```c
#ifndef WAMCONTROL_H_
#define WAMCONTROL_H_

#include "wamDisplay.h"
#include <stdint.h>

// Call this before using any wamControl_ functions.
void wamControl_init();

// Call this to set how much time is consumed by each tick of the controlling state machine.
void wamControl_setMsPerTick(uint16_t msPerTick);

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

// Standard tick function.
void wamControl_tick();

// Returns a random value that indicates how long the mole should sleep before awaking.
wamDisplay_moleTickCount_t wamControl_getRandomMoleAsleepInterval();

// Returns a random value that indicates how long the mole should stay awake before going dormant.
wamDisplay_moleTickCount_t wamControl_getRandomMoleAwakeInterval();

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

// Get the current allowable count of active moles.
uint16_t wamControl_getMaxActiveMoles();

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

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

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

#endif /* WAMCONTROL_H_ */

```

```c
#include "wamControl.h"
#include "wamDisplay.h"
#include "supportFiles/display.h"
#include <stdlib.h>
#include <stdio.h>

//**********MY_VAR**********/
#define ADC_WAIT_TIME 1                 //time to wait in the ADC state
#define FAIL_HIT -1                     //value of a invalid hit
#define INCREMENT_LEVEL_VALUE 4         //increase the level after this many hits
#define RANDOM_NUM_RANGE 20             //the range of number that the random number can be selected from

uint16_t msPerTick = 1;                 //how many milliseconds there are in each tick
uint16_t maxMissCount = 0;              //number of total misses for the game
uint16_t maxActiveMoles = 1;            //number of the total mole that can be active

int16_t x = 0;                          //x coordinate for the touch data
int16_t y = 0;                          //y coordinate for the touch data
uint8_t z = 0;                          //z coordinate for the touch data

uint8_t wamAdcCounter = 0;                 // the counter that keeps track of the number of times that the SM has looped through adc state
wamDisplay_point_t hitCoor;             //VAR to save the touch coodinates

enum wamControl_st_t {                  //the state machine for the control
    wamInit_st,                            //start state
    activate_mole_st,                   //activates a random mole state
    wamAdc_st,                             //delay state
    whack_mole_st,                      //state to regester hit
} wamState = wamInit_st;                   //sets the start state

void wamControl_init(){                 //function to reset all the parameters you want
    wamControl_setMaxActiveMoles(true);    //resets the activeMole counter
    wamState = wamInit_st;                 //sets the start state once again
}

void wamControl_setMsPerTick(uint16_t setVal){      //sets the number of milliseconds per tick
    msPerTick = setVal;
}

uint16_t wamControl_getMsPerTick(){                 //gets the number of milliseconds per tick
    return msPerTick;
}
void wamControl_tick(){                             //function for the state machine
    switch (wamState){
    case wamInit_st:                                   //start state, just transitions to the activate mole st...
        break;
    case activate_mole_st:                          //state that activate the moles
        if (wamDisplay_getActiveMoleCount() < wamControl_getMaxActiveMoles()){ //if the number of current active moles is less then the max number of moles you can have
            wamDisplay_activateRandomMole();                                    //if that was true then activate a random mole
            break;
```

```
51          }
52          wamDisplay_updateAllMoleTickCounts();                                   //no
  matter what update the mole info
53          break;
54      case wamAdc_st:                                                             //state
  that let the SM settle
55          wamAdcCounter++;
  //increment the wamAdcCounter so that the SM can go to the next state after a couple
  ticks
56          break;
57      case whack_mole_st:                                                         //state
  that lets you hit the mole
58          display_getTouchedPoint(&x, &y, &z);
  //function to get the touch coordinates from the screen
59          hitCoor.x = x;                                                          //saves
  the x coordinate
60          hitCoor.y = y;                                                          //saves
  the y coordinate
61          if (wamDisplay_whackMole(&hitCoor) != FAIL_HIT){                        //if
  the hit wasn't a failed hit then increment the hit count
62              if (wamDisplay_getHitScore() != false &&
  wamDisplay_getHitScore()%INCREMENT_LEVEL_VALUE == false){ //if the hit score hits the
  reaches the increment level value
63                  wamDisplay_incrementLevel();
  //of that is true then increment the level
64              }
65              if (wamDisplay_getLevel() < INCREMENT_LEVEL_VALUE){
  //if the level is less then the level that you need to increment at then increase the
  number of moles
66
  //this is so that there arnt more moles thanlevel
67                  wamControl_setMaxActiveMoles(wamDisplay_getLevel()+1);
  // increase the number of moles
68              }
69          }
70          display_clearOldTouchData();
  //clear the old touch data to prepare for the next touch
71          break;
72      }
73      switch (wamState){
74      case wamInit_st:                                    //start state, just
  transitions to the activate mole st...
75          wamState = activate_mole_st;                    //transition to the next state
76          break;
77      case activate_mole_st:
78          if (display_isTouched()){                       //if the screen is touched
  then
79              wamState = wamAdc_st;                       //transition to the next state
80          }
81          break;
82      case wamAdc_st:
83          if (wamAdcCounter >= ADC_WAIT_TIME && display_isTouched()){ //if the counter
  is finished and the display is touched then enter loop
84              wamAdcCounter = false;                                  //reset the
  counter
85              wamState = whack_mole_st;               //transition to the next state
86          }
87          else{
```

```
88              display_clearOldTouchData();            //clear the old touch data just incase its still there
89              wamAdcCounter = false;                                          //reset the counter
90              wamState = activate_mole_st;                //transition to the next state
91          }
92          break;
93      case whack_mole_st:
94          wamState = activate_mole_st;                //transition to the next state
95          break;
96      default:
97          break;
98      }
99  }
100 wamDisplay_moleTickCount_t wamControl_getRandomMoleAsleepInterval(){
101     return (rand()%(RANDOM_NUM_RANGE) +RANDOM_NUM_RANGE) - wamDisplay_getLevel(); //get a random number for time asleep and short it based off the level
102 }
103
104 wamDisplay_moleTickCount_t wamControl_getRandomMoleAwakeInterval(){
105     return  (rand()%(RANDOM_NUM_RANGE) +RANDOM_NUM_RANGE) - wamDisplay_getLevel();//get a random number for time awake and short it based off the level
106 }
107
108 void wamControl_setMaxActiveMoles(uint16_t count){
109     maxActiveMoles = count;                                                 //set the max active mole count
110 }
111
112 uint16_t wamControl_getMaxActiveMoles(){
113     return maxActiveMoles;                                                  //get the max active mole count
114 }
115
116 void wamControl_setRandomSeed(uint32_t seed){
117     srand(seed);                                                            //set the seed for the random number
118 }
119
120 void wamControl_setMaxMissCount(uint16_t missCount){                        // Set the maximum number of misses until the game is over.
