

```

/**
 * @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
 * controls 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;
    // Booleans for the switches
    this.sw_1 = false;
    this.sw_3 = false;
    // Booleans 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;
```