```
mastervehicle_pseudocode.txt
Author: Wyatt Smith
Editor: George Huang
/****************************************************************************/

#include "ES_Configure.h"
#include "ES_Framework.h"
#include "MasterVehicle.h"
#include "ES_DeferRecall.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "inc/hw_timer.h"
#include "inc/hw_nvic.h"
#include "StageSM.h"
#include "ShootSM.h"
#include "SupplySM.h"
#include "Location.h"
#include "OwnPWM.h"
#include "LOCMaster.h"
#include "Servo_Control.h"
#include "Ultrasonic.h"

/*********************************some defines and variables declaration*******************/


/*---------------------------- Module Variables ---------------------------*/

set up a CurrentState variable for state machine
set up a Priority
set up a team variable to identify whether we are red or green
set up variables to store destinations (update when we have new active locations)
set up a variable to store how many balls we still have, NUM_BALL_AVAILABLE
set up variables to store the scores
set up variables storing ultrasonic coordinate information (X and Y) for
all staging, shooting, and supplying locations.

/*------------------------------ Module Code ------------------------------*/

/***************************************************************************/

bool InitMasterVehicleSM ( uint8_t Priority )
{
    save our priority
    Initialize port and pin needed for LEDs/IR Emitters/team selection switch
    Initialize location module
    Initialize PWM for Servo
    Start the Master State machine(this module) with ES_ENTRY
    Initialize NUM_BALL_AVAILABLE with 4 balls
}

/***************************************************************************/

bool PostMasterVehicleSM( ES_Event ThisEvent )
{
  post event to this service by calling the priority of this service
}
```

```
/****************************************************************************/

ES_Event RunMasterVehicleSM( ES_Event CurrentEvent )
{
    switch ( CurrentState )
   {
        case INIT_STATE :
             if an event is active (not ES_NO_EVENT)
             {
                switch Event Type
                {
                   case CONSTRUCTION_START:
                      set NextState to IDLE_STATE
                      set MakeTransition to true
                      consume event (set ReturnEvent type to ES_NO_EVENT)
                      break;

                   default:
                      break;
                }
             }
             break;

        case IDLE_STATE :
            if an event is active (not ES_NO_EVENT)
             {
                switch Event Type
                {
                   case STAGE_ACTIVE:
                        update game status to reflect current active staging area
                        set NextState to STAGING_STATE
                        set MakeTransition to true
                        consume event (set ReturnEvent type to ES_NO_EVENT)
                        break;

                   case SHOOT_ACTIVE:
                        update game status to reflect current active shooting area
                        set NextState to SHOOTING_STATE
                        set MakeTransition to true
                        consume event (set ReturnEvent type to ES_NO_EVENT)
                        break;

                   case SHOOT_ACTIVE_4: //If event is shoot active (for last 18 seconds)
                        update game status to reflect current active staging area
                        set NextState to SHOOTING_STATE
                        set MakeTransition to true
                        consume event (set ReturnEvent type to ES_NO_EVENT)
                        break;

                   case NO_BALL:
                        update game status to reflect current active staging area
                        set NextState to SUPPLYING_STATE
                        set MakeTransition to true
                        consume event (set ReturnEvent type to ES_NO_EVENT)
                        break;

                   case CONSTRUCTION_END:
                        set next state to init state (we want to stop everything)
                        set MakeTransition to true
                        consume event (set ReturnEvent type to ES_NO_EVENT)
```

```
                    break;

                default:
                    break;
            }
         }
         break;




    case STAGING_STATE :

        if an event is active (not ES_NO_EVENT)
         {
            switch Event Type
            {
                case ES_TIMEOUT:
                    set NextState to IDLE_STATE
                    set MakeTransition to true
