

GAME STATE MACHINE

```

/*****

Header file for the Game State Machine

based on the Gen2 Events and Services Framework

*****/

#ifndef GameState_H
#define GameState_H

// Event Definitions
#include "ES_Configure.h" /* gets us event definitions */
#include "ES_Types.h" /* gets bool type for returns */

// States in FSM
typedef enum
{
    InitPState, WelcomeScreen, GALeader, GAFollower, GARoundComplete,
    GameComplete, Demo
}GameState_t;

// Public Function Prototypes
bool InitGameState(uint8_t Priority);
bool PostGameState(ES_Event_t ThisEvent);
ES_Event_t RunGameState(ES_Event_t ThisEvent);

// Event Checkers
bool CheckTouchSensor();

// Query Functions
void queryHighScores(uint16_t* score1, uint16_t* score2, uint16_t* score3);

#endif /* GameState_H */

/*****
Module
    GameState.c

Revision
    1.0.0

Description
    GameState is an FSM that manages the current game state of the game.

Notes

History
When      Who      What/Why
-----
11/03/20   kcao   Implementation of Demo
10/31/20   kcao   Integration with Dotstar Service
10/30/20   kcao   Integration with Display Service
10/29/20   kcao   Integration with Sequence State Machine
10/28/20   kcao   File creation
*****/
/*----- Include Files -----*/
/* include header files for this state machine as well as any machines at
the next lower level in the hierarchy that are sub-machines to this machine

```

```

*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "GameState.h"
#include "hal.h"
#include "Seq.h"
#include "Display.h"
#include "Dotstar.h"
#include "MasterReset.h"

/*----- Module Defines -----*/

#define SENSOR_INPUT_PIN 10 // corresponds to real pin 11 for touch sensor

/*----- Module Functions -----*/
/* prototypes for private functions for this machine.They should be
functions
relevant to the behavior of this state machine
*/
static bool UpdateHighScores(const uint16_t score);
static int compareScores(const void *a, const void *b);
static void masterReset();

/*----- Module Variables -----*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match that of enum in header file
static GameState_t CurrentState;
static uint16_t highScores[4];
static uint16_t roundNumber;
static uint8_t lastTouchSensorState;

// with the introduction of Gen2, we need a module level Priority var as
well
static uint8_t MyPriority;

/*----- Module Code -----*/
/*****
Function
    InitGameState

Parameters
    uint8_t : the priority of this service

Returns
    bool, false if error in initialization, true otherwise

Description
    Saves away the priority, sets up the initial transition and does any
    other required initialization for this state machine

Notes

Author
    K Cao, 10/28/20
*****/
bool InitGameState(uint8_t Priority)
{
    ES_Event_t InitEvent;
    MyPriority = Priority;
    CurrentState = InitPState;
    InitEvent.EventType = ES_INIT;

```

```

    if (ES_PostToService(MyPriority, InitEvent) == true){
        return true;
    } else {
        return false;
    }
}

/*****
Function
    PostGameState

Parameters
    EF_Event_t ThisEvent, the event to post to the queue

Returns
    boolean False if the Enqueue operation failed, True otherwise

Description
    Posts an event to this state machine's queue

Notes

Author
    K Cao, 10/28/20
*****/
bool PostGameState(ES_Event_t ThisEvent)
{
    return ES_PostToService(MyPriority, ThisEvent);
}

/*****
Function
    RunGameState

Parameters
    ES_Event_t : the event to process

Returns
    ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
    State machine that manages what the current game state is and
communicates
    directly with the display and dotstar. The FSM also manages the scores.

Notes
    uses nested switch/case to implement the machine.

Author
    K Cao, 10/28/20
*****/
ES_Event_t RunGameState(ES_Event_t ThisEvent)
{
    ES_Event_t ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT;

    switch (CurrentState)
    {
        case InitPState:
        {
            if (ThisEvent.EventType == ES_INIT)
            {
                // Init high scores

