    Suscon interlocking. Deciding that each interlocking will be it's own component.
    (3 Hours)

August 23, 2019:
    - Tried to figure out how to change an image in ReactJS, which ended up being a
    real big pain, having to import each image file similar to importing a scrip file.
    This solution seems weird but it's the only way I've been getting this to work so far.
    (4 Hours)

August 24, 2019:
    - Started to get all the crossover buttons working, and changing the image based on clicks,
    did this for both Suscon and Mill
    (1 Hour)

    - Converted what I did for the crossover buttons to use the 'state' object which is part of
    a ReactJS Component to more closely follow the conventions of ReactJS
    (2 Hours)

    -

August 25, 2019:
    - Set up the click function for the signals in Suscon
    (0.5 Hours)

- Changed the setup so all the components that make up the Main Line section are wrapped in one
component, this will make it eaiser for passing information from the components to the CTC controller
both ways.
(0.5 Hours)

- Changed all the track block colors to be based off the state in their component class.
(0.25 Hours)

- Setup the beginnings of the CTC Controller class which will act as the backend to controll all the
train movements, also created a ctc_block class which is be a instance for each track block on the railroad
(0.75 Hours)

- Tested out have the informmation in the CTC Controller class change the state of the MainLine component
by using the props property that is built into react to pass the status of 2 blocks into the component itself.
Luckily it works, and I now have the ability CTC Controller change the pannel.
(0.5 Hours)

August 26, 2019:
- Changed how to pass an object as props in the ReactJS component, instead of a variable for each piece of
information that needs to be passed
(0.5 Hours)

- Created a script for the Suscon interlocking to controll the train movements, and started to make this class
change the state of the UI and have the changes reflected in either the ReactJS component or the JS script
(1.5 Hours)

- Got the ReactJS component to send it's changes up to the script class for the interlocking, and having the
changes in the the script be reflecting in the Component, at this point the data is being sent both ways for
the curent status of the crossovers in the interlocking
(1 Hour)

August 27, 2019:
- Started converting Ridgewood Junction from HTML and CSS to a new ReactJS component, also made some changes from
my original design, and learned you do draw a diagonal line in CSS
(0.75 Hours)

- Converted Laurel to a new ReactJS component and cleaned up some of my previous designs
(0.5 Hours)

- Did some work on my drawing for the switch buttons on the pannel
(0.25 Hours)

- Converted the West Secaucus from into a ReactJS component
(0.5 Hours)

August 28, 2019:
- Drawing the Bergen County Line, finished about 95% of the line, creating all the components.
(5 Hours)

August 29, 2019:
- Finished Drawing all the tracks on the Bergen County side of the pannel
(1 Hours)

- Finished drawing of Laurel
(0.25 Hours)

- Finished drawing of ridgewood Junction
(0.25 Hours)

August 30, 2019:
- Reconfigured pascack Junction
(0.25 Hours)

- Finished the drawing in HX
(0.5 Hours)

- Finished drawings in Pascack Junction
(0.25 Hours)

- Finished drawings in BT and add tags for tracks that lead to other lines
(0.25 Hours)

- Orgainized the files structure a bit, and dealt with all the changing of
import statments
(0.25 Hours)

- Created a component for Harriman, starting the next line of tracks for the pannel
(0.5 Hours)

- Created a comonent for Sterling, and the tracks between the interlockings, also works on
parts of hilburn, and sf for the leads to the yard between the hilburn and sf
(1 Hour)

August 31, 2019:
- Finished drawing the the Main Line
(2 Hours)

- Continuted work on the Southern Tier line, finished about 85%
(4.5 Hours)

September 1, 2019:
- Finished All drawings on the pannel
(2 Hours)

September 2, 2019:
- Started connecting all the track blocks on the Main Line to the ctc scripts, now having
the ctc class controlling the front end of the pannel.
(2.5 Hours)

September 3, 2019:
- Fixed the bugs I was having trying to connect the ctc class to show the correct informating
when the frontend refreshes the screen.
(0.75 Hours)

- Created a new class for a game clock, so start trying to get the trains to move accross the
screen to
(0.5 Hours)

- Debugged a problem in the clock class, and now I have a hardcoded train moving accross some of
the blocks
(0.25 Hours)