                    consume event (set ReturnEvent type to ES_NO_EVENT)
                    check if there is any active event on our way out
                    break;

                case SHOOT_ACTIVE_4:
                     if event is shoot active (during the last 18 seconds)
                    set NextState to IDLE_STATE
                    set MakeTransition to true
                    consume event (set ReturnEvent type to ES_NO_EVENT)
                    check if there is any active event on our way out
                    break;

                case FINISHED_STAGING: //if we successfully verify frequency
                    set NextState to IDLE_STATE
                    set MakeTransition to true
                    consume event (set ReturnEvent type to ES_NO_EVENT)
                    break;

                case CONSTRUCTION_END: //If event is construction end
                    set NextState to INIT_STATE
                    set MakeTransition to true
                    consume event (set ReturnEvent type to ES_NO_EVENT)
                    break;

                default:
                  break;
            }
         }
         break;
```

```
case SHOOTING_STATE :

    if an event is active (not ES_NO_EVENT)
    {
        switch Event Type
        {

            case ES_TIMEOUT: (If event is 20 SECOND SHOOTING TIMEOUT)
                set NextState to IDLE_STATE
                set MakeTransition to true
                consume event (set ReturnEvent type to ES_NO_EVENT)
                check if there is any active event on our way out
                break;

            case SCORE_CHANGED:
                set NextState to IDLE_STATE
                set MakeTransition to true
                consume event (set ReturnEvent type to ES_NO_EVENT)
                check if there is any active event on our way out
                break;

            case NO_BALL:
                set NextState to SUPPLYING_STATE
                set MakeTransition to true
                consume event (set ReturnEvent type to ES_NO_EVENT)
                break;

            case CONSTRUCTION_END:
                set NextState to INIT_STATE
                set MakeTransition to true
                consume event (set ReturnEvent type to ES_NO_EVENT)
                break;

         default:
           break;
    }
 }
 break;

case SUPPLYING_STATE :

    if an event is active (not ES_NO_EVENT)
     {
        switch Event Type
        {
            case LOADED_COMPLETE:
                set NextState to IDLE_STATE
                set MakeTransition to true
                consume event (set ReturnEvent type to ES_NO_EVENT)
                check if there is any active event on our way out
                break;

            case CONSTRUCTION_END:
                set NextState to INIT_STATE
                set MakeTransition to true
                consume event (set ReturnEvent type to ES_NO_EVENT)
                check if there is any active event on our way out
                break;

            default:
                break;
                }
            }
            break;

        default:
            break;
    }

    if we are making a state transition
    {
       set CurrentEvent type to ES_EXIT;
       call run function for SM with CurrentEvent, to execute exit function of current state
       update the state
       call run function again, with EntryEventKind (defaults to ES_ENTRY),
       to execute entry function for new state

    }

    //return ReturnEvent
}



/****************************************************************************/

void StartMasterVehicleSM ( ES_Event CurrentEvent )
{
  set CurrentState to INIT_STATE
  call SM run function with CurrentState to init lower level state machines
}


/* This function return the current state of this state machine */
MasterVehicleState_t QueryMasterVehicleSM ( void )
{
   return the CurrentState
}

/***************************************************************************/

static ES_Event DuringInitState( ES_Event Event)
{
        // set ReturnEvent to Event;

        if event type is ES_ENTRY or ES_ENTRY_HISTORY
        {
            read team input and set team collor
            make sure construction led is off
            make sure motor is not running when we start the game
        }
        else if event type is ES_EXIT
        {
            Turn on construction LED (when we exit init state, that means game is on)
        }
        else{
            (No lower state machine)
        }

        return ReturnEvent
}
```

```
static ES_Event DuringIdleState( ES_Event Event)
{
        // set ReturnEvent to Event;

        if event type is ES_ENTRY or ES_ENTRY_HISTORY
