

wamControl.c

```
2 * wamControl.c
7
8 #include "wamControl.h"
9 #include <stdlib.h> //needed inclusion to use srand and rand
10 #include "supportFiles/display.h"
11
12 #define MIN 0 //used to initialize many variables
13 #define TICK_DIVIDER 21 //used with modulo operator to make sure tick counts
    are in a certain range
14 #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
15 #define ASLEEP_TICK_MIN 10
16 #define DIFFICULTY_MULTIPLIER 2 //will be multiplied by level to make tick counts
    smaller
17 #define MIN_COUNT 4 //the minimum tick count that it will be set to if it
    gets to 0
18
19 enum wamControl_st_t{
20     init_st, //first state
21     waiting_for_touch_st, //state where we wait for the use to touch board
22     debounce_st, //used to settle adc
23     end_st //last state before starting SM over
24 }wamCurrentState; //variable used to track the current state
25
26 static wamDisplay_point_t touchedPoint; //global variable used store touch data
27 static uint8_t z; //needed to meet parameters of
    getTouchedPoint function
28 static uint16_t period, maxMisses, maxMoles; //variables which will be set by the main
    function
29 static uint32_t randomSeed;
30 static bool gameOver; //used by main to determine if the game
    has been completed
31
32 // Call this before using any wamControl_ functions.
33 void wamControl_init(){
34     wamCurrentState = init_st; //SM should always start in this state
35     gameOver = false; //assume the game is not over at the beginning
36     srand(randomSeed); //this needs only be done once for the whole program
37 }
38
39 // Call this to set how much time is consumed by each tick of the controlling state
    machine.
40 // This information makes it possible to set the awake and sleep time of moles in ms,
    not ticks.
41 void wamControl_setMsPerTick(uint16_t msPerTick){
42     period = msPerTick;
43 }
44
45 // This returns the time consumed by each tick of the controlling state machine.
46 uint16_t wamControl_getMsPerTick(){
47     return period;
48 }
49
50 // Standard tick function.
51 void wamControl_tick(){
52     //switch statement for transition actions and state changes
53     switch(wamCurrentState){
54     case init_st:
```

wamControl.c

```

55     wamCurrentState = waiting_for_touch_st; //go immediately to the next state. No
other actions needed
56     break;
57     case waiting_for_touch_st:
58         if(wamDisplay_getMissScore() == maxMisses){ //first check if the user has
reached the max amount of misses
59             wamDisplay_drawGameOverScreen(); //if we get into the if statement, the
game is over. Draw the game over screen
60             gameOver = true; //make sure to mark this high so main knows the game is
over
61             wamCurrentState = end_st; //go to the end_st
62         }
63         else if(display_isTouched()){ //if the max miss count has not been reached and
the display is touched
64             display_clearOldTouchData(); //always clear the old touch data
65             wamDisplay_updateAllMoleTickCounts(); //call this function to draw/erase
the moles as is necessary
66             wamCurrentState = debounce_st;
67         }
68         break;
69     case debounce_st:
70         wamDisplay_updateAllMoleTickCounts(); //there has been a tick during gameplay,
so make sure to call this
71         display_getTouchedPoint(&touchedPoint.x, &touchedPoint.y, &z); //use display
function to get touch coordinates
72         wamDisplay_whackMole(&touchedPoint); //call this to see if the user hit an
active mole
73         wamCurrentState = waiting_for_touch_st; //go back to waiting state
74         break;
75     case end_st:
76         if(display_isTouched()){ //we only start the SM back over if the game over
screen is touched
77             gameOver = false; //reset the variables
78             touchedPoint.x = MIN;
79             touchedPoint.y = MIN;
80             z = MIN;
81             wamCurrentState = init_st; //go back to beginning of SM
82         }
83         break;
84     default:
85         break;
86 }
87
88 //switch statement for state actions
89 switch(wamCurrentState){
90     case init_st: //nothing to be done in init_st
91         break;
92     case waiting_for_touch_st:
93         if(wamDisplay_getActiveMoleCount() < maxMoles){ //if the current active mole
amount is less than the permitted max
94             wamDisplay_activateRandomMole(); //activate another mole
95         }
96         wamDisplay_updateAllMoleTickCounts(); //we have to update the tick counts
97         break;
98     case debounce_st: //nothing to be done in debounce_st
99         break;
100    case end_st: //nothing to be done in end_st
101        break;