```

```

for (uint8_t i = 0; i < 4; i++)
{
    highScores[i] = 0;
}

// Update display to Welcome Screen
ES_Event_t DisplayEvent;
DisplayEvent.EventType = ES_DISPLAY_WELCOME;
PostDisplay(DisplayEvent);

// Update dotstar to be on random colors
ES_Event_t DotstarEvent;
DotstarEvent.EventType = ES_RANDOM;
PostDotstar(DotstarEvent);

// Init touch sensor state and demo timer
lastTouchSensorState = digitalRead(SENSOR_INPUT_PIN);
ES_Timer_InitTimer(DEMO_TIMER, 15000);
CurrentState = WelcomeScreen;
}
}
break;

case WelcomeScreen:
{
    switch (ThisEvent.EventType)
    {
        case ES_SENSOR_PRESSED:
        {
            // Update display with ready screen and round number
            roundNumber = 1;
            ES_Event_t DisplayEvent;
            DisplayEvent.EventType = ES_DISPLAY_READY;
            DisplayEvent.EventParam = roundNumber;
            PostDisplay(DisplayEvent);

            // Update dotstar to be off
            ES_Event_t DotstarEvent;
            DotstarEvent.EventType = ES_OFF;
            PostDotstar(DotstarEvent);

            // Update sequence state machine for first round
            ES_Event_t SequenceEvent;
            SequenceEvent.EventType = ES_FIRST_ROUND;
            PostSequence(SequenceEvent);

            // Init ready timer
            ES_Timer_InitTimer(READY_TIMER, 1000);
            CurrentState = GALeader;
        }
        break;

        case ES_TIMEOUT:
        {
            if (ThisEvent.EventParam == DEMO_TIMER)
            {
                // Update display with Demo Screen
                ES_Event_t DisplayEvent;
                DisplayEvent.EventType = ES_DISPLAY_DEMO;
                PostDisplay(DisplayEvent);
            }
        }
    }
}

```



```

        // Update dotstar to flash green
        ES_Event_t DotstarEvent;
        DotstarEvent.EventType = ES_GREEN;
        PostDotstar(DotstarEvent);

        CurrentState = GARoundComplete;
    }
    break;

    case ES_GAME_COMPLETE:
    {
        // Update display to Game Complete Screen
        ES_Event_t DisplayEvent;
        DisplayEvent.EventType = ES_DISPLAY_GAMECOMPLETE;
        PostDisplay(DisplayEvent);

        // Update dotstar to flash green/red based on whether high score
        achieved
        uint16_t score = ThisEvent.EventParam;
        ES_Event_t DotstarEvent;
        if (UpdateHighScores(score)) {
            DotstarEvent.EventType = ES_GREEN;
        } else {
            DotstarEvent.EventType = ES_RED;
        }
        PostDotstar(DotstarEvent);

        // Init Game Over Timer
        ES_Timer_InitTimer(GAMEOVER_TIMER, 30000);
        CurrentState = GameComplete;
    }
    break;

    case ES_MASTER_RESET:
    {
        masterReset();
    }
    break;

    default:
        ;
}
break;

case GARoundComplete:
{
    switch (ThisEvent.EventType)
    {
        case ES_SENSOR_PRESSED:
        {
            // Update display to Ready Screen with round number
            roundNumber++;
            ES_Event_t DisplayEvent;
            DisplayEvent.EventType = ES_DISPLAY_READY;
            DisplayEvent.EventParam = roundNumber;
            PostDisplay(DisplayEvent);

            // Update dotstar to turn off
            ES_Event_t DotstarEvent;
            DotstarEvent.EventType = ES_OFF;

```

```

        PostDotstar(DotstarEvent);

        // Init Ready Timer
        ES_Timer_InitTimer(READY_TIMER, 1000);
        CurrentState = GALeader;
    }
    break;

    case ES_MASTER_RESET:
    {
        masterReset();
    }
    break;

    default:
        ;
    }
}
break;

case GameComplete:
{
    switch (ThisEvent.EventType)
    {
        case ES_SENSOR_PRESSED:
        {
            // Update display to Welcome Screen
            ES_Event_t DisplayEvent;
            DisplayEvent.EventType = ES_DISPLAY_WELCOME;
            PostDisplay(DisplayEvent);

            // Update dotstar with random colors
            ES_Event_t DotstarEvent;
            DotstarEvent.EventType = ES_RANDOM;
            PostDotstar(DotstarEvent);

            // Init Demo Timer
            ES_Timer_InitTimer(DEMO_TIMER, 15000);
            CurrentState = WelcomeScreen;
        }
        break;

        case ES_TIMEOUT:
        {
            if (ThisEvent.EventParam == GAMEOVER_TIMER) {
                // Update display to Welcome Screen
                ES_Event_t DisplayEvent;
                DisplayEvent.EventType = ES_DISPLAY_WELCOME;
                PostDisplay(DisplayEvent);

                // Update dotstar with random colors
                ES_Event_t DotstarEvent;
                DotstarEvent.EventType = ES_RANDOM;
                PostDotstar(DotstarEvent);

                // Init Demo Timer
                ES_Timer_InitTimer(DEMO_TIMER, 15000);
                CurrentState = WelcomeScreen;
            }
        }
        break;
    }
}

