```c
 * verifySequence_runTest.c

#include "verifySequence_runTest.h"
#include "buttonHandler.h"
#include "simonDisplay.h"
#include "supportFiles/display.h"
#include "supportFiles/utils.h"
#include "globals.h"
#include <stdio.h>
#include <stdint.h>
#include "../Lab2_switch_button/buttons.h"

//**********CODE_FROM_PROF**********//
#define MESSAGE_X 0
//#define MESSAGE_Y (display_width()/4)
#define MESSAGE_Y (display_height()/2)
#define MESSAGE_TEXT_SIZE 2
//#define MESSAGE_STARTING_OVER
#define BUTTON_0 0  // Index for button 0
#define BUTTON_1 1  // Index for button 1
#define BUTTON_2 2  // Index for button 2
#define BUTTON_3 3  // Index for button 3

//**********MY_CODE**********//

#define ENABLE_FLAG_ON 1      //Var for when the FLAG is o
#define ENABLE_FLAG_OFF 0     // Var for when the enable flag is off
#define TIME_OUT_NUM 20
#define TRUE 1                        // sets true to 1
#define FALSE 0                       // sets false to 0
uint8_t verifyEnableFlag = 0;          //  declares the enable flag
uint8_t verifyIsCompleteFlag = 0;   // flag that shows if the SM is complete
uint8_t timeOutErrorFlag = 0;       // flag that show if the game has timed out
uint8_t userInputErrorFlag = 0;     // flag for when the user makes an error
uint8_t timeOut = 0;                // time out VAR
uint8_t indexInArray = 0;           // what index in the array are they



enum verifySequence_st_m{
    init_vs_st,         //state number 1
    enable_vs_st,         //state number 2
    wait_release_vs_st,         //state number 3
    region_vs_st,         //state number 4
    incrament_array_vs_st,  // state number 5
    finish_vs_st         //state number 6
} verifySequenceCurrentState = init_vs_st;

// State machine will run when enabled.
void verifySequence_enable(){
    verifyEnableFlag = ENABLE_FLAG_ON;     //sets the flag to on
}

// This is part of the interlock. You disable the state-machine and then enable it
  again.
void verifySequence_disable(){
    verifyEnableFlag = ENABLE_FLAG_OFF;     //sets the flag to off
```

```
63 }
64
65 // Used to detect if there has been a time-out error.
66 bool verifySequence_isTimeOutError(){
67     return timeOutErrorFlag;
68 }
69
70 // Used to detect if the user tapped the incorrect sequence.
71 bool verifySequence_isUserInputError(){
72     return userInputErrorFlag;
73 }
74
75 // Used to detect if the verifySequence state machine has finished verifying.
76 bool verifySequence_isComplete(){
77     return verifyIsCompleteFlag;
78 }
79
80 // Standard tick function.
81 void verifySequence_tick(){
82     switch(verifySequenceCurrentState){
83     case init_vs_st:         //state number 1 for moore
84         break;
85     case enable_vs_st:         //state number 2 for moore
86         buttonHandler_enable();         // enable the button handler
87         break;
88     case wait_release_vs_st:         //state number 3 for moore
89         timeOut++;                  // increment the timne out counter
90         break;
91     case region_vs_st:         //state number 4 for moore
92         buttonHandler_disable();            // disable the button handler
93         if(buttonHandler_getRegionNumber() != globals_getSequenceValue(indexInArray)
   ){ // if the area touch doesnt match the value the array has then user made an error
94             userInputErrorFlag = TRUE;
   // raise flag
95         }
96         break;
97     case incrament_array_vs_st:  // state number 5 for moore
98         break;
99     case finish_vs_st:         //state number 6 for moore
100        break;
101    }
102
103    switch(verifySequenceCurrentState){
104    case init_vs_st:         //state number 1 for mealy
105        if(verifyEnableFlag){
106            timeOut = FALSE;                                    // reset the VAR
107            indexInArray = FALSE;                                    // reset the VAR
108            verifyIsCompleteFlag = FALSE;                                    // reset the
   VAR
109            timeOutErrorFlag = FALSE;                                  // reset the VAR
110            userInputErrorFlag = FALSE;                                  // reset the
   VAR
111            verifySequenceCurrentState = enable_vs_st;
112        }
113        break;
114    case enable_vs_st:         //state number 2 for mealy
115        verifySequenceCurrentState = wait_release_vs_st;
116        break;
```

```
117     case wait_release_vs_st:         //state number 3 for mealy
118         if(timeOut == TIME_OUT_NUM){
119             verifyIsCompleteFlag = TRUE;                                    // raise the
    flag that the Verify squence is done
120             timeOutErrorFlag = TRUE;                                        // raise the
    flag that is took to long to press
121             simonDisplay_eraseAllButtons();                                 // erase all
    the buttons from the srceen
122             verifySequenceCurrentState = finish_vs_st;
123         }
124         else if(display_isTouched()){