- Configured the ctc_suscon class to create a route through the interlocking based on the status
of the switches and what signal you click
(0.5 Hours)

- Got the Suscon component to draw the routing in the UI givien the route status of the ctc_suscon
class
(4 Hours)

- Now the route draw turns to occupied if you set the interlocking

to that state and created a
    function to deal with that and will also clear the drawing when the hard coded train passes the
    location
    (2 Hours)

September 4, 2019:
    - Finally found a good way to store and display routes, using something that I
    already had in place.
    (1 Hour)

    - Created a class for the the trains, and now have a actual train moving around the pannel
    (2.5 Hours)

    - Figures out how to have the routes passes to the train, and now the train can move based off of
    how the player has created routes on the pannel
    (3 Hours)

    - Spent a while trying to get the interlocking to become occupied when a train is present, have yet
    to find a good way to do this and will have to revist it.
    (1.5 Hours)

September 5, 2019:
    - Again tried to figure out how to occupie the interlocking in a good way, and still have yet to
    come up with a ellagent solution
    (2 Hours)

    - West Secaucus is now fully operational
    (3 Hours)

    - Routing for Ridgewood Junction is now complete, all thats left is getting the drawings up and
    running, which might by interesting
    (2 Hours)

    - Testing 2 trains at once, and it seems to be working finished
    (0.5 Hours)

September 6, 2019:
    - Reworked how routes are given to trains to make it much easier to have trains to be going in both
    directions
    (1.5 Hours)

    - Fixed issue with the routings for all the current interlockings

(2 Hours)

September 7, 2019:
- Finished all the drawing for routes in Ridgewood, and have decided to finish the routing first, and
then go back to the drawings, because the take the most time
(6 Hours)

September 9, 2019:
- Finsihed the routings for Laurel
(2.5 Hours)

- Finished the routings at the WC interlocking, will still have to go back and add the drawins when
a route is lined
(1.5 Hours)

- Finished the routings in the SF interlocking
(1 Hours)

- Finished the routings in the Hilburn interlocking
(0.5 Hours)

- Did some debugging to fix an minor issue with train movements through hilburn interlocking, at this
point now a train can run from Laurel to Sterling
(0.25 Hours)

September 10, 2019:
- Got the switch in sterling to throw
(0.25 Hours)

- Connected all the blocks on the pannel to actual block classes in the ctc class
(1 Hour)

- Fixed a bug in the suscon interlocking routing
(0.25 Hours)

September 11, 2019:
- Hooked up the blocks on the southern tier section to the ctc mainline class
(0.5 Hours)

- Finished the switches and routing for Sterling, Harriman, and Central Valley interlockings
(2 Hours)

September 12, 2019:
- Finished Switches and routing for Hudson Junction

(0.5 Hours)

- Finished switches and routing for Howells
(0.5 Hours)

- Finished switches and routing for CP OV and for CP BC
(1 Hours)

September 15, 2019:
- Finished Switches and routins for CP Port, CP PA and CP Sparrow. Trains can now run the entire
length of the pannel
(2 Hours)

- Finished route drawings for CP BC
(0.5 Hours)

- Finished route drawings for CP OV
(0.25 Hours)

- Finished route drawings for CP Howells
(0.25 Hours)

September 16, 2019:
- Finished route drawings for CP Port
(0.25 Hours)

- Finished route drawigns fro CP Hall
(0.5 Hours)

- Finished route drawings for CP Sparrow
(0.25 Hours)

- Finished route drawings for CP Hudson Junction
(0.25 Hours)

- Finished route drawings for CP Central Valley
(0.25 Hours)

- Finally figured out how to occupy and interlocking, right now it just working for interlocking that have only one route,
but this is the majority of the interlockings, so I'll be able to get most of the interlockings working 100% in short order.
(0.5 Hours)

- Added occuping interlocking for CP Hudson Junction
(0.5 Hours)

- Commenting
(0.5 Hours)

- Can now occupy CP Hall, this is an interlocking that can have multiple routes, which was gonna be
more difficult than one route interlockings to get it to work.
(0.5 Hours)

- Can now occupied CP Howells
(0.25 Hours)