```

```

        case ES_MASTER_RESET:
        {
            masterReset();
        }
        break;

        default:
            ;
    }
}
break;

case Demo:
{
    switch (ThisEvent.EventType)
    {
        case ES_TIMEOUT:
        {
            if (ThisEvent.EventParam == LAST_DIRECTION_TIMER) {
                // Complete a master reset of both the game state and sequence

                masterReset();
                ES_Event_t SequenceEvent;
                SequenceEvent.EventType = ES_MASTER_RESET;
                PostSequence(SequenceEvent);
            }
        }
        break;

        default:
            ;
    }
}
break;

default:
    ;
}
return ReturnEvent;
}

```

FSMs

```

/*****

```

Function
queryHighScores

Parameters
Three uint16_ts passed by reference

Returns
Nothing

Description
Query function for the display service to update scores. Three unsigned 16-bit integers must be passed by reference.

Author
K Cao, 10/28/20

```

*****/
void queryHighScores(uint16_t* score1, uint16_t* score2, uint16_t* score3){
    *score1 = highScores[0];
}

```



```

    *score2 = highScores[1];
    *score3 = highScores[2];
}

/*****
event checkers
*****/

/*****
Function
    CheckTouchSensor

Parameters
    Nothing

Returns
    bool, true if touch sensor changes from pressed to unpressed

Description
    Event checker to check if touch sensor pressed.

Notes
    Only checks in specific states - WelcomeScreen, GARoundComplete,
    GameComplete

Author
    K Cao, 10/28/20
*****/
bool CheckTouchSensor() {
    bool eventStatus = false;
    if ((CurrentState == WelcomeScreen) || (CurrentState == GARoundComplete)
||
        (CurrentState == GameComplete))
    {
        uint8_t currentTouchSensorState = digitalRead(SENSOR_INPUT_PIN);
        if ((currentTouchSensorState != lastTouchSensorState) &&
            (currentTouchSensorState == LOW)) {
            // Update game state of touch sensor press
            ES_Event_t TouchSensorEvent;
            TouchSensorEvent.EventType = ES_SENSOR_PRESSED;
            PostGameState(TouchSensorEvent);

            // Update master reset state machine of a detected input
            ES_Event_t InputEvent;
            InputEvent.EventType = ES_INPUT_DETECTED;
            PostMasterReset(InputEvent);

            eventStatus = true;
        }
        lastTouchSensorState = currentTouchSensorState;
    }
    return eventStatus;
}

/*****
Function
    UpdateHighScores

Parameters
    uint16_t score, achieved by the player after game is complete

```

Returns

bool, true if latest score in top 3, false if not

Description

Helper function that maintains and updates the high scores.

Notes

Newest score written to fourth array component before being sorted with QuickSort.

Author

K Cao, 10/28/20

```
*****/  
static bool UpdateHighScores(const uint16_t score){  
    // Sort high scores with QuickSort  
    highScores[3] = score;  
    qsort(highScores, 4, 2, compareScores);  
    // Check if in top 3 scores  
    bool highScoreFlag = false;  
    for (uint8_t i = 0; i < 3; i++){  
        if (highScores[i] == score){  
            highScoreFlag = true;  
            break;  
        }  
    }  
    return highScoreFlag;  
}
```

```
/******
```

Function

compareScores

Parameters

Two generic (void) pointers

Returns

Integer value - positive if b is greater than a, negative if a is greater than b, and zero if a is equal to b.

Description

Comparison function for two unsigned 16-bit integers that returns an integer value based on C's strcmp standards.

Notes

Generic implementation required for C's qsort.

Author

K Cao, 10/29/20

```
*****/  
static int compareScores(const void *a, const void *b){  
    return *(const uint16_t *)b - *(const uint16_t *)a;  
}
```

```
/******
```

Function

masterReset

Parameters

Nothing

Returns
Nothing

Description
Completes a master reset of the display, dotstar and demo timers.

