```
shootsm_pseudocode.txt
Author: Wyatt Smith

/****************************************************************************/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ShootSM.h"
#include "Location.h"
#include "LOCMaster.h"
#include "MasterVehicle.h"
#include "PWMTiva.h"
#include "Servo_Control.h"

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


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

declare a CurrentState variable for state machine
declare current destination variables for X and Y
declare boolean shooter reset complete flag
declare score-holding variables for both before and after shooting
declare last 18 seconds flag
declare score changed flag
declare shot delay time, initialize to 2s

/*------------------------------ Module Code ------------------------------*/
/***************************************************************************/
ES_Event RunShootSM( ES_Event CurrentEvent )
{
    set MakeTransition flag to false (default)
    set next state to current state (default)
    set EntryEventKind variable to ES_ENTRY (default to normal entry)
    set return event to current event (default: assume not consuming event)

    switch ( CurrentState )
    {
        case SHOOT_WAITING :
            Execute During function for the state
            (ES_ENTRY & ES_EXIT are processed here to allow the lower level
            state machines to re-map or consume the event.)

            if an event is active (not ES_NO_EVENT)
            {
                switch (CurrentEvent.EventType)
                {
                    case SHOOT_ACTIVE :
                        reset shot delay timer to 4.5 seconds
                        declare variable to store array index of destination

                        if team is green
                        {
                            get destination index (queryActiveShootingGreen())
                            if destination is not 0 or 5
                            {
                                if destination is 4 (meaning last 18 seconds)
                                {
                                    set destination to G1
                                    set last_eighteen_second to true
                                    reduce shot delay time to 0.5s
                                }
                            set CurrentX/YDestination variables to
                             coordinates based on destination index
                        }
                    }else{ (team is red)
                        get destination index (queryActiveShootingRed())
                        if destination is not 0 or 5
                        {
                            if destination is 4 (meaning last 18 seconds)
                            {
                                set destination to R1
                                set last_eighteen_second to true
                                reduce shot delay time to 0.5s
                            }
                            set CurrentX/YDestination variables
                           to coordinates based on destination index
                        }
                    }
                    set NextState to SHOOT_MOVE_Y
                    set MakeTransition to true
                    consume event (set ReturnEvent type to ES_NO_EVENT)
                    break;

                default:
                    break;
            }
        }
        break;

    case SHOOT_MOVE_Y :
        Execute During function for the state
        (ES_ENTRY & ES_EXIT are processed here to allow the lower level
        state machines to re-map or consume the event.)

        if an event is active (not ES_NO_EVENT)
        {
            switch (CurrentEvent.EventType)
            {
                case ES_TIMEOUT:
                    if event parameter is STAGE_TIMER
                    {
                        consume event (set ReturnEvent type to ES_NO_EVENT)
                    }
                    break;

                case Y_REACHED :
                    set next state to SHOOT_MOVE_X
                    set MakeTransition flag to true
                    consume event (set ReturnEvent type to ES_NO_EVENT)
                    break;

                case CONSTRUCTION_END:
                    set next state to SHOOT_WAITING
                    set MakeTransition flag to true
                    set return event to CONSTRUCTION_END (to post to upper SM)
                    break;
                default:
                    break;
            }
        }
        break;
        case SHOOT_MOVE_X :
            Execute During function for the state
            (ES_ENTRY & ES_EXIT are processed here to allow the lower level
            state machines to re-map or consume the event.)

            if an event is active (not ES_NO_EVENT)
            {
                switch (CurrentEvent.EventType)
                {
                    case ES_TIMEOUT:
                        if event parameter is STAGE_TIMER
                        {
                            consume event (set ReturnEvent type to ES_NO_EVENT)
                        }
                        break;

                    case X_REACHED :
                        if we verify y location and it's correct
                        {
                            set PWM to normal speed
                            set next state to RESET_SHOOTER
                        }else{ (y location is wrong)
                            set PWM to slower speed for location correction
                            set next state to SHOOT_MOVE_Y
                        }
                        set MakeTransition flag to true
                        consume event (set ReturnEvent type to ES_NO_EVENT)
                        break;

                    case CONSTRUCTION_END:
                        set next state to SHOOT_WAITING
                        set MakeTransition flag to true
                        set return event to CONSTRUCTION_END (to post to upper SM)
                        break;

                    default:
                        break;
                }
            }
            break;

        case SHOOTING :
            Execute During function for the state
            (ES_ENTRY & ES_EXIT are processed here to allow the lower level
            state machines to re-map or consume the event.)

            if an event is active (not ES_NO_EVENT)
            {
                switch (CurrentEvent.EventType)
                {
                    case SCORE_CHANGED:
                        post event to master vehicle SM
                        consume event (set return event type to ES_NO_EVENT)
                        (we consume this event, because we have to make sure we go to reset
                        finished shooting first, if we already started the process)
                        break;

                    case ES_TIMEOUT:
                        if we get Stage timer timeout
                        {
                            ignore it: consume this event
                        }
                        if we get start shooting timer timeout event
                        {
                            latch the latch servo (latch_LatchServo())
                            dispense ball (dispense_Ball())
                            set flag for shooter reset to false
                            start latch dispense timer
                            consume event
                        }else if we get latch dispense timer timeout
                        {
                            reset dispense servo arm