- Can now occupy CP BC and CP OV and CP Port
(0.5 Hours)

September 17, 2019:
- Drawings and occuping for CP PA
(1.5 Hours)

- Can now occupy CP Sparrow
(0.25 Hours)

- Drawing and occupy at Hilburn
(0.5 Hours)

- Drawings and Occupy at CP Sterling
(0.5 Hours)

- Drawing and Occupy finsihed at CP Harriman
(0.5 Hours)

- General Bug Fixes
(0.5 Hours)

- Drawigns and Occupy finished at SF
(0.75 Hours)

- Occupy finished at West Secaucus
(0.25 Hours)

- Route drawings and Occupy finished at Suscon
(0.75 Hours)

September 18, 2019:
- Route drawings and Occupy finished at Mill
(0.25 Hours)

- Bug Fixes at CP Port
(0.25 Hours)

- Trains now delete if they reach the ends of the railraod, either when they reach a
yard, or the end of pannels west of CP Sparrow or east of Laurel

(0.5 Hours)

- Fixed all east facing yard leads so trains to leave from them, had to change the way
those blocks were named
(0.25 Hours)

- Route drawings and Occupy finished at Ridgewood Junction
(3 Hours)

- Route drawings and Occupy finished at WC
(1.5 Hours)

- Fixing the Blocks on the Bergen County Line
(0.75 Hours)

- Setting up the switches to working at BT
(0.5 Hours)

- Setting up the switches at both Pascack and HX, now all the interlockings have working
switches
(0.75 Hours)

- Routing working for the BT interlocking
(0.5 Hours)

September 19, 2019:
- Routing working at Pascack interlocking
(0.5 Hours)

- Routing working at HX interlocking
(0.5 Hours)

- Bug fixes with routing around the laurel area
(0.75 Hours)

- Route drawings and occupy working at BT interlocking
(1 Hour)

- Route drawings and occupy working at Pascack interlocking
(0.75 Hours)

- Route drawings and occupy working at HX interlocking
(1 Hours)

September 20, 2019:
- Working on Drawings for Laurel Interlocking, finished about a third of them
(2 Hours)

- Debugging for the Laurel Interlocking
(0.75 Hours)

- Continued Drawings for Laurel Interlocking, finsihed about 2/3 of the them at this point
(2 Hours)

September 21, 2019:
- Finished drawings for the laurel interlocking and debugging some of the drawings that change
when another track has a different status.
(2 Hours)

- Finished the routing and occupy for Laurel Interlocking
(0.5 Hours)

- Rewrote the drawings for the WC interlocking using the new system I created and used at Laurel,
this fixed the bugs that had arrised in that interlocking
(3 Hours)

- Started adding symbol tags to the pannel for the blocks, finished all the symbols on the
southern tier section of the pannel
(1.5 Hours)

- Documenting Laurel.jsx
(0.75 Hours)

- Bug fixes at West Secaucus
(0.25 Hours)

September 22, 2019:
- Finished locations on all the symbols for the main line, still have to get the workings.
(0.75 Hours)

- CSS fixes for the text symbos on the pannel to keep it from moving around if the windo is resized
(0.25 Hours)

- Got the symbols of the Main Line working correctly
(1 Hour)

- Got the trains to get the size of the current block so it's more realistically timed as a train
moves accross the railroad.
(1 Hour)

- Finished the symbols on the bergen line
(0.75 Hours)

- Finished commenting for all of Laurel Interlocking
(0.5 Hours)

- Finished commenting for all of Mill Interlocking
(0.5 Hours)

- Finsihed commenting on all of the JSX components for the Main Line section
(2.5 Hours)

- Finsihed commenting on all of the JSX components for the Bergen County Line section
(1 Hour)

September 23, 2019:
- Commenting on JSX components
(3 Hours)

- Commenting finished on all CSS Files
(1 Hour)

- Commenting on all controller JS classes for the Bergen Line & Southern Tier Line
(4 Hours)

- Finish commenting on all the controller JS classes which was for the Main Line and others
(3 Hours)

September 24, 2019:
- Finished the manual and what ever was left of the documentation
(6 Hours)

TOTAL: 156.25 Hours