121     maxMissCount = missCount;                                               //set the number of miss that you can have before the game ends
122 }
123
124 bool wamControl_isGameOver(){
    //Function to see if the game is over.
125     if (maxMissCount == wamDisplay_getMissScore()){
    //if the number of user misses equals the max number of misses then end the game
126         wamControl_setMaxActiveMoles(true);
    //reset the max active moles
127         return true;
    //say that the game is over
128     }
129     return false;
130 }
131
```

132

```c
#ifndef WAMDISPLAY_H_
#define WAMDISPLAY_H_

#include <stdint.h>
#include <stdbool.h>

// The index used to tell the display which mole is active/inactive.
typedef int16_t wamDisplay_moleIndex_t;
// Mole coordinates represented by this.
typedef int16_t wamDisplay_coord_t;
// Mole origins, whack coordinates, etc. are represented by this.
typedef struct {wamDisplay_coord_t x, y;} wamDisplay_point_t;
// Represents a mole clock interval (asleep or awake).
typedef uint32_t wamDisplay_moleTickCount_t;

// Make it possible to select games with different numbers of moles.
// The display will determine the actual layout. Moles are always indexed by a single integer.
typedef enum {wamDisplay_moleCount_9, wamDisplay_moleCount_6, wamDisplay_moleCount_4} wamDisplay_moleCount_e;

// Provide support to set games with varying numbers of moles. This function
// would be called prior to calling wamDisplay_init();
void wamDisplay_selectMoleCount(wamDisplay_moleCount_e moleCount);

// Call this before using any wamDisplay_ functions.
void wamDisplay_init();

// Draw the game display with a background and mole holes.
void wamDisplay_drawMoleBoard();

// Draw the initial splash (instruction) screen.
void wamDisplay_drawSplashScreen();

// Draw the game-over screen.
void wamDisplay_drawGameOverScreen();

// Selects a random mole and activates it.
// Activating a mole means that the ticksUntilAwake and ticksUntilDormant counts are initialized.
// See the comments for wamDisplay_moleInfo_t for details.
// Returns true if a mole was successfully activated. False otherwise. You can
// use the return value for error checking as this function should always be successful
// unless you have a bug somewhere.
bool wamDisplay_activateRandomMole();

// This takes the provided coordinates and attempts to whack a mole. If a
// mole is successfully whacked, all internal data structures are updated and
// the display and score is updated. You can only whack a mole if the mole is awake (visible).
// The return value can be used during testing (you could just print which mole is
// whacked without having to implement the entire game).
wamDisplay_moleIndex_t wamDisplay_whackMole(wamDisplay_point_t* whackOrigin);

// This updates the ticksUntilAwake/ticksUntilDormant clocks for all of the moles.
void wamDisplay_updateAllMoleTickCounts();

// Returns the count of currently active moles.
```

```
55 // A mole is active if it is not dormant, if:
56 // ticksUntilAwake or ticksUntilDormant are non-zero (in the moleInfo_t struct).
57 uint16_t wamDisplay_getActiveMoleCount();
58
59 // Sets the hit value in the score window.
60 void wamDisplay_setHitScore(uint16_t hits);
61
62 // Gets the current hit value.
63 uint16_t wamDisplay_getHitScore();
64
65 // Sets the miss value in the score window.
66 void wamDisplay_setMissScore(uint16_t misses);
67
68 // Gets the miss value.
69 // Can be used for testing and other functions.
70 uint16_t wamDisplay_getMissScore();
71
72 // Sets the level value on the score board.
73 void wamDisplay_incrementLevel();
74
75 // Retrieves the current level value.
76 // Can be used for testing and other functions.
77 uint16_t wamDisplay_getLevel();
78
79 // Completely draws the score screen.
80 // This function renders all fields, including the text fields for "Hits" and "Misses".
81 // Usually only called once when you are initializing the game.
82 void wamDisplay_drawScoreScreen();
83
84 // Make this function available for testing purposes.
85 void wamDisplay_incrementMissScore();
86
87 // Reset the scores and level to restart the game.
88 void wamDisplay_resetAllScoresAndLevel();
89
90 // Test function that can be called from main() to demonstrate milestone 1.
91 // Invoking this function should provide the same behavior as shown in the Milestone 1
   video.