```

wamControl.c

```
102     default:
103         break;
104     }
105 }
106
107 // Returns a random value that indicates how long the mole should sleep before
    awaking.
108 wamDisplay_moleTickCount_t wamControl_getRandomMoleAsleepInterval() {
109     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
110     if(count <= MIN) count = MIN_COUNT; //if the tick count gets below zero, set it
        back to 4 as the minimum
111
112     return count; //return this randomly generated tick count
113 }
114
115 // Returns a random value that indicates how long the mole should stay awake before
    going dormant.
116 wamDisplay_moleTickCount_t wamControl_getRandomMoleAwakeInterval() {
117     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
118     if(count <= MIN) count = MIN_COUNT; //if the tick count gets below zero, set it
        back to 4 as the minimum
119
120     return count; //return this randomly generated tick count
121 }
122
123 // Set the maximum number of active moles.
124 void wamControl_setMaxActiveMoles(uint16_t count) {
125     maxMoles = count;
126 }
127
128 // Get the current allowable count of active moles.
129 uint16_t wamControl_getMaxActiveMoles() {
130     return wamDisplay_getActiveMoleCount();
131 }
132
133 // Set the seed for the random-number generator.
134 void wamControl_setRandomSeed(uint32_t seed) {
135     randomSeed = seed;
136 }
137
138 // Set the maximum number of misses until the game is over.
139 void wamControl_setMaxMissCount(uint16_t missCount) {
140     maxMisses = missCount;
141 }
142
143 // Use this predicate to see if the game is finished.
144 bool wamControl_isGameOver() {
145     return gameOver;
146 }
147
148
```

wamDisplay.c

```
1 #include "supportFiles/display.h" //needed to use display functions
2 #include "wamDisplay.h"
3 #include "wamControl.h"
4 #include <math.h>
5
6 #define NINE_MOLES 9 //used to create 9 mole board
7 #define SIX_MOLES 6 //used to create 6 mole board
8 #define FOUR_MOLES 4 //used to create 4 mole board
9 #define INC 1 //used to increment the hits, misses, and level
10 #define NEW_LEVEL 5 //used with modulo to determine if the player moves to the next
    level
11
12 //these are used as indices for each mole in the array of moles
13 #define MOLE_0 0
14 #define MOLE_1 1
15 #define MOLE_2 2
16 #define MOLE_3 3
17 #define MOLE_4 4
18 #define MOLE_5 5
19 #define MOLE_6 6
20 #define MOLE_7 7
21 #define MOLE_8 8
22 #define DIDNT_TOUCH_MOLE -1 //this is used if the user didn't touch a mole
23
24 //used to identify the rows of the board
25 #define ROW_0 0
26 #define ROW_1 1
27 #define ROW_2 2
28
29 #define TEXT_SIZE_1 3 //bigger text size
30 #define TEXT_SIZE_2 2 //smaller text size
31
32 //start screen message cursor coordinates
33 #define CURSOR_1_X (DISPLAY_WIDTH/2)-125
34 #define CURSOR_1_Y (DISPLAY_HEIGHT/2)-15
35 #define CURSOR_2_X CURSOR_1_X-15
36 #define CURSOR_2_Y (DISPLAY_HEIGHT/2)+15
37
38 #define X_ORIGIN 0 //needed to set cursor for hits, misses, and levels information. As
    well as drawing mole board
39 #define Y_ORIGIN 0 //used for drawing mole board
40
41 //gameplay info cursor coordinates
42 #define INGAME_CURSOR_Y DISPLAY_HEIGHT-20
43 #define INGAME_CURSOR_X2 (DISPLAY_WIDTH/8)+7
44 #define INGAME_CURSOR_X3 (DISPLAY_WIDTH/4)+16
45 #define INGAME_CURSOR_X4 (DISPLAY_WIDTH/2)-5
46 #define INGAME_CURSOR_X5 (DISPLAY_WIDTH/2)+48
47 #define INGAME_CURSOR_X6 7*DISPLAY_WIDTH/8
48
49 //game over screen message cursor coordinates
50 #define CURSOR_3_X (DISPLAY_WIDTH/2)-80
51 #define CURSOR_3_Y (DISPLAY_HEIGHT/2)-40
52 #define CURSOR_4_X (DISPLAY_WIDTH/2)-35
53 #define CURSOR_4_Y (DISPLAY_HEIGHT/2)
54 #define CURSOR_5_X (DISPLAY_WIDTH/2)-55
55 #define CURSOR_5_Y (DISPLAY_HEIGHT/2)+20
56 #define CURSOR_6_Y (DISPLAY_HEIGHT/2)+40
```