Notes

Author
K Cao, 10/29/20

```
*****/
static void masterReset(){
    // Update display to Welcome Screen
    ES_Event_t DisplayEvent;
    DisplayEvent.EventType = ES_DISPLAY_WELCOME;
    PostDisplay(DisplayEvent);

    // Update dotstar to random colors
    ES_Event_t DotstarEvent;
    DotstarEvent.EventType = ES_RANDOM;
    PostDotstar(DotstarEvent);

    // Init Demo Timer
    ES_Timer_InitTimer(DEMO_TIMER, 15000);
    CurrentState = WelcomeScreen;
}
```

SEQUENCE STATE MACHINE

```
/*
 * File:   Seq.h
 * Author: chris
 *
 * Created on October 28, 2020, 7:09 AM
 */

#ifndef SEQ_H
#define SEQ_H

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

#include <xc.h>
#include <p32xxxx.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/attrs.h>
#include <sys/kmem.h>

#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_DeferRecall.h"
#include "ES_ShortTimer.h"
#include "ES_Port.h"

//type def
typedef enum{
    PseudoInit=0,
    SequenceCreate,
    SequenceDisplay,
    SequenceInput
}SequenceState_t;

//functions
bool InitSequence(uint8_t Priority);
bool PostSequence(ES_Event_t ThisEvent);
ES_Event_t RunSequence(ES_Event_t ThisEvent);

//Event Checker
bool CheckXYVal (void);

//Complementary functions
bool Input_Check(uint32_t *adcResults);
uint8_t Input_Direction(uint32_t *adcResults);

#endif /* SEQ_H */

/*****
Module
```

Seq.c

Revision
1.0.0

Description

Sequence is a state machine that manages the creation/appending of the sequence and going through the sequence during the demo and gameplay.

Notes

History

When	Who	What/Why
10/30/20	kcao	Integration with Display Service
10/29/20	kcao	Integration with Game State
10/29/20	cbarresi	Creation and implementation

*****/

```
#include "Seq.h"
#include "PIC32_AD_Lib.h"
// Hardware
#include <xc.h>
#include <proc/p32mx170f256b.h>
#include <sys/attribs.h> // for ISR macros
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Event & Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_DeferRecall.h"
#include "ES_ShortTimer.h"
#include "ES_Port.h"

// OLED headers
#include "../u8g2Headers/u8g2TestHarness_main.h"
#include "../u8g2Headers/common.h"
#include "../u8g2Headers/spi_master.h"
#include "../u8g2Headers/u8g2.h"
#include "../u8g2Headers/u8x8.h"

// Game Services
#include "GameState.h"
#include "Display.h"
#include "hal.h"
#include "MasterReset.h"

/*----- Module Defines -----*/

#define ROUND_TIME 15

/*----- Module Functions -----*/
/* prototypes for private functions for this machine.They should be
functions
relevant to the behavior of this state machine
*/
static void updateScore();
static bool inputChecker(uint32_t *adcResults);
```

```

static uint16_t bitPack(const uint8_t score, const uint8_t time, const
uint8_t input);
static void masterReset();
uint8_t Input_Direction(uint32_t *adcResults);

/*----- Module Variables -----*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;

static uint8_t seqArray[150]; //array containing random directions
static uint8_t arrayLength; //counter variable that contains length of
array
static uint8_t score; //initial player score
static uint8_t seqIndex; //Sequence Index
static uint8_t playtimeLeft; //Play time counter
static uint8_t roundNumber; //Round number
static uint8_t displayCounter; // Display Counter

static SequenceState_t CurrentState; //State Machine Current State Variable

static uint32_t adcResults[2]; //Array for Joystick AD converter function
static uint8_t lastTouchSensor; //Last value for event checker
static uint32_t Neutral[2]; //Array containing neutral positions for X, Y
static uint8_t input; //variable to pass user input to OLED

/*----- Module Code -----*/

/*****
Function
    InitSequence
Parameters
    uint8_t : the priority of this service
Returns
    bool, false if error in initialization, true otherwise
Description
    Initialize Joystick pins and program, sets input for touch button
button.
    * Initializes lastTouchSensor for event checking function CheckXYVal
*****/

bool InitSequence(uint8_t Priority)
{
    ES_Event_t InitEvent;
    MyPriority = Priority;
    //Initialize Framework Pins for Joystick
    //Pins RB2-Y, RB3-X are analog inputs, RA2-Z is digital Input,
    //Set Touch Sensor as Digital Input RB4-TS
    ANSELBbits.ANSB2 = 1;
    ANSELBbits.ANSB3 = 1;
    TRISBbits.TRISB2 = 1;
    TRISBbits.TRISB3 = 1;
    TRISAbits.TRISA2 = 1;
    TRISBbits.TRISB4 = 1;

    //Initialization of AD converter
    ADC_ConfigAutoScan((BIT4HI | BIT5HI), 2);

    //Initializing lastTouchSensor for event checker
    lastTouchSensor = PORTBbits.RB4;

    //Set current State