92 void wamDisplay_runMilestone1_test();
93
94 #endif /* WAMDISPLAY_H_ */
95
```

```c
#include "supportFiles/display.h"
#include "wamDisplay.h"
#include "wamControl.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//**********MY_VARS**********/

#define MOLE_HOLE_RADIUS 26 //size of the mole hole
#define MOLE_RADIUS 25      //size of the mole
#define TEXT_SIZE_L 3       //size of the big text
#define TEXT_SIZE_S 2       //size of the small text
#define STR_SIZE 20         //size of the strings that are being used in sprintf

#define BOARD_OFFSET_Y 18       //the size of the space where the score is printed out
#define BOARD_HOLE_OFFSET_X 120 //how big the moles should be from the x board center
#define BOARD_HOLE_OFFSET_Y 65  //how big the moles should be from the y board center

#define BOARD_SIZE_X DISPLAY_WIDTH                      //size of the board in the x direction
#define BOARD_SIZE_Y DISPLAY_HEIGHT - BOARD_OFFSET_Y    //size of the board in the y direction
#define BOARD_ORIGIN_X_Y 0                              //where the board starts from
#define TRUE 1                                          //define BOOL TRUE
#define FALSE 0                                         //define BOOL FALSE or reset
#define FAIL -1                                         //define fail as -1
#define BOARD_NUM_HOLES_4 4                             //num of holes for a 4 board
#define BOARD_NUM_HOLES_6 6                             //num of holes for a 6 board
#define BOARD_NUM_HOLES_9 9                             //num of holes for a 9 board

#define START_MSG_0 "Whack a Mole!"                     //Text for start screen
#define START_MSG_1 "Touch Screen to Start"             //Text for start screen

#define BOARD_STAT_0 "Hits:%d"                          //text for hits with space for numbers
#define BOARD_STAT_1 "Miss:%d"                          //text for misses with space for numbers
#define BOARD_STAT_2 "Level:%d"                         //text for level with space for numbers

#define GAME_OVER_MSG_0 "Game Over"                       //Text for end screen
#define GAME_OVER_MSG_1 "Hits:%d"                         //text for Hits with space for numbers
#define GAME_OVER_MSG_2 "Misses:%d"                       //text for misses with space for numbers
#define GAME_OVER_MSG_3 "Final Level:%d"                  //text for final level with space for numbers
#define GAME_OVER_MSG_4 "(Touch to Try Again)"            //Text for end screen

#define START_MSG_0_X 45                                //x coordinates for start screen message 0
#define START_MSG_1_X 30                                //x coordinates for start screen message 1

#define START_MSG_0_Y 100                                //y coordinates for start screen message 0
```

```
48 #define START_MSG_1_Y 130                                  //y coordinates for start
   screen message 1
49
50 #define BOARD_STAT_0_X 0                                    //x coordinates for hits
   message 1
51 #define BOARD_STAT_1_X 100                                  //x coordinates for misses
   message 1
52 #define BOARD_STAT_2_X 200                                  //x coordinates for level
   message 1
53 #define BOARD_STAT_0_Y DISPLAY_HEIGHT-BOARD_OFFSET_Y       //y coordinates for the line
   of board stats
54
55 #define GAME_OVER_MSG_0_X 80                               //x coordinates for end screen
   message 0
56 #define GAME_OVER_MSG_1_X 125                               //x coordinates for end
   screen message 1
57 #define GAME_OVER_MSG_2_X 115                               //x coordinates for end
   screen message 2
58 #define GAME_OVER_MSG_3_X 85                               //x coordinates for end screen
   message 3
59 #define GAME_OVER_MSG_4_X 45                               //x coordinates for end screen
   message 4
60
61 #define GAME_OVER_MSG_0_Y 50                               //y coordinates for end screen
   message 0
62 #define GAME_OVER_MSG_1_Y 90                               //y coordinates for end screen
   message 1
63 #define GAME_OVER_MSG_2_Y 110                               //y coordinates for end
   screen message 2
64 #define GAME_OVER_MSG_3_Y 130                               //y coordinates for end
   screen message 3
65 #define GAME_OVER_MSG_4_Y 170                               //y coordinates for end
   screen message 4
66
67 #define MOLE_HOLE_0_X (MOLE_HOLE_1_X - BOARD_HOLE_OFFSET_X)     //x coordinate for the
   column 1 of holes
68 #define MOLE_HOLE_1_X DISPLAY_WIDTH/2                        //x coordinate for
   the column 2 of holes
69 #define MOLE_HOLE_2_X (MOLE_HOLE_1_X + BOARD_HOLE_OFFSET_X)     //x coordinate for the
   column 3 of holes
70
71 #define MOLE_HOLE_0_Y (MOLE_HOLE_1_Y - BOARD_HOLE_OFFSET_Y)     //y coordinate for the
   row 1 of holes
72 #define MOLE_HOLE_1_Y (DISPLAY_HEIGHT-BOARD_OFFSET_Y)/2         //y coordinate for the
   row 2 of holes
73 #define MOLE_HOLE_2_Y (MOLE_HOLE_1_Y + BOARD_HOLE_OFFSET_Y)     //y coordinate for the
   row 3 of holes
74
75 #define MOLE_HOLE_0_0 0                                                  //mole hole 0
76 #define MOLE_HOLE_1_0 1                                                  //mole hole 1
77 #define MOLE_HOLE_2_0 2                                                  //mole hole 2
78 #define MOLE_HOLE_0_1 3                                                  //mole hole 3
79 #define MOLE_HOLE_1_1 4                                                  //mole hole 4
80 #define MOLE_HOLE_2_1 5                                                  //mole hole 5
81 #define MOLE_HOLE_0_2 6                                                  //mole hole 6
82 #define MOLE_HOLE_1_2 7                                                  //mole hole 7
83 #define MOLE_HOLE_2_2 8                                                  //mole hole 8
84 #define MOLE_HOLE_FAIL 10                                                //mole hole that does
```

```
   not exist
 85
 86 //**********MY_VAR*********/
 87 uint8_t numMoleHoles = 0;                                    // VAR to update with
   the current number of mole holes
 88 char statStr0[STR_SIZE];                                    // string to hold the
   hits
 89 char statStr1[STR_SIZE];                                    // string to hold the
   misses
 90 char statStr2[STR_SIZE];                                    // string to hold the
   level
 91
 92 char endStatMsgStr1[STR_SIZE];                                   // string to hold
   the hits
 93 char endStatMsgStr2[STR_SIZE];                                   // string to hold
   the misses
 94 char endStatMsgStr3[STR_SIZE];                                   // string to hold
   the level
 95
 96
 97 uint16_t statCount0 = 0;                                    // VAR for hits
 98 uint16_t statCount1 = 0;                                    // VAR for misses
 99 uint16_t statCount2 = 0;                                    // VAR for level
100
101 wamDisplay_point_t moleCoord;                               //VAR to hole the x
   and y coordinates of the touch
102
103
104 //**********PROTOTYPES*********/
105 void wamDisplay_drawAllMoleHoles();                                  //draw
   the all the mole holes
106 void wamDisplay_drawMoleHole(uint8_t moleHoleReion, uint8_t drwHole );          //draw
   one mole hole
107 void wamDisplay_drawMole(wamDisplay_point_t moleReion, uint8_t drwMole);        //draw
   one mole hole
108 void wamDisplay_getMoleCoor(uint16_t index);
   //coordinates for the mole
109
110
111
112
113 /********************* typedefs *********************/
114 // This keeps track of all mole information.