        {
            (nothing to do here)
        }
        else if event type is ES_EXIT
        {
            (nothing to do here)
        }
        else{
            (No lower state machine)
        }

        return ReturnEvent
}

static ES_Event DuringStagingState( ES_Event Event)
{
        // set ReturnEvent to Event;

     if event type is ES_ENTRY or ES_ENTRY_HISTORY
        {
            post STAGE_ACTIVE to StageSM
        }
        else if event type is ES_EXIT
        {
            call StageSM Run function to execute exit
            stop the 2s staging limit timer, STAGE_TIMER
        }
        else{
            call StageSM run function with Event to pass it down
        }

        return ReturnEvent
}

static ES_Event DuringShootingState( ES_Event Event)
{
        // set ReturnEvent to Event;

        if event type is ES_ENTRY or ES_ENTRY_HISTORY
        {
            post SHOOT_ACTIVE to ShootSM by calling ShootSM Start function
            start 20s timer limit for shooting
        }
        else if event type is ES_EXIT
        {
            call ShootSM Run function to execute exit
        }
        else{
            call StageSM run function with Event to pass it down
        }

        return ReturnEvent
}
```

```
static ES_Event DuringSupplyingState( ES_Event Event)
{
        // set ReturnEvent to Event;

        if event type is ES_ENTRY or ES_ENTRY_HISTORY
        {
            post NO_BALL to SupplySM by calling SupplySM Start function
        }
        else if event type is ES_EXIT
        {
            call SupplySM Run function to execute exit
            check which event is currently active at exit
        }
        else{
            call SupplySM run function with Event to pass it down
        }

        return ReturnEvent
}

static void Init_LED_Port(){
        Initialize Ports B, E, F and correct pins for leds and team selection switch
       Initialize IR LED (for resupply)
}

/* This function reads value from PB0 and sets team accordingly */
static void Team_Selection(){
        if team selection switch pin is low
            set team to RED

        else if switch pin is high
            set team to GREEN

        if team is red
            light up red LED

        else if team is green
            light up green LED
}

static void Set_Construction_LED(){
        turn on construction LED
}

static void Clear_Construction_LED(){
        turn off construction LED
}

static void update_Status(){
        set status to status from LOC master module
}

int get_Team(void){
        return team, green or red
}

uint32_t get_Supply_location_x(){
        if team is red
        {
            return X coordinate of the red supply station
        }
        else{
            return X coordinate of the green supply station
        }
}

uint32_t get_Supply_location_y(){
        if team is red
        {
            return Y coordinate of the red supply station
        }
        else
        {
            return Y coordinate of the green supply station
        }
}

/* The folowing functions return the loations of stage and shoot areas for red and green */
uint32_t get_Stage_Green_X(uint8_t index){

    return X coordinate of this stage area
}
uint32_t get_Stage_Green_Y(uint8_t index){
    return Y coordinate of this stage area
}
uint32_t get_Stage_Red_X(uint8_t index){
    return X coordinate of this stage area
}
uint32_t get_Stage_Red_Y(uint8_t index){
    return Y coordinate of this stage area
}
uint32_t get_Shoot_Green_X(uint8_t index){
    return X coordinate of this shooting area
}
uint32_t get_Shoot_Green_Y(uint8_t index){
    return Y coordinate of this shooting area
}
uint32_t get_Shoot_Red_X(uint8_t index){
    return X coordinate of this shooting area
}
uint32_t get_Shoot_Red_Y(uint8_t index){
    return Y coordinate of this shooting area
}

int get_num_ball(void){
    return NUM_BALL_AVAILABLE, which gets updated by shooting and supplying
}

/* This function increases the current number of balls by 1*/
void increment_num_ball(void){
    increase NUM_BALL_AVAILABLE by 1
}

/* This function decreases the current number of balls by 1*/
void decrement_num_ball(void){
    decrease NUM_BALL_AVAILABLE by 1
}

/* This function updates the score variable using the inputs */
void update_score(uint8_t red_input, uint8_t green_input){
    write the corresponding input to the corresponding score
}
```

```
/* This function returns the current red score */
uint8_t get_red_score(void){
    return red_score, which is updated by LOC
}

/* This function returns the current green score */
uint8_t get_green_score(void){
    return green_score, which is updated by LOC
}
```