```
/*******************************************************************************
 Module
    Round2HSM.c

 Description
    This is a template file for implementing state machines.


*******************************************************************************/
/*------------------------------ Include Files ------------------------------*/
// Basic includes for a program using the Events and Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <stdio.h>

/* Include header files for this state machine as well as any machines at the
 next lower level in the hierarchy that are sub-machines to this machine
 */
#include "Round2HSM.h"


/*------------------------------ Module Defines ------------------------------*/
// define constants for the states for this machine
// and any other local defines

/*------------------------------ Module Functions ------------------------------*/
/* prototypes for private functions for this machine, things like during
 functions, entry & exit functions.They should be functions relevant to the
 behavior of this state machine
 */
static ES_Event DuringSoccering2HSM(ES_Event Event);
static ES_Event DuringCharging2HSM(ES_Event Event);


/*------------------------------ Module Variables ------------------------------*/
// everybody needs a state variable, you may need others as well
static Round2State_t CurrentState;

/*------------------------------ Module Code ------------------------------*/
/*******************************************************************************
 Function
 RunRound1HSM

 Parameters
 ES_Event: the event to process

 Returns
 ES_Event: an event to return

 Description
 add your description here


*******************************************************************************/
ES_Event RunRound2HSM(ES_Event CurrentEvent)
{
```

```
    unsigned char MakeTransition = false;/* are we making a state transition?
*/
    Round2State_t NextState = CurrentState;
    ES_Event EntryEventKind = {ES_ENTRY,0};// default to normal entry to new
state
    ES_Event ReturnEvent = CurrentEvent; // assume we are not consuming event
    ES_Event PostEvent;

    #ifdef PRINT_ROUND2_HSM_CALLS

        printf("RunRound2HSM\n\r");

    #endif

    switch(CurrentState)
    {
        case Soccering2:

            #ifdef PRINT_ROUND2_HSM_STATES

                printf("Soccering2\n\r");

            #endif

            // Execute During function for Soccering. ES_ENTRY & ES_EXIT are
processed here allow the lowere level state machines to re-map or consume the
event
            CurrentEvent = DuringSoccering2HSM(CurrentEvent);
            // Process any events
            if(CurrentEvent.EventType != ES_NO_EVENT) // if an event is
active
            {
                switch(CurrentEvent.EventType)
                {
                    case Out_of_Ammo: // event put into Completing state

                        #ifdef PRINT_ROUND1_HSM_EVENTS

                            printf("Out_of_Ammo\n\r");

                        #endif

                        NextState = Charging2;
                        MakeTransition = true;
                        EntryEventKind.EventType = ES_ENTRY;
                        ReturnEvent = CurrentEvent;
                        break;
                }
            }
            break;

        case Charging2:

            #ifdef PRINT_ROUND2_HSM_STATES

                printf("Charging2\n\r");
```

```
            #endif

            // Execute During function for Charging. ES_ENTRY & ES_EXIT are
processed here allow the lowere level state machines to re-map or consume the
event
            CurrentEvent = DuringCharging2HSM(CurrentEvent);
            // Process any events
            if(CurrentEvent.EventType != ES_NO_EVENT) // if an event is
active
            {
                switch(CurrentEvent.EventType)
                {

                                        case ES_TIMEOUT :

                                                if(CurrentEvent.EventParam ==
LANCE_DELAY_TIMER)
                                                {
                                                        PostEvent.EventType =
LowerLance;

        PostMasterHSM(PostEvent);

                                                }

                                                break;

                }
            }
            break;



             }



    //   If we are making a state transition
    if (MakeTransition == true)
    {
        // Execute exit function for current state
        CurrentEvent.EventType = ES_EXIT;
        RunRound2HSM(CurrentEvent);

        CurrentState = NextState; //Modify state variable

        // Execute entry function for new state
        // this defaults to ES_ENTRY
        RunRound2HSM(EntryEventKind);
    }
    return(ReturnEvent);
}
/*****************************************************************************
 Function
    StartRound2HSM

 Parameters
```

```
    None

 Returns
    None

 Description
    Does any required initialization for this state machine


*****************************************************************************/
void StartRound2HSM(ES_Event CurrentEvent)
{
    // local variable to get debugger to display the value of CurrentEvent
    ES_Event LocalEvent = CurrentEvent;

   #ifdef PRINT_ROUND2_HSM_CALLS

        printf("StartRound2HSM\n\r");

    #endif

    // to implement entry to a history state or directly to a substate
    // you can modify the initialization of the CurrentState variable
    // otherwise just start in the entry state every time the state machine
    // is started
    if(ES_ENTRY_HISTORY != CurrentEvent.EventType) // query ammunition
    {
        CurrentState = Soccering2; // could be Soccering
    }
    // call the entry function (if any) for the ENTRY_STATE
    RunRound2HSM(CurrentEvent);
}

/*****************************************************************************
 Function
    QueryRound2HSM

 Parameters
    None

 Returns
    Round1State_t The current state of the Template state machine

 Description
    returns the current state of the Template state machine


*****************************************************************************/
Round2State_t QueryRound2HSM(void)
{

    #ifdef PRINT_ROUND2_HSM_CALLS

        printf("QueryRound2HSM\n\r");

    #endif
```

```
    return(CurrentState);
}

/*****************************************************************************
 private functions
 *****************************************************************************/
static ES_Event DuringSoccering2HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption

    #ifdef PRINT_ROUND2_HSM_CALLS

        printf("DuringSocceringHSM\n\r");

    #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
                // Implement any entry actions required for this state machine

        // after that start any lower level machines that run in this state
        StartSocceringSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level

    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
        RunSocceringSM(Event);

        // repeat for any concurrently running state machines

        // now do any local exit functionality

    }
    else // do the 'during' function for this state
    {
        // run any lower level state machine
        ReturnEvent = RunSocceringSM(Event);

        // repeat for any concurrent lower level machines

        // do any activity that is repeated as long as we are in this state

    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringCharging2HSM(ES_Event Event)
{
```

```
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption

    #ifdef PRINT_ROUND2_HSM_CALLS

        printf("DuringCharging2HSM\n\r");

    #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
                // Implement any entry actions required for this state machine

        // after that start any lower level machines that run in this state
                Event.EventParam = 2;
        StartDrivingHSM(Event);
                //StartShootingHSM(Event); only enable if our strategy gives
us balls at the start of round 2
                StartJoustingSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
        RunDrivingHSM(Event);
                StopJoustingSM(Event);

        // repeat for any concurrently running state machines

        // now do any local exit functionality

    }
    else // do the 'during' function for this state
    {
        // run any lower level state machine
        RunDrivingHSM(Event);
                RunJoustingSM(Event);

        // repeat for any concurrent lower level machines

        // do any activity that is repeated as long as we are in this state
    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}
```