115 typedef struct {
116         wamDisplay_point_t origin;  // This is the origin of the hole for this mole.
117         // A mole is active if either of the tick counts are non-zero. The mole is
   dormant otherwise.
118         // During operation, non-zero tick counts are decremented at a regular rate by
   the control state machine.
119         // The mole remains in his hole until ticksUntilAwake decrements to zero and
   then he pops out.
120         // The mole remains popped out of his hole until ticksUntilDormant decrements
   to zero.
121         // Once ticksUntilDomant goes to zero, the mole hides in his hole and remains
   dormant until activated again.
122         wamDisplay_moleTickCount_t ticksUntilAwake;  // Mole will wake up (pop out of
   hole) when this goes from 1 -> 0.
123         wamDisplay_moleTickCount_t ticksUntilDormant; // Mole will go dormant (back in
```

```
   hole) this goes 1 -> 0.
124 } wamDisplay_moleInfo_t;
125
126 // This will contain pointers to all of the mole info records.
127 // This will ultimately be treated as an array of pointers.
128 static wamDisplay_moleInfo_t** wamDisplay_moleInfo;
129
130 // Allocates the memory for wamDisplay_moleInfo_t records.
131 // Computes the origin for each mole assuming a simple row-column layout:
132 // 9 moles: 3 rows, 3 columns, 6 moles: 2 rows, 3 columns, 4 moles: 2 rows, 2 columns
133 // Also inits the tick counts for awake and dormant.
134 void wamDisplay_computeMoleInfo() {
135     wamDisplay_moleInfo = (wamDisplay_moleInfo_t**) malloc(numMoleHoles *
   sizeof(wamDisplay_moleInfo_t*));  //Initialize moleInfo through MALLOC
136     for(uint16_t i = FALSE; i< numMoleHoles; i++){
137         wamDisplay_moleInfo[i] = (wamDisplay_moleInfo_t*)
   malloc(sizeof(wamDisplay_moleInfo_t));            //give memory to all of the mole
   holes
138     }
139     for (uint16_t j=FALSE; j<numMoleHoles; j++) {
   //initializes the values for origin, awake and dormant for all moles
140         wamDisplay_getMoleCoor(j);
   //get the mole coordinates
141         wamDisplay_moleInfo[j]->origin = moleCoord;
   //initializes the values for origin
142         wamDisplay_moleInfo[j]->ticksUntilAwake = FALSE;
   //initializes the values for awake
143         wamDisplay_moleInfo[j]->ticksUntilDormant = FALSE;
   //initializes the values for dormant
144     }
145 }
146
147 void wamDisplay_selectMoleCount(wamDisplay_moleCount_e moleCount){      //sets the
   mole count depending on the board
148     switch(moleCount){
149     case wamDisplay_moleCount_4:                                        //if the board
   has 4 holes
150         numMoleHoles = BOARD_NUM_HOLES_4;                               //set the mole
   VAR
151         break;
152     case wamDisplay_moleCount_6:                                        //if the board
   has 6 holes
153         numMoleHoles = BOARD_NUM_HOLES_6;                               //set the mole
   VAR
154         break;
155     case wamDisplay_moleCount_9:                                        //if the board
   has 9 holes
156         numMoleHoles = BOARD_NUM_HOLES_9;                               //set the mole
   VAR
157         break;
158     }
159     wamDisplay_computeMoleInfo();                                       //update the
   mole info
160 };
161
162 void wamDisplay_init(){// Call this before using any wamDisplay_ functions.
163     display_setTextColor(DISPLAY_WHITE, DISPLAY_BLACK);                 //reset the
   text color behind the text
```

```
164 };
165
166 void wamDisplay_drawMoleBoard(){// Draw the game display with a background and mole
    holes.
167     display_fillRect(BOARD_ORIGIN_X_Y, BOARD_ORIGIN_X_Y, BOARD_SIZE_X, BOARD_SIZE_Y,
    DISPLAY_GREEN); //draw green background
168     wamDisplay_drawScoreScreen();
    //draw the score bar
169     wamDisplay_drawAllMoleHoles();
    //cut out the holes in the board
170 };
171
172 void wamDisplay_drawSplashScreen(){ // Draw the initial splash (instruction) screen.
173     display_fillScreen(DISPLAY_BLACK);                                  //reset the screen
174     display_setTextColor(DISPLAY_WHITE);                                //set the text
    color
175
176     display_setTextSize(TEXT_SIZE_L);                                   //set the text
    size
177     display_setCursor(START_MSG_0_X , START_MSG_0_Y);                   //set the cursor
    for the text
178     display_println(START_MSG_0);                                       //print the msg
179
180     display_setTextSize(TEXT_SIZE_S);                                    //set the text
    size
181     display_setCursor(START_MSG_1_X, START_MSG_1_Y);                    //set the cursor
    for the text
182     display_println(START_MSG_1);                                       //print the msg
183 };
184
185
186 void wamDisplay_drawGameOverScreen(){ // Draw the game-over screen.
187     display_fillScreen(DISPLAY_BLACK);
188     display_setTextColor(DISPLAY_WHITE);
189
190     sprintf(endStatMsgStr1, GAME_OVER_MSG_1, statCount0);       //combine the message
    and the number of hits
191     sprintf(endStatMsgStr2, GAME_OVER_MSG_2, statCount1);       //combine the message
    and the number of misses
192     sprintf(endStatMsgStr3, GAME_OVER_MSG_3, statCount2);       //combine the message
    and the number of levels
193
194     display_setTextSize(TEXT_SIZE_L);                                   //set the text
    size
195     display_setCursor(GAME_OVER_MSG_0_X , GAME_OVER_MSG_0_Y);               //set the
    cursor for the text
196     display_println(GAME_OVER_MSG_0);                                       //print the
    game over message
197
198     display_setTextSize(TEXT_SIZE_S);                                   //set the text
    size
199     display_setCursor(GAME_OVER_MSG_1_X , GAME_OVER_MSG_1_Y);               //set the
    cursor for the text
200     display_println(endStatMsgStr1);                                        // String
    of the Hits
201
202     display_setCursor(GAME_OVER_MSG_2_X , GAME_OVER_MSG_2_Y);               //set the
    cursor for the text
```

```
203     display_println(endStatMsgStr2);                                    // String
    of the Misses
204
205     display_setCursor(GAME_OVER_MSG_3_X , GAME_OVER_MSG_3_Y);           //set the
    cursor for the text
206     display_println(endStatMsgStr3);                                    // String
    of the End Level
207
208     display_setCursor(GAME_OVER_MSG_4_X , GAME_OVER_MSG_4_Y);           //set the
    cursor for the text
209     display_println(GAME_OVER_MSG_4);                                   //prints
    the "touch to play" msg
210 }
211
212 // Selects a random mole and activates it.