```

```

CurrentState = PseudoInit;
// post the initial transition event
InitEvent.EventType = ES_INIT;
if (ES_PostToService(MyPriority, InitEvent) == true)
{
    return true;
}
else
{
    return false;
}
}

/*****
Function
PostSequence
Parameters
    EF_Event ThisEvent ,the event to post to the queue
Returns
    bool false if the queue operation failed, true otherwise
Description
    Posts an event to this state machine's queue
*****/
bool PostSequence(ES_Event_t ThisEvent)
{
    return ES_PostToService(MyPriority, ThisEvent);
}

/*****
Function
RunSequence
Parameters
    ES_Event : the event to process
Returns
    ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise
Description
    Initializes the game score, round, and array length. Obtains neutral
values
    * for X and Y axis of joystick. Creates a random directions to display to
    * the user and later check against the user input. Post events to Game
    * State Machine and the OLED State Machine. Updates the score and round
values
*****/
ES_Event_t RunSequence(ES_Event_t ThisEvent)
{
    ES_Event_t ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
    switch(CurrentState)
    {
        case PseudoInit:
        {
            if (ThisEvent.EventType == ES_INIT)
            {
                // Read X and Y values from Joystick to obtain neutral positions
                ADC_MultiRead(adcResults);
                Neutral[0] = adcResults[0]; // Y neutral position
                Neutral[1] = adcResults[1]; // X neutral position
                CurrentState = SequenceCreate;
            }
        }
    }
}

```

```

break;

case SequenceCreate:
{
    switch (ThisEvent.EventType)
    {
        case ES_FIRST_ROUND:
        {
            // Initialize array length, round and score
            seqIndex = 0;
            arrayLength = 4;
            roundNumber = 1;
            score = 0;

            // Randomly initialize a sequence
            srand(ES_Timer_GetTime());
            for (uint8_t i = 0; i < arrayLength; i++){
                seqArray[i] = (rand() % 80) / 10;
            }
        }
        break;

        case ES_NEXT_ROUND:
        {
            // Append random direction to sequence
            seqIndex = 0;
            seqArray[arrayLength] = (rand() % 80) / 10;
            arrayLength++;
            roundNumber++;
        }
        break;

        case ES_TIMEOUT:
        {
            if ((ThisEvent.EventParam == READY_TIMER) ||
                (ThisEvent.EventParam == DEMO_SCREEN_TIMER))
            {
                // Inform display service to demonstrate input and
starts first direction timer
                displayCounter = 0;
                ES_Event_t DisplayEvent;
                DisplayEvent.EventType = ES_DISPLAY_INSTRUCTION;
                DisplayEvent.EventParam = seqArray[displayCounter];
                PostDisplay(DisplayEvent);
                displayCounter++;
                ES_Timer_InitTimer(DIRECTION_TIMER, 750);
                CurrentState = SequenceDisplay;
            }
        }
        break;

        case ES_MASTER_RESET:
        {
            masterReset();
        }
        break;

        default: {} break;
    }
}
break;

