```c
 * wamControl.c

#include "wamControl.h"
#include <stdlib.h> //needed inclusion to use use srand and rand
#include "supportFiles/display.h"

#define MIN 0                      //used to initialize many variables
#define TICK_DIVIDER 21            //used with modulo operator to make sure tick counts are in a certain range
#define AWAKE_TICK_MIN 10          //this and the next one ensure that the tick counts will be 10 at the least for the first level
#define ASLEEP_TICK_MIN 10
#define DIFFICULTY_MULTIPLIER 2 //will be multiplied by level to make tick counts smaller
#define MIN_COUNT 4                //the minimum tick count that it will be set to if it gets to 0

enum wamControl_st_t{
    init_st,                //first state
    waiting_for_touch_st, //state where we wait for the use to touch board
    debounce_st,            //used to settle adc
    end_st                  //last state before starting SM over
}wamCurrentState; //variable used to track the current state

static wamDisplay_point_t touchedPoint;      //global variable used store touch data
static uint8_t z;                            //needed to meet parameters of getTouchedPoint function
static uint16_t period, maxMisses, maxMoles; //variables which will be set by the main function
static uint32_t randomSeed;
static bool gameOver;                        //used by main to determine if the game has been completed

// Call this before using any wamControl_ functions.
void wamControl_init(){
    wamCurrentState = init_st; //SM should always start in this state
    gameOver = false; //assume the game is not over at the beginning
    srand(randomSeed); //this needs only be done once for the whole program
}

// Call this to set how much time is consumed by each tick of the controlling state machine.
// This information makes it possible to set the awake and sleep time of moles in ms, not ticks.
void wamControl_setMsPerTick(uint16_t msPerTick){
    period = msPerTick;
}

// This returns the time consumed by each tick of the controlling state machine.
uint16_t wamControl_getMsPerTick(){
    return period;
}

// Standard tick function.
void wamControl_tick(){
    //switch statement for transition actions and state changes
    switch(wamCurrentState){
    case init_st:
        wamCurrentState = waiting_for_touch_st; //go immediately to the next state. No other actions needed
        break;
    case waiting_for_touch_st:
        if(wamDisplay_getMissScore() == maxMisses){ //first check if the user has reached the max amount of misses
            wamDisplay_drawGameOverScreen(); //if we get into the if statement, the game is over. Draw the game over screen
            gameOver = true; //make sure to mark this high so main knows the game is over
            wamCurrentState = end_st; //go to the end_st
        }
        else if(display_isTouched()){ //if the max miss count has not been reached and the display is touched
            display_clearOldTouchData(); //always clear the old touch data
            wamDisplay_updateAllMoleTickCounts(); //call this function to draw/erase the moles as is necessary
            wamCurrentState = debounce_st;
        }
        break;
    case debounce_st:
        wamDisplay_updateAllMoleTickCounts(); //there has been a tick during gameplay, so make sure to call this
        display_getTouchedPoint(&touchedPoint.x, &touchedPoint.y, &z); //use display function to get touch coordinates
        wamDisplay_whackMole(&touchedPoint); //call this to see if the user hit an active mole
        wamCurrentState = waiting_for_touch_st; //go back to waiting state
        break;
    case end_st:
        if(display_isTouched()){ //we only start the SM back over if the game over screen is touched
            gameOver = false; //reset the variables
            touchedPoint.x = MIN;
            touchedPoint.y = MIN;
            z = MIN;
            wamCurrentState = init_st; //go back to beginning of SM
        }
        break;
    default:
        break;
    }

    //switch statement for state actions
    switch(wamCurrentState){
    case init_st: //nothing to be done in init_st
        break;
    case waiting_for_touch_st:
        if(wamDisplay_getActiveMoleCount() < maxMoles){ //if the current active mole amount is less than the permitted max
            wamDisplay_activateRandomMole(); //activate another mole
        }
        wamDisplay_updateAllMoleTickCounts(); //we have to update the tick counts
        break;
    case debounce_st: //nothing to be done in debounce_st
        break;
    case end_st: //nothing to be done in end_st
        break;
    default:
        break;
    }
}

// Returns a random value that indicates how long the mole should sleep before awaking.
wamDisplay_moleTickCount_t wamControl_getRandomMoleAsleepInterval(){
    wamDisplay_moleTickCount_t count = (rand() % TICK_DIVIDER) + AWAKE_TICK_MIN - DIFFICULTY_MULTIPLIER*wamDisplay_getLevel(); //the tick count will be less as we progress through levels
    if(count <= MIN) count = MIN_COUNT; //if the tick count gets below zero, set it back to 4 as the minimum

    return count; //return this randomly generated tick count
}

// Returns a random value that indicates how long the mole should stay awake before going dormant.
wamDisplay_moleTickCount_t wamControl_getRandomMoleAwakeInterval(){
    wamDisplay_moleTickCount_t count = (rand() % TICK_DIVIDER) + AWAKE_TICK_MIN - DIFFICULTY_MULTIPLIER*wamDisplay_getLevel(); //the tick count will be less as we progress through levels
    if(count <= MIN) count = MIN_COUNT; //if the tick count gets below zero, set it back to 4 as the minimum

    return count; //return this randomly generated tick count
}

// Set the maximum number of active moles.
void wamControl_setMaxActiveMoles(uint16_t count){
    maxMoles = count;
}

// Get the current allowable count of active moles.
uint16_t wamControl_getMaxActiveMoles(){
    return wamDisplay_getActiveMoleCount();
}

// Set the seed for the random-number generator.
void wamControl_setRandomSeed(uint32_t seed){
    randomSeed = seed;
}

// Set the maximum number of misses until the game is over.
void wamControl_setMaxMissCount(uint16_t missCount){
    maxMisses = missCount;
}

// Use this predicate to see if the game is finished.
bool wamControl_isGameOver(){
    return gameOver;
}


```