213 // Activating a mole means that the ticksUntilAwake and ticksUntilDormant counts are
    initialized.
214 // See the comments for wamDisplay_moleInfo_t for details.
215 // Returns true if a mole was successfully activated. False otherwise. You can
216 // use the return value for error checking as this function should always be
    successful
217 // unless you have a bug somewhere.
218 bool wamDisplay_activateRandomMole(){
219     uint8_t randMoleHole = rand()%(numMoleHoles); // get random Number for what region
    to activate
220     //if active dont
221     while(wamDisplay_moleInfo[randMoleHole]->ticksUntilAwake == FALSE &&
222             wamDisplay_moleInfo[randMoleHole]->ticksUntilDormant == FALSE){
    //if the mole is not awake and dormant then you can activate it
223         if(wamDisplay_getActiveMoleCount() < wamDisplay_getLevel()+1 ){
    //not let there be more moles than the current level + 1
224
225             wamDisplay_moleInfo[randMoleHole]->ticksUntilAwake =
    wamControl_getRandomMoleAwakeInterval();// get how long the mole will be awake
226             wamDisplay_moleInfo[randMoleHole]->ticksUntilDormant =
    wamControl_getRandomMoleAsleepInterval(); // get how long the mole is asleep
227             return TRUE;
228         }
229         else{
230             return FALSE;
231         }
232     }
233     return FALSE;
234 };
235
236
237 // This takes the provided coordinates and attempts to whack a mole. If a
238 // mole is successfully whacked, all internal data structures are updated and
239 // the display and score is updated. You can only whack a mole if the mole is awake
    (visible).
240 // The return value can be used during testing (you could just print which mole is
241 // whacked without having to implement the entire game).
242 wamDisplay_moleIndex_t wamDisplay_whackMole(wamDisplay_point_t* whackOrigin){
243     for(int i = FALSE;i <numMoleHoles; i++){
244         if (wamDisplay_moleInfo[i]->ticksUntilAwake == FALSE &&
245                     wamDisplay_moleInfo[i]->ticksUntilDormant != FALSE){
246              if(abs(whackOrigin->x - wamDisplay_moleInfo[i]->origin.x ) < MOLE_RADIUS
    && abs(whackOrigin->y - wamDisplay_moleInfo[i]->origin.y ) < MOLE_RADIUS){// if mole
```

```
   is awake and non-dormant
247                 wamDisplay_moleInfo[i]->ticksUntilDormant = FALSE;
   // make the mole dormant
248                 wamDisplay_drawMole(wamDisplay_moleInfo[i]->origin, FALSE);
   // erase the mole
249                 wamDisplay_setHitScore(statCount0+1);
   // increment the number of hits
250                 wamDisplay_drawScoreScreen();
   //update the board stats
251                 return i;
   //return the mole index
252             }
253         }
254     }
255     return FAIL; // the hit failed
256 };
257
258 // This updates the ticksUntilAwake/ticksUntilDormant clocks for all of the moles.
259 void wamDisplay_updateAllMoleTickCounts(){
260     for(int i = FALSE; i<numMoleHoles; i++){
261         if (wamDisplay_moleInfo[i]->ticksUntilAwake > FALSE){          // if it is
   not suppose to be awake then decrement it
262             wamDisplay_moleInfo[i]->ticksUntilAwake--;                 // decrement
   the tick
263             if (wamDisplay_moleInfo[i]->ticksUntilAwake == FALSE){ // when the timer
   hits 0 draw the mole
264                 wamDisplay_drawMole(wamDisplay_moleInfo[i]->origin, TRUE); // draw the
   mole
265             }
266         }
267         else if (wamDisplay_moleInfo[i]->ticksUntilDormant > FALSE){           // now
   that the mole is awake, decrement down the dormant timer
268             wamDisplay_moleInfo[i]->ticksUntilDormant--;                       //
   decrement the dormant counter
269             if (wamDisplay_moleInfo[i]->ticksUntilDormant == FALSE){           //
   check for last second
270                 wamDisplay_drawMole(wamDisplay_moleInfo[i]->origin, FALSE);
   //erase the mole
271                 statCount1++;
   //missed the mole, increment the counter
272                 wamDisplay_drawScoreScreen();                                  //
   update the score
273             }
274         }
275     }
276 };
277
278 // Returns the count of currently active moles.
279 // A mole is active if it is not dormant, if:
280 // ticksUntilAwake or ticksUntilDormant are non-zero (in the moleInfo_t struct).
281 uint16_t wamDisplay_getActiveMoleCount(){
282     uint16_t moleCounter = FALSE;                    //define a mole counter var
283     for (int i = FALSE; i < numMoleHoles; i++){
284         if(wamDisplay_moleInfo[i]->ticksUntilAwake > FALSE ||
   wamDisplay_moleInfo[i]->ticksUntilDormant > FALSE){ //check if the mole[i] is either
   awake or dorment
285             moleCounter++;
   // if so then increment the counter
```

```
286         }
287     }
288     return moleCounter;
    //return the counter
289 };
290 
291 // Sets the hit value in the score window.
292 void wamDisplay_setHitScore(uint16_t hits){
293     statCount0 = hits;
    //set hits
294 };
295 
296 // Gets the current hit value.
297 uint16_t wamDisplay_getHitScore(){
298     return statCount0;
    //get number of hits
299 };
300 
301 // Sets the miss value in the score window.
302 void wamDisplay_setMissScore(uint16_t misses){
303     statCount1 = misses;
    //set misses
304 };
305 
306 // Gets the miss value.
307 // Can be used for testing and other functions.
308 uint16_t wamDisplay_getMissScore(){
309     return statCount1;
    //get miss
310 };
311 
312 // Sets the level value on the score board.
