

```

/**
 * @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 controls 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;

```