

```

/**
 * @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
 * controls 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() {
    // Booleans for the switches
    this.sw_1 = false;
    // Booleans 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_0V;
```