313 void wamDisplay_incrementLevel(){
314     statCount2++;
    //increment level
315 };
316 
317 // Retrieves the current level value.
318 // Can be used for testing and other functions.
319 uint16_t wamDisplay_getLevel(){
320     return statCount2;
    //get the level
321 };
322 
323 // Completely draws the score screen.
324 // This function renders all fields, including the text fields for "Hits" and
    "Misses".
325 // Usually only called once when you are initializing the game.
326 void wamDisplay_drawScoreScreen(){
327     display_setTextSize(TEXT_SIZE_S);                    // set the text size
328     sprintf(statStr0, BOARD_STAT_0, statCount0);       //combine the message and the
    number of hits
329     sprintf(statStr1, BOARD_STAT_1, statCount1);       //combine the message and the
    number of misses
330     sprintf(statStr2, BOARD_STAT_2, statCount2);       //combine the message and the
    number of level
331     display_setCursor(BOARD_STAT_0_X, BOARD_STAT_0_Y);  // set cursor for hit msg
332     display_println(statStr0);                           // print hit msg
```

```
333     display_setCursor(BOARD_STAT_1_X, BOARD_STAT_0_Y);  // set cursor for miss msg
334     display_println(statStr1);                          // print miss msg
335     display_setCursor(BOARD_STAT_2_X, BOARD_STAT_0_Y);  // set cursor for level msg
336     display_println(statStr2);                          // print level msg
337 };
338 
339 // Make this function available for testing purposes.
340 void wamDisplay_incrementMissScore(){
341     statCount1++;
    //increment misses
342 };
343 
344 // Reset the scores and level to restart the game.
345 void wamDisplay_resetAllScoresAndLevel(){
346     statCount0 = FALSE;                     //reset hit var
347     statCount1 = FALSE;                     //reset misses var
348     statCount2 = FALSE;                     //reset level var
349     for (uint16_t l=FALSE; l<numMoleHoles; l++) {
350         free(wamDisplay_moleInfo[l]);   //deallocates the memory for the single mole
351         wamDisplay_moleInfo[l] = NULL;
352     }
353     free(wamDisplay_moleInfo);   // Deallocates arrays memory
354     wamDisplay_moleInfo = NULL;
355 
356 };
357 
358 // Test function that can be called from main() to demonstrate milestone 1.
359 // Invoking this function should provide the same behavior as shown in the Milestone 1
    video.
360 void wamDisplay_runMilestone1_test(){
361     uint8_t splashFlag = FALSE;         //flag to say the main screen was printed
362     uint8_t boardFlag = FALSE;         //flag to say the board screen was printed
363     uint8_t doneFlag = FALSE;         //flag to say the end screen was printed
364     uint8_t endFlag = FALSE;         //flag to say the the whole thing was printed
    once
365     wamDisplay_init();             //init the screen
366     while (TRUE){
367         if (!splashFlag){                       //if main screen has not been printed
    then
368             wamDisplay_drawSplashScreen();      //print the main screen
369             splashFlag = TRUE;                  //raise the flag
370         }
371         if (display_isTouched()){
372             splashFlag = FALSE;                 //lower flag for the next time through
373             while (TRUE){
374                 wamDisplay_drawScoreScreen();
375                 if (!boardFlag){
376                     wamDisplay_drawMoleBoard();      //print the board screen
377                     boardFlag = TRUE;                  //raise the flag
378                 }
379                 if (display_isTouched()){
380                     boardFlag = FALSE;                 //lower flag for the next time
    through
381                     while (TRUE){
382                         if (!doneFlag){
383                             wamDisplay_drawGameOverScreen();           //print the end
    screen
384                             doneFlag = TRUE;                  //raise the flag
```

```
385                         }
386                         if (display_isTouched()){
387                             doneFlag = FALSE;                 //lower flag for the
    next time through
388                             endFlag = TRUE;                  //raise the flag saying
    that the sequence has been printed once
389                             break;
390                         }
391                     }
392                     break;
393                 }
394             }
395         }
396         if(endFlag){                                          // if the 3 screens have
    been printed then exit the loop
397             break;
398         }
399     }
400 };
401
402 // Helper Function to Draw a Mole
403 void wamDisplay_drawMole(wamDisplay_point_t moleReion, uint8_t drwMole){
404     if(drwMole){
405         display_fillCircle(moleReion.x, moleReion.y, MOLE_RADIUS, DISPLAY_RED);
    //if you want to draw the mole then print the circle red
406     }
407     else{
408         display_fillCircle(moleReion.x, moleReion.y, MOLE_RADIUS, DISPLAY_BLACK);
    //if you want to erase the mole then print the circle black
409     }
410 }
411
412 // Helper Function to Draw all the Mole Holes
413 void wamDisplay_drawAllMoleHoles(){
414     if (numMoleHoles == 4){                                                  //draw all
    the holes for a 4 board
415         wamDisplay_drawMoleHole(MOLE_HOLE_0_0, TRUE);                        //draw
    hole 1
416         wamDisplay_drawMoleHole(MOLE_HOLE_2_0, TRUE);                        //draw
    hole 2
417         wamDisplay_drawMoleHole(MOLE_HOLE_0_2, TRUE);                        //draw
    hole 3
418         wamDisplay_drawMoleHole(MOLE_HOLE_2_2, TRUE);                        //draw
    hole 4
419     }
420     else if(numMoleHoles == 6){                                               //draw
    all the holes for a 6 board
421         wamDisplay_drawMoleHole(MOLE_HOLE_0_0, TRUE);                        //draw
    hole 1
422         wamDisplay_drawMoleHole(MOLE_HOLE_1_0, TRUE);                        //draw
    hole 2
423         wamDisplay_drawMoleHole(MOLE_HOLE_2_0, TRUE);                        //draw
    hole 3
424         wamDisplay_drawMoleHole(MOLE_HOLE_0_2, TRUE);                        //draw
    hole 4