wamDisplay.c

```

57 #define CURSOR_7_Y (DISPLAY_HEIGHT/2)+80
58 //x coordinate cursor values specifically for the number of misses and hits, and final
   level
59 #define INFO_CURSOR_1_X (DISPLAY_WIDTH/2)+10
60 #define INFO_CURSOR_2_X (DISPLAY_WIDTH/2)+13
61 #define INFO_CURSOR_3_X (DISPLAY_WIDTH/2)+25
62
63 //Start screen messages
64 #define START_SCREEN_LINE_1 "Whack a Mole!"
65 #define START_SCREEN_LINE_2 "Touch Screen To Start"
66
67 //Game over screen messages
68 #define END_SCREEN_LINE_1 "Game Over"
69 #define HIT_COUNT "Hits:"
70 #define MISS_COUNT "Misses:"
71 #define FINAL_LEVEL "Final Level:"
72 #define RETRY_MESSAGE "(Touch to Try Again)"
73
74 //real time gameplay info strings
75 #define GAMEPLAY_MISS "Miss:"
76 #define GAMEPLAY_HIT "Hit:"
77 #define GAMEPLAY_LEVEL "Level:"
78
79 #define MAX_ROWS 3 //max amount of rows the game board could have
80 #define MAX_COLS 3 //max amount of columns the game board could have
81 #define MIN_ROWS 2 //minimum amount of rows the game board could have
82 #define MIN_COLS 2 //minimum amount of columns the game board could have
83
84 #define MIN 0 //used for initializing various variables and for knowing where
   we are in the array of moles
85 #define SECOND_ENTRY 1 //used to identify where in the array of moles we are
86 #define POWER 2 //used for scaling when assigning origin coordinates to moles.
   Lessens the need to hardcode certain parts
87
88 #define CIRCLE_0_X DISPLAY_WIDTH/10 //x-coordinate for the circles in first column
89 #define RIGHT_COL_CIR_X CIRCLE_0_X*9
90 #define BOT_CIR_Y DISPLAY_HEIGHT-62 //y-coordinate for the circles in the bottom
   row
91 #define TOP_CIR_Y Y_ORIGIN+32 //y-coordinate for circles in top row
92 #define CIR_RADIUS 25 //radius of circles
93 #define Y_OFFSET 30 //needed to make sure there is space for the
   gameplay info under the drawn board
94 #define HALF_WIDTH DISPLAY_WIDTH/2 //needed for x-coordinate of circles in middle
   column
95 #define MID_CIR_Y (DISPLAY_HEIGHT/2)-16 //y-coordinate for circles in middle row
96
97 /***** typedefs *****/
98 // This keeps track of all mole information.
99 typedef struct {
100     wamDisplay_point_t origin; // This is the origin of the hole for this mole.
101     // A mole is active if either of the tick counts are non-zero. The mole is dormant
   otherwise.
102     // During operation, non-zero tick counts are decremented at a regular rate by the
   control state machine.
103     // The mole remains in his hole until ticksUntilAwake decrements to zero and then
   he pops out.
104     // The mole remains popped out of his hole until ticksUntilDormant decrements to
   zero.