```



```

case SequenceDisplay:
{
    switch(ThisEvent.EventType)
    {
        case ES_TIMEOUT:
        {
            switch (ThisEvent.EventParam)
            {
                case DIRECTION_PAUSE_TIMER:
                {
                    // Inform display service to demonstrate input
                    and starts subsequent direction timers
                    ES_Event_t DisplayEvent;
                    DisplayEvent.EventType =
                    ES_DISPLAY_INSTRUCTION;
                    DisplayEvent.EventParam =
                    seqArray[displayCounter];
                    PostDisplay(DisplayEvent);

                    // If not last direction
                    if (displayCounter < (arrayLength - 1)){
                        ES_Timer_InitTimer(DIRECTION_TIMER, 750);
                    }

                    // If last direction
                    if (displayCounter == (arrayLength - 1)){
                        ES_Timer_InitTimer(LAST_DIRECTION_TIMER,
                    750);
                    }
                    displayCounter++;
                }
                break;

                case DIRECTION_TIMER:
                {
                    //Post all arrows unhighlighted to create
                    blinking effect
                    ES_Timer_InitTimer(DIRECTION_PAUSE_TIMER, 250);
                    ES_Event_t DisplayEvent;
                    DisplayEvent.EventType =
                    ES_DISPLAY_INSTRUCTION;
                    DisplayEvent.EventParam = 8; //All arrows blank
                    PostDisplay(DisplayEvent);
                }
                break;

                case GO_TIMER:
                {
                    playtimeLeft = ROUND_TIME;
                    // Inform display service to update to play
                    screen and starts input timer
                    ES_Event_t DisplayEvent;
                    input = 8; // all arrows on
                    DisplayEvent.EventType =
                    ES_DISPLAY_PLAY_UPDATE;
                    DisplayEvent.EventParam = bitPack(score,
                    playtimeLeft, input);
                    PostDisplay(DisplayEvent);
                    ES_Timer_InitTimer(INPUT_TIMER, 1000);
                    ES_Timer_InitTimer(INSTRUCTION_TIMER, 101);
                }
            }
        }
    }
}

```

```

        CurrentState = SequenceInput;
    }
    break;

    default: {} break;
}
}
break;

case ES_MASTER_RESET:
{
    masterReset();
}
break;

default: {} break;
}

}
break;

case SequenceInput:
{
    switch (ThisEvent.EventType)
    {
        case ES_TIMEOUT:
        {
            if (ThisEvent.EventParam == INPUT_TIMER)
            {
                if (playtimeLeft > 0)
                {
                    // Inform display service to update time
                    playtimeLeft--;
                    ES_Timer_InitTimer(INPUT_TIMER, 1000);
                }

                else if (playtimeLeft == 0)
                {
                    // Update sequence state machine
                    CurrentState = SequenceCreate;
                    // Inform GameState machine
                    ES_Event_t GameStateEvent;
                    GameStateEvent.EventType = ES_GAME_COMPLETE;
                    GameStateEvent.EventParam = score;
                    PostGameState(GameStateEvent);
                }
            }
            if (ThisEvent.EventParam == INSTRUCTION_TIMER)
            {
                // Read X and Y values from Joystick
                ADC_MultiRead(adcResults);

                // Post to OLED
                ES_Event_t DisplayEvent;
                DisplayEvent.EventType = ES_DISPLAY_PLAY_UPDATE;
                DisplayEvent.EventParam = bitPack(score,
playtimeLeft, Input_Direction(adcResults));
                PostDisplay(DisplayEvent);

                // Restart Timer
                ES_Timer_InitTimer(INSTRUCTION_TIMER, 101);
            }
        }
    }
}

```

```

    }
}
break;

case ES_INCORRECT_INPUT:
{
    // Update sequence state machine
    CurrentState = SequenceCreate;

    // Inform GameState machine
    ES_Event_t GameStateEvent;
    GameStateEvent.EventType = ES_GAME_COMPLETE;
    GameStateEvent.EventParam = score;
    PostGameState(GameStateEvent);
}
break;

case ES_CORRECT_INPUT:
{
    updateScore();
    seqIndex++;
}
break;

case ES_CORRECT_INPUT_FINAL:
{
    updateScore();

    // Inform Display service
    ES_Event_t DisplayEvent;
    DisplayEvent.EventType = ES_DISPLAY_PLAY_UPDATE;
    DisplayEvent.EventParam = bitPack(score, playtimeLeft,
input);

    PostDisplay(DisplayEvent);

    // Update sequence state machine
    CurrentState = SequenceCreate;
    ES_Event_t SequenceEvent;
    SequenceEvent.EventType = ES_NEXT_ROUND;
    PostSequence(SequenceEvent);

    // Inform GameState machine
    ES_Event_t GameStateEvent;
    GameStateEvent.EventType = ES_ROUND_COMPLETE;
    PostGameState(GameStateEvent);
}
break;

case ES_MASTER_RESET:
{
    masterReset();
}
break;

default: {} break;
}
}
break;

default: {} break;
}