125             timeOut = FALSE;                                                // if display
    is touch reset the timeout VAR
126         }
127         else if(buttonHandler_releaseDetected()){                           // if the
    screen was released then move to next state
128             verifySequenceCurrentState = region_vs_st;
129         }
130         break;
131     case region_vs_st:                                                       //state
    number 4 for mealy
132         verifySequenceCurrentState = increment_array_vs_st;                 //move to
    next state
133         break;
134     case increment_array_vs_st:                                         // state number 5
    for mealy
135         if (indexInArray == globals_getSequenceIterationLength()){
136             verifyIsCompleteFlag = TRUE;                                // set the
    complete flag to true
137             simonDisplay_eraseAllButtons();                             // erase all the
    buttons
138             verifySequenceCurrentState = finish_vs_st;
139         }
140         else if(userInputErrorFlag){
141             verifyIsCompleteFlag = TRUE;                                // set the
    complete flag to true
142             simonDisplay_eraseAllButtons();                             // erase all the
    buttons
143             verifySequenceCurrentState = finish_vs_st;
144         }
145         else {
146             indexInArray++;                                             // increment the
    nember in the array that the program is checking
147             verifySequenceCurrentState = enable_vs_st;
148         }
149         break;
150     case finish_vs_st:         //state number 6 for mealy
151         if(!verifyEnableFlag){                                          // wait til the
    enable flag is lowered to exit the state machine
152            verifySequenceCurrentState = init_vs_st;
153         }
154         break;
155     }
156
157 }
158
159 // Prints the instructions that the user should follow when
160 // testing the verifySequence state machine.
```

```
161 // Takes an argument that specifies the length of the sequence so that
162 // the instructions are tailored for the length of the sequence.
163 // This assumes a simple incrementing pattern so that it is simple to
164 // instruct the user.
165 void verifySequence_printInstructions(uint8_t length, bool startingOver) {
166     display_fillScreen(DISPLAY_BLACK);              // Clear the screen.
167     display_setTextSize(MESSAGE_TEXT_SIZE);     // Make it readable.
168     display_setCursor(MESSAGE_X, MESSAGE_Y);    // Rough center.
169     if (startingOver) {                                             // Print a message
    if you start over.
170         display_fillScreen(DISPLAY_BLACK);          // Clear the screen if starting
    over.
171         display_setTextColor(DISPLAY_WHITE);        // Print whit text.
172         display_println("Starting Over. ");         // Starting over message.
173     }
174     // Print messages are self-explanatory, no comments needed.
175     // These messages request that the user touch the buttons in a specific sequence.
176     display_println("Tap: ");
177     display_println();
178     switch (length) {
179     case 1:
180         display_println("red");
181         break;
182     case 2:
183         display_println("red, yellow ");
184         break;
185     case 3:
186         display_println("red, yellow, blue ");
187         break;
188     case 4:
189         display_println("red, yellow, blue, green ");
190         break;
191     default:
192         break;
193     }
194     display_println("in that order.");
195     display_println();
196     display_println("hold BTN0 to quit.");
197 }
198
199 // Just clears the screen and draws the four buttons used in Simon.
200 void verifySequence_drawButtons() {
201     display_fillScreen(DISPLAY_BLACK);  // Clear the screen.
202     simonDisplay_drawButton(BUTTON_0);  // Draw the four buttons.
203     simonDisplay_drawButton(BUTTON_1);
204     simonDisplay_drawButton(BUTTON_2);
205     simonDisplay_drawButton(BUTTON_3);
206 }
207
208 // This will set the sequence to a simple sequential pattern.
209 #define MAX_TEST_SEQUENCE_LENGTH 4  // the maximum length of the pattern
210 uint8_t verifySequence_testSequence[MAX_TEST_SEQUENCE_LENGTH] = {0, 1, 2, 3};  // A
    simple pattern.
211 #define MESSAGE_WAIT_MS 4000  // Display messages for this long.
212
213 // Increment the sequence length making sure to skip over 0.
214 // Used to change the sequence length during the test.
215 int16_t incrementSequenceLength(int16_t sequenceLength) {
```

```
216     int16_t value = (sequenceLength + 1) % (MAX_TEST_SEQUENCE_LENGTH+1);
217     if (value == 0) value++;
218     return value;
219 }
220 
221 // Used to select from a variety of informational messages.
222 enum verifySequence_infoMessage_t {
223     user_time_out_e,            // means that the user waited too long to tap a color.
224     user_wrong_sequence_e,      // means that the user tapped the wrong color.