```

wamDisplay.c

```

105 // Once ticksUntilDomant goes to zero, the mole hides in his hole and remains
    dormant until activated again.
106 wamDisplay_moleTickCount_t ticksUntilAwake; // Mole will wake up (pop out of
    hole) when this goes from 1 -> 0.
107 wamDisplay_moleTickCount_t ticksUntilDormant; // Mole will go dormant (back in
    hole) this goes 1 -> 0.
108 } wamDisplay_moleInfo_t;
109
110 // This will contain pointers to all of the mole info records.
111 // This will ultimately be treated as an array of pointers.
112 static wamDisplay_moleInfo_t** wamDisplay_moleInfo;
113
114 static uint8_t moleAmount; //number of moles based on data from switches
115 static uint8_t numRows; //this and numCols are assigned at same time as moleAmount.
    Prevents need for excessive switch statements
116 static uint8_t numCols;
117
118 static uint16_t activeMoles; //used to track how many moles are currently active
119 static uint16_t hitCount, missCount, level; //variables to store the gameplay
    information
120
121 // Allocates the memory for wamDisplay_moleInfo_t records.
122 // Computes the origin for each mole assuming a simple row-column layout:
123 // 9 moles: 3 rows, 3 columns, 6 moles: 2 rows, 3 columns, 4 moles: 2 rows, 2 columns
124 // Also inits the tick counts for awake and dormant.
125 void wamDisplay_computeMoleInfo() {
126     // Setup all of the moles, creates and inits mole info records.
127     // Create the container array. It contains pointers to each of the mole-hole info
    records.
128     wamDisplay_moleInfo = (wamDisplay_moleInfo_t**) malloc(moleAmount *
        sizeof(wamDisplay_moleInfo_t)); //initialization of pointer to a pointer. memory must
    be allocated
129     for(uint8_t i = 0; i<moleAmount; i++) //for loop used to allocate memory for a
        pointer to each mole. This makes our pointer to a pointer become a pointer to an array
    of pointers
130         wamDisplay_moleInfo[i] = (wamDisplay_moleInfo_t*)
            malloc(sizeof(wamDisplay_moleInfo_t));
131
132     uint8_t counter = MIN; //used to keep track of iteration through array of mole
    pointers
133     for(uint8_t r = 0; r<numRows; r++){ //iterate through rows
134         for(uint8_t c = 0; c<numCols; c++){ //iterate through columns
135             wamDisplay_moleInfo[counter]->ticksUntilAwake = MIN; //initialize the
                ticks until awake and ticks until dormant of each mole
136             wamDisplay_moleInfo[counter]->ticksUntilDormant = MIN;
137             //if the board has 4 moles and we're in the second column in our iteration
138             if(numCols == MIN_COLS && c == SECOND_ENTRY)
                wamDisplay_moleInfo[counter]->origin.x = RIGHT_COL_CIR_X; //set the x-coordinate to
                the corresponding value
139             else if(c == SECOND_ENTRY) wamDisplay_moleInfo[counter]->origin.x =
                HALF_WIDTH; //or if it's not a 4 mole board, we're in the middle column
140             else if(c == MIN) wamDisplay_moleInfo[counter]->origin.x = CIRCLE_0_X;
                //or if it's the first column, regardless of size of board, the x-coordinate will be
                the same
141             else wamDisplay_moleInfo[counter]->origin.x = RIGHT_COL_CIR_X; //or if
                it's the last column and it's a 6 mole or 9 mole board
142
143             //if it's a 4 mole or 6 mole board and we're in the bottom row in our

```

wamDisplay.c

```

iteration
144         if(numRows == MIN_ROWS && r == SECOND_ENTRY)
wamDisplay_moleInfo[counter]->origin.y = BOT_CIR_Y; //set the y-coordinate to the
corresponding value
145         else if(r == SECOND_ENTRY) wamDisplay_moleInfo[counter]->origin.y =
MID_CIR_Y; //of it's a 9 mole board and we're in the second row, it's the middle row
146         else if(r == MIN) wamDisplay_moleInfo[counter]->origin.y = TOP_CIR_Y; //or
if we're in the top row, regardless of the size of board, the y-coordinate will be the
same
147         else wamDisplay_moleInfo[counter]->origin.y = BOT_CIR_Y; //or if it's the
bottom row and it's a 9 mole board
148
149         counter++; //increment to move to the next mole
150     }
151 }
152 }
153
154
155 // Provide support to set games with varying numbers of moles. This function
156 // would be called prior to calling wamDisplay_init();
157 void wamDisplay_selectMoleCount(wamDisplay_moleCount_e moleCount){
158     switch(moleCount){ //gets mole count from switch data
159         case wamDisplay_moleCount_9:
160             moleAmount = NINE_MOLES; //if mole count is 9, then we set this helper global
variable to 9
161             numCols = MAX_COLS; //9 moles means 3 columns
162             numRows = MAX_ROWS; //and 3 rows
163             break;
164         case wamDisplay_moleCount_6:
165             moleAmount = SIX_MOLES; //if mole count is 6, then we set this helper global
variable to 6
166             numCols = MAX_COLS; //6 moles means 3 columns
167             numRows = MIN_ROWS; //and 2 rows
168             break;
169         case wamDisplay_moleCount_4:
170             moleAmount = FOUR_MOLES; //if mole count is 4, then we set this helper global
variable to 4
171             numCols = MIN_COLS; //4 moles means 2 columns
172             numRows = MIN_ROWS; //and 2 rows
173             break;
174         default:
175             break;
176     }
177
178     wamDisplay_computeMoleInfo(); //due to main not calling this function, we do it
here
179 }
180
181 // Call this before using any wamDisplay_ functions.
182 void wamDisplay_init(){
183     hitCount = MIN; //initialize the necessary variables
184     missCount = MIN;
185     level = MIN;
186     activeMoles = MIN;
187 }
188
189 // Draw the game display with a background and mole holes.
190 void wamDisplay_drawMoleBoard(){