                            tension the spring servo
                            set shooter reset flag to false, again
                            (to ensure it's still false)
                            start ready to shoot timer to make sure this process is completed
                            consume event
                        }else if dispensing the ball is completed (READY_TO_SHOOT_TIMER)
                        {
                            unlatch the latch servo to shoot
                            make sure that this reset flag is still false
                            decrement number of balls we have
                            start shot delay timer

                            (here we are still not changing state, have to wait for the timer
                            to timeout to make sure it finishes shooting; however,
                            if number of balls
                            is less than 0, we post no_ball event to the main state machine)

                            if number of balls is <= 0
                            {
                                consume event
                                post no ball event to master vehicle SM
                            }else{
                                consume event
                            }
                        }else if shooting is complete (SHOT_DELAY_TIMER)
                        {
                            set next state to RESET_SHOOTER
                            set MakeTransition flag to true
                            consume event
                            set shooter reset complete flag to false
                        }else if 20s timeout comes and we still haven't shot,
                            go back to waiting
                        {
                            set next state to SHOOT_WAITING
                            set MakeTransition flag to true
                            set return event to ES_TIMEOUT
                        }else if 20s timeout comes and we HAVE shot
                            set next state to RESET_SHOOTER
                            set MakeTransition flag to true
                            consume event
                            repost CurrentEvent to master SM so
                            we can handle this in reset_shooter state
                        }
                        break;

                    case NO_BALL:
                        if shooter reset complete flag is false (we need to reset)
                        {
                            set return event to ES_NO_EVENT (consumes event)
                            post current event to master vehicle SM
```

```
                    (keeps us cycling to here until shooting done)
                }else{
                    set next state to SHOOT_WAITING
                    set MakeTransition flag to true
                    set return event to NO_BALL

            case CONSTRUCTION_END:
                set next state to SHOOT_WAITING
                set MakeTransition flag to true
                set return event to CONSTRUCTION_END
                break;

            default:
                break;
        }
    }
    break;

case RESET_SHOOTER :
    Execute During function for the state
    (ES_ENTRY & ES_EXIT are processed here to allow the lower level
    state machines to re-map or consume the event.)

    if an event is active (not ES_NO_EVENT)
    {
        switch (CurrentEvent.EventType)
        {
            (Notes: Here, we can exit shooting SM only when we finished reset shooter
             When reset_delay_timer is done, we will set the boolean reset_complete to
            true. if score_changed, construction_end or ES_timeout that is not reset
            shooter timer comes, we can post these event back to mastervehicle until
            the boolean reset_complete is set to true when the shooter is reset, we
            also query the system to see if score change or not only when
             reset_complete is true and score is not changed that we can go back to
            shooting state)

            case SCORE_CHANGED:
                set NextState to SHOOT_WAITING
                set MakeTransition to true
                set ReturnEvent to SCORE_CHANGED to propagate it to upper SM
                break;

            case ES_TIMEOUT:
                if timeout is for 20s shooting timer
                {
                    if shooter reset flag shows reset is complete
                    {
                        set next state to SHOOT_WAITING
                        set MakeTransition flag to true
                        set return event to ES_TIMEOUT to propagate to upper SM
                    }else{ (we haven't finished resetting the shooter)
                        repost current event to master vehicle SM
                        set return event to ES_NO_EVENT
                    }
                }else if timeout is for shooter reset timer
                    (we know we've finished resetting)
                {
                    set shooter reset flag to true
                    update score for our team
                    if it is the last 18 seconds
                    {
```

```
                                set next state to SHOOTING
                                set MakeTransition flag to true
                                set return event to ES_NO_EVENT
                            }else{
                                if score went up
                                {
                                    set next state to SHOOT_WAITING
                                    set MakeTransition flag to true
                                    set return event to SCORE_CHANGED
                                }else if score hasn't changed
                                {
                                    set next state to SHOOTING
                                    set MakeTransition flag to true
                                    set return event to ES_NO_EVENT
                                }
                            }
                        }
                        break;

                    case NO_BALL:
                        if shooter reset complete flag is true
                        {
                            set next state to SHOOT_WAITING
                            set MakeTransition flag to true
                            set return event to NO_BALL
                        }else{ (no balls but not done resetting -- keep posting until we are)
                            post current event (NO_BALL) to master vehicle SM
                            set return event to ES_NO_EVENT
                        }
                        break;

                    case CONSTRUCTION_END:
                        set next state to SHOOT_WAITING
                        set MakeTransition flag to true
                        set return event to CONSTRUCTION_END
                        break;

                    default:
                        break;
                }
            }
            break;

        default:
            break;

        if MakeTransition flag is true
        {
            set current event to ES_EXIT
            call shootsm's run function with current event to execute exit

            set current state to next state

            call shootsm's run function on EntryEventKind
            (this defaults to ES_ENTRY, so this executes new state's entry function)
        }
        return the return event
    }
}

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