225     user_correct_sequence_e,    // means that the user tapped the correct sequence.
226     user_quit_e                 // means that the user wants to quite.
227 };
228 
229 // Prints out informational messages based upon a message type (see above).
230 void verifySequence_printInfoMessage(verifySequence_infoMessage_t messageType) {
231     // Setup text color, position and clear the screen.
232     display_setTextColor(DISPLAY_WHITE);
233     display_setCursor(MESSAGE_X, MESSAGE_Y);
234     display_fillScreen(DISPLAY_BLACK);
235   switch(messageType) {
236   case user_time_out_e:  // Tell the user that they typed too slowly.
237         display_println("Error:");
238         display_println();
239         display_println("  User tapped sequence");
240         display_println("  too slowly.");
241     break;
242   case user_wrong_sequence_e:  // Tell the user that they tapped the wrong color.
243         display_println("Error: ");
244         display_println();
245         display_println("  User tapped the");
246         display_println("  wrong sequence.");
247     break;
248   case user_correct_sequence_e:  // Tell the user that they were correct.
249         display_println("User tapped");
250         display_println("the correct sequence.");
251     break;
252   case user_quit_e:             // Acknowledge that you are quitting the test.
253     display_println("quitting runTest().");
254     break;
255   default:
256     break;
257   }
258 }
259 
260 #define TICK_PERIOD_IN_MS 100
261 // Tests the verifySequence state machine.
262 // It prints instructions to the touch-screen. The user responds by tapping the
263 // correct colors to match the sequence.
264 // Users can test the error conditions by waiting too long to tap a color or
265 // by tapping an incorrect color.
266 void verifySequence_runTest() {
267     display_init();  // Always must do this.
268     buttons_init();  // Need to use the push-button package so user can quit.
269     int16_t sequenceLength = 1;  // Start out with a sequence length of 1.
270     verifySequence_printInstructions(sequenceLength, false);  // Tell the user what to
    do.
271     utils_msDelay(MESSAGE_WAIT_MS);  // Give them a few seconds to read the
    instructions.
```

```
272     verifySequence_drawButtons();    // Now, draw the buttons.
273     // Set the test sequence and it's length.
274     globals_setSequence(verifySequence_testSequence, MAX_TEST_SEQUENCE_LENGTH);
275     globals_setSequenceIterationLength(sequenceLength);
276     // Enable the verifySequence state machine.
277     verifySequence_enable();  // Everything is interlocked, so first enable the
    machine.
278     // Need to hold button until it quits as you might be stuck in a delay.
279     while (!(buttons_read() & BUTTONS_BTN0_MASK)) {
280         // verifySequence uses the buttonHandler state machine so you need to "tick"
    both of them.
281         verifySequence_tick();  // Advance the verifySequence state machine.
282         buttonHandler_tick();   // Advance the buttonHandler state machine.
283         utils_msDelay(TICK_PERIOD_IN_MS);       // Wait for a tick period.
284         // If the verifySequence state machine has finished, check the result,
285         // otherwise just keep ticking both machines.
286         if (verifySequence_isComplete()) {
287             if (verifySequence_isTimeOutError()) {                // Was the user too
    slow?
288                 verifySequence_printInfoMessage(user_time_out_e); // Yes, tell the
    user that they were too slow.
289             } else if (verifySequence_isUserInputError()) {       // Did the user tap
    the wrong color?
290                 verifySequence_printInfoMessage(user_wrong_sequence_e); // Yes, tell
    them so.
291             } else {
292                 verifySequence_printInfoMessage(user_correct_sequence_e); // User was
    correct if you get here.
293             }
294             utils_msDelay(MESSAGE_WAIT_MS);                           // Allow the
    user to read the message.
295             sequenceLength = incrementSequenceLength(sequenceLength);  // Increment
    the sequence.
296             globals_setSequenceIterationLength(sequenceLength);        // Set the
    length for the verifySequence state machine.
297             verifySequence_printInstructions(sequenceLength, true);    // Print the
    instructions.
298             utils_msDelay(MESSAGE_WAIT_MS);                           // Let the user
    read the instructions.
299             verifySequence_drawButtons();                             // Draw the
    buttons.
300             verifySequence_disable();                                 // Interlock:
    first step of handshake.
301             verifySequence_tick();                                    // Advance the
    verifySequence machine.
302             utils_msDelay(TICK_PERIOD_IN_MS);                         // Wait for
    tick period.
303             verifySequence_enable();                                  // Interlock:
    second step of handshake.
304             utils_msDelay(TICK_PERIOD_IN_MS);                         // Wait for
    tick period.
305         }
306     }
307     verifySequence_printInfoMessage(user_quit_e);  // Quitting, print out an
    informational message.
308 }
309
310
```