```

wamDisplay.c

```

191     display_fillScreen(DISPLAY_BLACK);
192
193     display_fillRect(X_ORIGIN, Y_ORIGIN, DISPLAY_WIDTH, DISPLAY_HEIGHT-Y_OFFSET,
        DISPLAY_GREEN); //draw the green rectangle that will take up most of the screen
194     for(uint8_t i = 0; i<moleAmount; i++) //for loop for drawing each mole hole in the
        board
195         display_fillCircle(wamDisplay_moleInfo[i]->origin.x,
            wamDisplay_moleInfo[i]->origin.y, CIR_RADIUS, DISPLAY_BLACK);
196
197     wamDisplay_drawScoreScreen(); //use this other helper function to draw the
        gameplay info under the board
198
199 }
200
201 // Draw the initial splash (instruction) screen.
202 void wamDisplay_drawSplashScreen() {
203     display_fillScreen(DISPLAY_BLACK);
204     //set cursor, text color, and text size for first line of start message
205     display_setCursor(CURSOR_1_X,CURSOR_1_Y);
206     display_setTextColor(DISPLAY_WHITE);
207     display_setTextSize(TEXT_SIZE_1);
208     //actually print first line of start message
209     display_println(START_SCREEN_LINE_1);
210
211     //set cursor and text size for second line of start message
212     display_setCursor(CURSOR_2_X,CURSOR_2_Y);
213     display_setTextSize(TEXT_SIZE_2);
214     //actually print second line of start message
215     display_println(START_SCREEN_LINE_2);
216 }
217
218 // Draw the game-over screen.
219 void wamDisplay_drawGameOverScreen() {
220     display_fillScreen(DISPLAY_BLACK);
221
222     display_setTextColor(DISPLAY_WHITE); //make sure text color is white
223
224     //set cursor and text size for line 1, then print line 1
225     display_setCursor(CURSOR_3_X,CURSOR_3_Y);
226     display_setTextSize(TEXT_SIZE_1);
227     display_println(END_SCREEN_LINE_1);
228
229     //set correct cursor location and text size, then print hit count
230     display_setCursor(CURSOR_4_X,CURSOR_4_Y);
231     display_setTextSize(TEXT_SIZE_2);
232     display_print(HIT_COUNT);
233     display_print(hitCount);
234
235     //set new cursor locations, then print miss count
236     display_setCursor(CURSOR_5_X,CURSOR_5_Y);
237     display_print(MISS_COUNT);
238     display_print(missCount);
239
240     //set new cursor location, then print final level reached
241     display_setCursor(CURSOR_3_X,CURSOR_6_Y);
242     display_print(FINAL_LEVEL);
243     display_print(level);
244