425         wamDisplay_drawMoleHole(MOLE_HOLE_1_2, TRUE);                        //draw
    hole 5
426         wamDisplay_drawMoleHole(MOLE_HOLE_2_2, TRUE);                        //draw
```

```
   hole 6
427     }
428     else if(numMoleHoles == 9){                                          //draw
   all the holes for a 9 board
429         wamDisplay_drawMoleHole(MOLE_HOLE_0_0, TRUE);
430         wamDisplay_drawMoleHole(MOLE_HOLE_1_0, TRUE);                    //draw
   hole 1
431         wamDisplay_drawMoleHole(MOLE_HOLE_2_0, TRUE);                    //draw
   hole 2
432         wamDisplay_drawMoleHole(MOLE_HOLE_0_1, TRUE);                    //draw
   hole 3
433         wamDisplay_drawMoleHole(MOLE_HOLE_1_1, TRUE);                    //draw
   hole 4
434         wamDisplay_drawMoleHole(MOLE_HOLE_2_1, TRUE);                    //draw
   hole 5
435         wamDisplay_drawMoleHole(MOLE_HOLE_0_2, TRUE);                    //draw
   hole 6
436         wamDisplay_drawMoleHole(MOLE_HOLE_1_2, TRUE);                    //draw
   hole 7
437         wamDisplay_drawMoleHole(MOLE_HOLE_2_2, TRUE);                    //draw
   hole 8
438     }
439 };
440
441 // Helper Function to one Mole Hole
442 void wamDisplay_drawMoleHole(uint8_t moleHoleReion, uint8_t drwHole ){
443     if(drwHole){
444         switch (moleHoleReion){
445         case MOLE_HOLE_0_0:
   //if the mole hole region is 0
446             display_fillCircle(MOLE_HOLE_0_X, MOLE_HOLE_0_Y, MOLE_HOLE_RADIUS,
   DISPLAY_BLACK);              //erase the mole hole in this region
447             break;
448         case MOLE_HOLE_1_0:
   //if the mole hole region is 1
449             display_fillCircle(MOLE_HOLE_1_X, MOLE_HOLE_0_Y, MOLE_HOLE_RADIUS,
   DISPLAY_BLACK);              //erase the mole hole in this region
450             break;
451         case MOLE_HOLE_2_0:
   //if the mole hole region is 2
452             display_fillCircle(MOLE_HOLE_2_X, MOLE_HOLE_0_Y, MOLE_HOLE_RADIUS,
   DISPLAY_BLACK);              //erase the mole hole in this region
453             break;
454         case MOLE_HOLE_0_1:
   //if the mole hole region is 3
455             display_fillCircle(MOLE_HOLE_0_X, MOLE_HOLE_1_Y, MOLE_HOLE_RADIUS,
   DISPLAY_BLACK);              //erase the mole hole in this region
456             break;
457         case MOLE_HOLE_1_1:
   //if the mole hole region is 4
458             display_fillCircle(MOLE_HOLE_1_X, MOLE_HOLE_1_Y, MOLE_HOLE_RADIUS,
   DISPLAY_BLACK);              //erase the mole hole in this region
459             break;
460         case MOLE_HOLE_2_1:
   //if the mole hole region is 5
461             display_fillCircle(MOLE_HOLE_2_X, MOLE_HOLE_1_Y, MOLE_HOLE_RADIUS,
   DISPLAY_BLACK);              //erase the mole hole in this region
462             break;
```

```
463         case MOLE_HOLE_0_2:
    //if the mole hole region is 6
464             display_fillCircle(MOLE_HOLE_0_X, MOLE_HOLE_2_Y, MOLE_HOLE_RADIUS,
    DISPLAY_BLACK);             //erase the mole hole in this region
465             break;
466         case MOLE_HOLE_1_2:
    //if the mole hole region is 7
467             display_fillCircle(MOLE_HOLE_1_X, MOLE_HOLE_2_Y, MOLE_HOLE_RADIUS,
    DISPLAY_BLACK);             //erase the mole hole in this region
468             break;
469         case MOLE_HOLE_2_2:
    //if the mole hole region is 8
470             display_fillCircle(MOLE_HOLE_2_X, MOLE_HOLE_2_Y, MOLE_HOLE_RADIUS,
    DISPLAY_BLACK);             //erase the mole hole in this region
471             break;
472         }
473     }
474     else{
    //I dont think I ever use this function
475         switch (moleHoleReion){
476         case MOLE_HOLE_0_0:
    //if the mole region is 0
477             display_fillCircle(MOLE_HOLE_0_X, MOLE_HOLE_0_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
478             break;
479         case MOLE_HOLE_1_0:
    //if the mole region is 1
480             display_fillCircle(MOLE_HOLE_1_X, MOLE_HOLE_0_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
481             break;
482         case MOLE_HOLE_2_0:
    //if the mole region is 2
483             display_fillCircle(MOLE_HOLE_2_X, MOLE_HOLE_0_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
484             break;
485         case MOLE_HOLE_0_1:
    //if the mole region is 3
486             display_fillCircle(MOLE_HOLE_0_X, MOLE_HOLE_1_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
487             break;
488         case MOLE_HOLE_1_1:
    //if the mole region is 4
489             display_fillCircle(MOLE_HOLE_1_X, MOLE_HOLE_1_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
490             break;
491         case MOLE_HOLE_2_1:
    //if the mole region is 5
492             display_fillCircle(MOLE_HOLE_2_X, MOLE_HOLE_1_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
493             break;
494         case MOLE_HOLE_0_2:
    //if the mole region is 6
495             display_fillCircle(MOLE_HOLE_0_X, MOLE_HOLE_2_Y, MOLE_RADIUS,
    DISPLAY_RED);                   //draw the mole in this region
496             break;
497         case MOLE_HOLE_1_2:
    //if the mole region is 7
498             display_fillCircle(MOLE_HOLE_1_X, MOLE_HOLE_2_Y, MOLE_RADIUS,
```

```
   DISPLAY_RED);                    //draw the mole in this region
499             break;
500         case MOLE_HOLE_2_2:
   //if the mole region is 8
501             display_fillCircle(MOLE_HOLE_2_X, MOLE_HOLE_2_Y, MOLE_RADIUS,
   DISPLAY_RED);                    //draw the mole in this region
502             break;
503         }
504     }
505 };
506 void wamDisplay_getMoleCoor(uint16_t index){
507     if( numMoleHoles == BOARD_NUM_HOLES_4){                          //gets the mole
   coordinates for the 4 board
508         switch (index){
509         case MOLE_HOLE_0_0:                                         //if the mole
   region is 0
510             moleCoord.x = MOLE_HOLE_0_X;                            //get the X
   coordinate for this mole
511             moleCoord.y = MOLE_HOLE_0_Y;                            //get the Y
   coordinate for this mole
512             break;
513         case MOLE_HOLE_1_0:                                         //if the mole
   region is 1
514              moleCoord.x = MOLE_HOLE_2_X;                           //get the X
   coordinate for this mole