```
void StartShootSM ( ES_Event CurrentEvent )
{
    if we are NOT starting with history
    {
        set current state to ENTRY_STATE (default entry)
    }

    call shootsm's run function to execute any entry function for the entry state
}

/****************************************************************************/
ShootingState_t QueryShootSM ( void )
{
    return the current state
}

/***************************************************************************
 private functions:
 ***************************************************************************/

static ES_Event DuringWaiting( ES_Event Event)
{
    set return event to Event(assme no re-mapping or comsumption)

    if event type is ES_ENTRY or ES_ENTRY_HISTORY
    {
        (unused)
    }else if event type is ES_EXIT
    {
        store initial score for our team
    }else{
        (unused)
    }

    return the return event
}

static ES_Event DuringMoveY( ES_Event Event)
{
    set return event to Event(assme no re-mapping or comsumption)

    if event type is ES_ENTRY or ES_ENTRY_HISTORY
    {
        move in Y to current destination's Y coordinate
    }else if event type is ES_EXIT
    {
        stop the motors
    }else{
        (unused)
    }
    return the return event
}

static ES_Event DuringMoveX( ES_Event Event)
{
    set return event to Event(assme no re-mapping or comsumption)

    if event type is ES_ENTRY or ES_ENTRY_HISTORY
    {
        move in X to current destination's X coordinate
    }else if event type is ES_EXIT
    {
        stop the motors
    }else{
        (unused)
    }

    return the return event
}

static ES_Event DuringShooting( ES_Event Event)
{
    set return event to Event(assme no re-mapping or comsumption)

    if event type is ES_ENTRY or ES_ENTRY_HISTORY
    {
        start shooter timer to trigger shooting process
    }else if event type is ES_EXIT
    {
        (unused)
    }else{
        (unused)
    }

    return the return event
}

static ES_Event DuringResetShooter( ES_Event Event)
{
    set return event to Event(assme no re-mapping or comsumption)

    if event type is ES_ENTRY or ES_ENTRY_HISTORY
    {
        start shooter reset timer
        reset COW dispenser servo arm (reset_DispenseServo())
        relax the spring tensioner servo to lower catapult arm (relax_SpringServo())
    }else if event type is ES_EXIT
    {
        (unused)
    }else{
        (unused)
    }

    return the return event
}







/***************************************************************************
 Helper Functions
 ***************************************************************************/
uint32_t queryShootingCurrentXDestination(void){
    return the current X destination variable
}

uint32_t queryShootingCurrentYDestination(void){
    return the current Y destination variable
```

```
}

/****************************************************************************
 Private Functions
 ****************************************************************************/

static void unlatch_LatchServo(void){
    move latch servo to unlatched angle (moveToAngle(...))
}

static void latch_LatchServo(void){
    move latch servo to latched angle (moveToAngle(...))
}

static void reset_DispenseServo(void){
    move dispenser servo to reset angle (moveToAngle(...))
}

static void dispense_Ball(void){
    move dispenser servo to dispensing angle (moveToAngle(...))
}

static void relax_SpringServo(void){
    move spring servo to untensioned angle (moveToAngle(...))
}

static void tension_SpringServo(void){
    move spring servo to tensioned angle (moveToAngle(...))
}

static void set_last_eighteen_second(void){
    set last 18 seconds flag to true
}

void set_shot_delay_for_last_eighteen(void){
    set shot delay time variable to half second
}

void reset_shot_delay_time(void){
    set shot delay time to four half seconds (2s)
}
```