```


wamDisplay.c

```

245 //set new cursor location and print the retry message
246 display_setCursor(CURSOR_1_X,CURSOR_7_Y);
247 display_println(RETRY_MESSAGE);
248 }
249
250 // Selects a random mole and activates it.
251 // Activating a mole means that the ticksUntilAwake and ticksUntilDormant counts are
    initialized.
252 // See the comments for wamDisplay_moleInfo_t for details.
253 // Returns true if a mole was successfully activated. False otherwise. You can
254 // use the return value for error checking as this function should always be
    successful
255 // unless you have a bug somewhere.
256 bool wamDisplay_activateRandomMole(){
257     bool activated = false; //use this to know we've successfully activated a mole
258     while(!activated){ //while loop to go until we've activated a mole
259         uint8_t randomMole;
260         randomMole = rand() % (moleAmount); //get a random mole index
261         if(wamDisplay_moleInfo[randomMole]->ticksUntilAwake == MIN &&
            wamDisplay_moleInfo[randomMole]->ticksUntilDormant == MIN){ //this is to make sure the
            mole we picked is inactive
262             wamDisplay_moleInfo[randomMole]->ticksUntilAwake =
            wamControl_getRandomMoleAsleepInterval(); //give that mole a randomly generated asleep
            tick count
263             wamDisplay_moleInfo[randomMole]->ticksUntilDormant =
            wamControl_getRandomMoleAwakeInterval(); //give that mole a randomly generated awake
            tick count
264             activated = true; //mark this high since we succeeded
265             activeMoles++; //make sure we track the activated mole amount
266         }
267     }
268
269     return activated; //return the boolean
270 }
271 //helper function to determine which mole was touched
272 int16_t wamDisplay_getWhichRow(wamDisplay_coord_t y){
273     if(y > (BOT_CIR_Y-CIR_RADIUS) && y < (BOT_CIR_Y+CIR_RADIUS)) return ROW_2; //use
        bottom circle origin coordinates and the radius to see if it's in the bottom row
274     if(y > (MID_CIR_Y-CIR_RADIUS) && y < (MID_CIR_Y+CIR_RADIUS)) return ROW_1; //use
        middle circle origin coordinates and radius to see if in middle row
275     if(y > (TOP_CIR_Y-CIR_RADIUS) && y < (TOP_CIR_Y+CIR_RADIUS)) return ROW_0; //use
        top circle origin coordinates and radius to see if in top row
276
277     return DIDNT_TOUCH_MOLE; //if it hasn't returned yet, the touch was in an area
        between rows
278 }
279
280 //helper function to see which mole was touched
281 int16_t wamDisplay_getWhichMole(wamDisplay_point_t* whackOrigin){
282     wamDisplay_coord_t currX = whackOrigin->x; //variables to store coordinate values
283     wamDisplay_coord_t currY = whackOrigin->y;
284     int16_t row = wamDisplay_getWhichRow(currY); //call the row helper function
285
286     if(currX > (CIRCLE_0_X-CIR_RADIUS) && currX < (CIRCLE_0_X+CIR_RADIUS)){ //if it's
        in the first column
287         if(row == ROW_0) return MOLE_0; //and first row, it's the first mole
288         if(row == ROW_1) return MOLE_3; //or second row, it's the fourth mole
289         if(row == ROW_2) return MOLE_6; //or third row, it's the seventh mole

```

wamDisplay.c

```

290     }
291     if(currX > (HALF_WIDTH-CIR_RADIUS) && currX < (HALF_WIDTH+CIR_RADIUS)){ //if it's
    in the the second column
292         if(row == ROW_0) return MOLE_1; //and first row, it's the second mole
293         if(row == ROW_1) return MOLE_4; //or second row, it's the fifth mole
294         if(row == ROW_2) return MOLE_7; //or third row, it's the eighth mole
295     }
296     if(currX > (RIGHT_COL_CIR_X-CIR_RADIUS) && currX < (RIGHT_COL_CIR_X+CIR_RADIUS)){
    //if it's in the third column
297         if(row == ROW_0) return MOLE_2; //and first row, it's the third mole
298         if(row == ROW_1) return MOLE_5; //or second row, it's the sixth mole
299         if(row == ROW_2) return MOLE_8; //or third row, it's the ninth mole
300     }
301
302     return DIDNT_TOUCH_MOLE; //otherwise, they didn't touch a mole
303 }
304
305 // This takes the provided coordinates and attempts to whack a mole. If a
306 // mole is successfully whacked, all internal data structures are updated and
307 // the display and score is updated. You can only whack a mole if the mole is awake
    (visible).
308 // The return value can be used during testing (you could just print which mole is
309 // whacked without having to implement the entire game).
310 wamDisplay_moleIndex_t wamDisplay_whackMole(wamDisplay_point_t* whackOrigin){
311     int16_t mole = wamDisplay_getWhichMole(whackOrigin); //get which mole was touched
312     switch(moleAmount){ //switch statement for different board sizes
313     case FOUR_MOLES: //if four mole board
314         if(mole == MOLE_0) mole = MOLE_0; //if first mole, index stays the same
315         else if(mole == MOLE_2) mole = MOLE_1; //if third mole, index changes to
    second in array
316         else if(mole == MOLE_6) mole = MOLE_2; //if seventh mole, index changes to
    third in array
317         else if(mole == MOLE_8) mole = MOLE_3; //if ninth mole, index changes to
    fourth in array
318         else mole = DIDNT_TOUCH_MOLE; //otherwise there was an erroneous touch
319         break;
320     case SIX_MOLES: //if six mole board
321         if(mole == MOLE_0) mole = MOLE_0; //if first mole, index stays the same
322         else if(mole == MOLE_1) mole = MOLE_1; //if second mole, index stays same
323         else if(mole == MOLE_2) mole = MOLE_2; //if third mole, index stays same
324         else if(mole == MOLE_6) mole = MOLE_3; //if seventh mole, index changes to 4th
    mole in array
325         else if(mole == MOLE_7) mole = MOLE_4; //if eighth mole, index changes to 5th
    mole in array
326         else if(mole == MOLE_8) mole = MOLE_5; //if ninth mole, index changes to 6th
    mole in array
327         else mole = DIDNT_TOUCH_MOLE; //otherwise there was an erroneous touch
328         break;
329     default:
330         break;
331     }
332
333     if(mole == DIDNT_TOUCH_MOLE) return mole; //if there was an erroneous touch not on
    a mole, return
334
335     wamDisplay_moleInfo_t* currMole = wamDisplay_moleInfo[mole]; //get the current
    mole
336