```

```

    return ReturnEvent;
}

/*****
Function
    CheckXYVal
Parameters
    void
Returns
    bool, true for event detected and posting, false for no event detected
Description

Event Checker
    This event checker takes reads the joystick x and y values when
    the touch button is pressed, preserving the input the user wants to give
to
    the game. Event Checker compares user input to sequence pattern and post
    Correct Input or Incorrect Input

Second Function: Post to Master Reset if the JoyStick is not in neutral
    position
*****/
bool CheckXYVal (void)
{
    static bool returnValue = false;
    static uint8_t currentTouchSensor;

    // Only checks during the SequenceInput state
    if ((CurrentState == SequenceInput) && (seqIndex <= (arrayLength - 1)))
    {
        ES_Event_t JoystickEvent;
        // Read Current Touch Sensor
        currentTouchSensor = PORTBbits.RB4;

        // Decision Matrix for executable action
        if (currentTouchSensor == lastTouchSensor)
        {
            // Do nothing; user has not decided on input if both are zero
            // or user has not released Z button

            returnValue = false;
        }
        else if (currentTouchSensor == 1 && lastTouchSensor == 0)
        {
            // Read X and Y values from Joystick
            ADC_MultiRead(adcResults);
            lastTouchSensor = currentTouchSensor;
            returnValue = true;
        }
        else if (lastTouchSensor == 1 && currentTouchSensor == 0)
        {
            printf("ADC %d      ", adcResults[0]);
            printf("ADC %d      \r\n", adcResults[1]);

            // Check if this is the last input to post correct event
            if (seqIndex < (arrayLength - 1)) // Not last input
            {
                //printf("seqIndex %d\r\n", seqIndex);
                if (inputChecker(adcResults) == true)
                {

```

```

        // Post Correct Event
        JoystickEvent.EventType = ES_CORRECT_INPUT;
        PostSequence(JoystickEvent);
        //printf("posted Correct Input\r\n");
    }
    else
    {
        // Post Incorrect Event
        JoystickEvent.EventType = ES_INCORRECT_INPUT;
        PostSequence(JoystickEvent);
        //printf("posted Incorrect Input\r\n");
    }
}
else if (seqIndex == (arrayLength - 1))    // Last input
{
    //printf("seqIndex2 %d\r\n",seqIndex);
    if (inputChecker(adcResults) == true)
    {
        //Post Correct Final Event
        JoystickEvent.EventType = ES_CORRECT_INPUT_FINAL;
        PostSequence(JoystickEvent);
        //printf("posted Correct Input F\r\n");
    }
    else
    {
        //Post Incorrect Event
        JoystickEvent.EventType = ES_INCORRECT_INPUT;
        PostSequence(JoystickEvent);
        //printf("posted Incorrect Input\r\n");
    }
}
lastTouchSensor = currentTouchSensor;
returnValue = true;
}
}

// Master Reset Code
ADC_MultiRead(adcResults);
Input_Direction(adcResults);
if (input != 8)
{
    ES_Event_t InputEvent;
    InputEvent.EventType = ES_INPUT_DETECTED;
    PostMasterReset(InputEvent);
}

return returnValue;
}

/*****
Function
    inputChecker
Parameters
    pointer adcResults
Returns
    bool, true for user input same as sequence pattern, false otherwise
Description

    This function compares the input of the X, Y axis of joystick and
    compares
    that input to the sequence of directions, being currently analyzed

```

```

*****/
static bool inputChecker(uint32_t *adcResults)
{
    static bool returnValue = false;
    // Switch case to analyze direction
    if(seqArray[seqIndex] == Input_Direction(adcResults))
    {
        returnValue = true;
    }
    else
    {
        returnValue = false;
    }

    return returnValue;
}

/*****
Function
    updateScore
Parameters
    void
Returns

Description

    Updates score everytime a user enters a correct input, double score for
    every 4 correct inputs
*****/
static void updateScore(){
    if (arrayLength <= 4) {
        score = score + 1;
    } else {
        score = score + (arrayLength / 4) * 1;
    }
}

/*****
Function
    bitPack

Parameters
    three uint8_ts representing score, time and input values

Returns
    uint16_t, where first 8 bits is the score, next 4 bits is the time and
    final 4 bits is the input value

Description
    sets current state to SequenceCreate

Author
    K. Cao, 10/29/20
*****/
static uint16_t bitPack(const uint8_t score, const uint8_t time, const
uint8_t input){
    uint16_t EventParam = score << 8;
    EventParam = EventParam | (time << 4);
    EventParam = EventParam | (input);
    return EventParam;
}