515              moleCoord.y = MOLE_HOLE_0_Y;                           //get the Y
   coordinate for this mole
516              break;
517         case MOLE_HOLE_2_0:                                         //if the mole
   region is 2
518              moleCoord.x = MOLE_HOLE_0_X;                           //get the X
   coordinate for this mole
519              moleCoord.y = MOLE_HOLE_2_Y;                           //get the Y
   coordinate for this mole
520              break;
521         case MOLE_HOLE_0_1:                                         //if the mole
   region is 3
522              moleCoord.x = MOLE_HOLE_2_X;                           //get the X
   coordinate for this mole
523              moleCoord.y = MOLE_HOLE_2_Y;                           //get the Y
   coordinate for this mole
524              break;
525         }
526     }
527     else if( numMoleHoles == BOARD_NUM_HOLES_6){                        //gets the
   mole coordinates for the 6 board
528         switch (index){
529             case MOLE_HOLE_0_0:                                     //if the mole
   region is 0
530                 moleCoord.x = MOLE_HOLE_0_X;                        //get the X
   coordinate for this mole
531                 moleCoord.y = MOLE_HOLE_0_Y;                        //get the Y
   coordinate for this mole
532                 break;
533             case MOLE_HOLE_1_0:                                     //if the mole
   region is 1
534                 moleCoord.x = MOLE_HOLE_1_X;                        //get the X
   coordinate for this mole
```

```
535                 moleCoord.y = MOLE_HOLE_0_Y;                    //get the Y
    coordinate for this mole
536                 break;
537             case MOLE_HOLE_2_0:                                 //if the mole
    region is 2
538                 moleCoord.x = MOLE_HOLE_2_X;                    //get the X
    coordinate for this mole
539                 moleCoord.y = MOLE_HOLE_0_Y;                    //get the Y
    coordinate for this mole
540                 break;
541             case MOLE_HOLE_0_1:                                 //if the mole
    region is 3
542                 moleCoord.x = MOLE_HOLE_0_X;                    //get the X
    coordinate for this mole
543                 moleCoord.y = MOLE_HOLE_2_Y;                    //get the Y
    coordinate for this mole
544                 break;
545             case MOLE_HOLE_1_1:                                 //if the mole
    region is 4
546                 moleCoord.x = MOLE_HOLE_1_X;                    //get the X
    coordinate for this mole
547                 moleCoord.y = MOLE_HOLE_2_Y;                    //get the Y
    coordinate for this mole
548                 break;
549             case MOLE_HOLE_2_1:                                 //if the mole
    region is 5
550                 moleCoord.x = MOLE_HOLE_2_X;                    //get the X
    coordinate for this mole
551                 moleCoord.y = MOLE_HOLE_2_Y;                    //get the Y
    coordinate for this mole
552                 break;
553         }
554     }
555     else if( numMoleHoles == BOARD_NUM_HOLES_9){                  //gets the
    mole coordinates for the 9 board
556         switch (index){
557             case MOLE_HOLE_0_0:                                 //if the mole
    region is 0
558                 moleCoord.x = MOLE_HOLE_0_X;                    //get the X
    coordinate for this mole
559                 moleCoord.y = MOLE_HOLE_0_Y;                    //get the Y
    coordinate for this mole
560                 break;
561             case MOLE_HOLE_1_0:                                 //if the mole
    region is 1
562                 moleCoord.x = MOLE_HOLE_1_X;                    //get the X
    coordinate for this mole
563                 moleCoord.y = MOLE_HOLE_0_Y;                    //get the Y
    coordinate for this mole
564                 break;
565             case MOLE_HOLE_2_0:                                 //if the mole
    region is 2
566                 moleCoord.x = MOLE_HOLE_2_X;                    //get the X
    coordinate for this mole
567                 moleCoord.y = MOLE_HOLE_0_Y;                    //get the Y
    coordinate for this mole
568                 break;
569             case MOLE_HOLE_0_1:                                 //if the mole
```

```
   region is 3
570                moleCoord.x = MOLE_HOLE_0_X;                    //get the X
   coordinate for this mole
571                moleCoord.y = MOLE_HOLE_1_Y;                    //get the Y
   coordinate for this mole
572                break;
573            case MOLE_HOLE_1_1:                                 //if the mole
   region is 4
574                moleCoord.x = MOLE_HOLE_1_X;                    //get the X
   coordinate for this mole
575                moleCoord.y = MOLE_HOLE_1_Y;                    //get the Y
   coordinate for this mole
576                break;
577            case MOLE_HOLE_2_1:                                 //if the mole
   region is 5
578                moleCoord.x = MOLE_HOLE_2_X;                    //get the X
   coordinate for this mole
579                moleCoord.y = MOLE_HOLE_1_Y;                    //get the Y
   coordinate for this mole
580                break;
581            case MOLE_HOLE_0_2:                                 //if the mole
   region is 6
582                moleCoord.x = MOLE_HOLE_0_X;                    //get the X
   coordinate for this mole
583                moleCoord.y = MOLE_HOLE_2_Y;                    //get the Y
   coordinate for this mole
584                break;
585            case MOLE_HOLE_1_2:                                 //if the mole
   region is 7
586                moleCoord.x = MOLE_HOLE_1_X;                    //get the X
   coordinate for this mole
587                moleCoord.y = MOLE_HOLE_2_Y;                    //get the Y
   coordinate for this mole
588                break;
589            case MOLE_HOLE_2_2:                                 //if the mole
   region is 8
590                moleCoord.x = MOLE_HOLE_2_X;                    //get the X
   coordinate for this mole
591                moleCoord.y = MOLE_HOLE_2_Y;                    //get the Y
   coordinate for this mole
592                break;
593        }
594    }
595}
596
597
```