```

wamDisplay.c

```

337     if(currMole->ticksUntilDormant != MIN && currMole->ticksUntilAwake == MIN){ //if
the mole theyr touched is active
338         display_fillCircle(currMole->origin.x, currMole->origin.y, CIR_RADIUS,
DISPLAY_BLACK); //erase the mole they touched
339         currMole->ticksUntilDormant = MIN; //reset the ticksUntilDormant
340         wamDisplay_setHitScore(hitCount+INC); //increment the hit count
341         activeMoles--; //that mole is no longer active, so we've gotta decrement this
variable
342     }
343
344     return mole; //return the index of the mole that was touched
345 }
346
347 // This updates the ticksUntilAwake/ticksUntilDormant clocks for all of the moles.
348 void wamDisplay_updateAllMoleTickCounts() {
349     for(uint8_t i = 0; i < moleAmount; i++){ //iterate through the moles array
350         if(wamDisplay_moleInfo[i]->ticksUntilAwake != MIN){ //if the mole is asleep
waiting to wake up
351             wamDisplay_moleInfo[i]->ticksUntilAwake--; //update tick count
352             if(wamDisplay_moleInfo[i]->ticksUntilAwake == MIN) //if it's time for it
to wake up
353                 display_fillCircle(wamDisplay_moleInfo[i]->origin.x,
wamDisplay_moleInfo[i]->origin.y, CIR_RADIUS, DISPLAY_RED); //draw the mole
354             }
355             else if(wamDisplay_moleInfo[i]->ticksUntilDormant != MIN){ //if the mole is
awake waiting to go to sleep
356                 wamDisplay_moleInfo[i]->ticksUntilDormant--; //update tick count
357                 if(wamDisplay_moleInfo[i]->ticksUntilDormant == MIN){ //if it's time for
it to go to sleep
358                     display_fillCircle(wamDisplay_moleInfo[i]->origin.x,
wamDisplay_moleInfo[i]->origin.y, CIR_RADIUS, DISPLAY_BLACK); //erase corresponding
mole
359                     wamDisplay_setMissScore(missCount+INC); //update miss count
360                     activeMoles--; //mole no longer active, so update this variable
361                 }
362             }
363         }
364     }
365
366 // Returns the count of currently active moles.
367 // A mole is active if it is not dormant, if:
368 // ticksUntilAwake or ticksUntilDormant are non-zero (in the moleInfo_t struct).
369 uint16_t wamDisplay_getActiveMoleCount() {
370     return activeMoles;
371 }
372
373 // Sets the hit value in the score window.
374 void wamDisplay_setHitScore(uint16_t hits){
375     display_setCursor(INGAME_CURSOR_X2, INGAME_CURSOR_Y); //set correct cursor
376     display_setTextColor(DISPLAY_BLACK); //set text to black
377     display_print(hitCount); //erase the previous hit count
378     hitCount = hits; //update the hit count
379
380     //print out new hit count
381     display_setCursor(INGAME_CURSOR_X2, INGAME_CURSOR_Y);
382     display_setTextColor(DISPLAY_WHITE);
383     display_print(hitCount);
384

```

wamDisplay.c

```
385     if((hitCount % NEW_LEVEL) == MIN) //if the player has gotten another 5 hits
386         wamDisplay_incrementLevel(); //they move on to the next level
387 }
388
389 // Gets the current hit value.
390 uint16_t wamDisplay_getHitScore() {
391     return hitCount;
392 }
393
394 // Sets the miss value in the score window.
395 void wamDisplay_setMissScore(uint16_t misses) {
396     display_setCursor(INGAME_CURSOR_X4, INGAME_CURSOR_Y); //set correct cursor
397     display_setTextColor(DISPLAY_BLACK); //need text to be black
398     display_print(missCount); //erase the previous miss count
399     missCount = misses; //update miss count
400
401     //reset cursor and print out new miss count
402     display_setCursor(INGAME_CURSOR_X4, INGAME_CURSOR_Y);
403     display_setTextColor(DISPLAY_WHITE);
404     display_print(missCount);
405 }
406
407 // Gets the miss value.
408 // Can be used for testing and other functions.
409 uint16_t wamDisplay_getMissScore() {
410     return missCount;
411 }
412
413 // Sets the level value on the score board.
414 void wamDisplay_incrementLevel() {
415     if(wamControl_getMaxActiveMoles() < moleAmount) //make sure we don't have more
        active moles than the board has space for
416         wamControl_setMaxActiveMoles(wamControl_getMaxActiveMoles()+INC); //add to max
        active moles to increase level difficulty
417
418     display_setCursor(INGAME_CURSOR_X6, INGAME_CURSOR_Y); //set correct cursor
419     display_setTextColor(DISPLAY_BLACK); //need text to be black
420     display_print(level); //erase previous level
421     level++; //update level
422
423     //set new cursor, correct text color and print new level
424     display_setCursor(INGAME_CURSOR_X6, INGAME_CURSOR_Y);
425     display_setTextColor(DISPLAY_WHITE);
426     display_print(level);
427 }
428
429 // Retrieves the current level value.
430 // Can be used for testing and other functions.
431 uint16_t wamDisplay_getLevel() {
432     return level;
433 }
434
435 // Completely draws the score screen.
436 // This function renders all fields, including the text fields for "Hits" and
    "Misses".
437 // Usually only called once when you are initializing the game.
438 void wamDisplay_drawScoreScreen() {
439     display_setCursor(X_ORIGIN, INGAME_CURSOR_Y ); //set cursor for gameplay info
```

wamDisplay.c

```
440     display_setTextColor(DISPLAY_WHITE); //make sure text color is white
441     display_setTextSize(TEXT_SIZE_2); //we want the smaller text size
442     display_print(GAMEPLAY_HIT); //prints out hits and the actual hit count
443     display_setCursor(INGAME_CURSOR_X2, INGAME_CURSOR_Y);
444     display_print(hitCount);
445
446     //set new cursor, print out "Miss:"
447     display_setCursor(INGAME_CURSOR_X3, INGAME_CURSOR_Y);
448     display_println(GAMEPLAY_MISS);
449     display_setCursor(INGAME_CURSOR_X4, INGAME_CURSOR_Y); //set new cursor, then print
out actual miss count
450     display_print(missCount);
451
452     //set new cursor and print out "Level:"
453     display_setCursor(INGAME_CURSOR_X5, INGAME_CURSOR_Y);
454     display_print(GAMEPLAY_LEVEL);
455     display_setCursor(INGAME_CURSOR_X6, INGAME_CURSOR_Y); //set new cursor and print
actual current level
456     display_print(level);
457 }
458
459 // Make this function available for testing purposes.
460 void wamDisplay_incrementMissScore() {
461
462 }
463
464 // Reset the scores and level to restart the game.
465 void wamDisplay_resetAllScoresAndLevel() {
466     hitCount = MIN; //reset global variables between games
467     missCount = MIN;
468     level = MIN;
469     moleAmount = MIN;
470     numCols = MIN;
471     numRows = MIN;
472     activeMoles = MIN;
473     wamControl_setMaxActiveMoles(INC); //reset max active moles to 1 so level 1 of
next game isn't impossible
474
475     //We HAVE to make sure to free up the memory and don't have the pointers pointing
to anything
476     //this is to prevent costly memory leaks
477     for (uint16_t l=0; l<moleAmount; l++) {
478         free(wamDisplay_moleInfo[l]); // This deallocates the memory.
479         wamDisplay_moleInfo[l] = NULL; // This step is not necessary but will keep
you from reusing deallocated memory.
480     }
481
482     free(wamDisplay_moleInfo); // Deallocates the container of arrays.
483     wamDisplay_moleInfo = NULL; // Also keeps you from reusing the deallocated
memory.
484 }
485
486 // Test function that can be called from main() to demonstrate milestone 1.
487 // Invoking this function should provide the same behavior as shown in the Milestone 1
video.
488 void wamDisplay_runMilestone1_test() {
489     wamDisplay_computeMoleInfo(); //initialize the moles first
490     wamDisplay_drawSplashScreen(); //draw the start message
```

wamDisplay.c

```
491     wamDisplay_drawMoleBoard(); //draw the game board
492     wamDisplay_drawGameOverScreen(); //then draw the game over message
493 }
494
```