```

```

}

/*****
Function
    masterReset
Parameters
    void
Returns

Description

    sets current state to SequenceCreate
*****/
static void masterReset(){
    CurrentState = SequenceCreate;
}

/*****
Function
    Input_Direction
Parameters
    pointer adcResults
Returns
    uint8_t input
Description

    categorizes the X and Y position of the Joystick into 8 cardinal
directions
*****/
uint8_t Input_Direction(uint32_t *adcResults)
{
    //Direction being analyzed
    if((adcResults[1] > 1) && (adcResults[1] < (Neutral[1] - 10))
&& (adcResults[0] >= (Neutral[0] - 20)) && (adcResults[0] <= (Neutral[0] +
20)))
    {
        input = 0;
    }
    else if((adcResults[1] <= 1) && (adcResults[0] >= (Neutral[0] -
20)) && (adcResults[0] <= (Neutral[0] + 20)))
    {
        input = 1;
    }
    else if((adcResults[1] > (Neutral[1] + 10)) && (adcResults[1] <
1020) && (adcResults[0] >= (Neutral[0] - 20)) && (adcResults[0] <=
(Neutral[0] + 20)))
    {
        input = 2;
    }
    else if((adcResults[1] >= 1020) && (adcResults[0] >=
(Neutral[0] - 20)) && (adcResults[0] <= (Neutral[0] + 20)))
    {
        input = 3;
    }
    else if((adcResults[0] > 1) && (adcResults[0] < (Neutral[0] -
10)) && (adcResults[1] >= (Neutral[1] - 20)) && (adcResults[1] <=
(Neutral[1] + 20)))
    {
        input = 4;
    }
}

```

```
        else if((adcResults[0] <= 1) && (adcResults[1] >= (Neutral[1] -
20)) && (adcResults[1] <= (Neutral[1] + 20)))
        {
            input = 5;
        }
        else if((adcResults[0] > (Neutral[0] + 10)) && (adcResults[0] <
1020) && (adcResults[1] >= (Neutral[1] - 20)) && (adcResults[1] <=
(Neutral[1] + 20)))
        {
            input = 6;
        }
        else if((adcResults[0] >= 1020) && (adcResults[1] >=
(Neutral[1] - 20)) && (adcResults[1] <= (Neutral[1] + 20)))
        {
            input = 7;
        }
        else
        {
            input = 8;
        }

    return input;
}
```


DISPLAY STATE MACHINE

```

/*****
Header file for the Display State Machine

based on the Gen2 Events and Services Framework
*****/

#ifndef Display_H
#define Display_H

// Event Definitions
#include "ES_Configure.h" /* gets us event definitions */
#include "ES_Types.h"     /* gets bool type for returns */

// typedefs for the states
// State definitions for use with the query function
typedef enum
{
    DisplayInitPState, DisplayAvailable, DisplayBusy
}DisplayState_t;

// Public Function Prototypes

bool InitDisplay(uint8_t Priority);
bool PostDisplay(ES_Event_t ThisEvent);
ES_Event_t RunDisplay(ES_Event_t ThisEvent);
DisplayState_t QueryDisplay(void);
bool Check4WriteDone(void);

#endif /* GameState_H */

/*****
Module
    Display.c

Revision
    1.0.0

Description
    This is a state machine that creates the different screens for the game
    and writes them to the OLED display.

Notes

History
When      Who      What/Why
-----
11/03/20  acg      added demo screen
10/30/20  kcao     integration with GameState and Seq
10/28/20  acg      first pass
*****/
/*----- Include Files -----*/
/* include header files for this state machine as well as any machines at
the
    next lower level in the hierarchy that are sub-machines to this machine

```

```

*/
// Hardware
#include <xc.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <proc/p32mx170f256b.h>
#include <sys/attrs.h> // for ISR macros

// Event & Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_DeferRecall.h"
#include "ES_ShortTimer.h"
#include "ES_Port.h"
#include "EventCheckers.h"

// OLED
#include "../u8g2Headers/common.h"
#include "../u8g2Headers/spi_master.h"
#include "../u8g2Headers/u8g2.h"
#include "../u8g2Headers/u8x8.h"

// My Modules
#include "Display.h"
#include "GameState.h"

/*----- Module Defines -----*/
#define LEFT 0
#define SUPERLEFT 1
#define RIGHT 2
#define SUPERRIGHT 3
#define UP 4
#define SUPERUP 5
#define DOWN 6
#define SUPERDOWN 7
#define BLANK 8

/*----- Module Functions -----*/
/* prototypes for private functions for this machine. They should be
functions
relevant to the behavior of this state machine
*/
static void welcomeScreen(void);
static void readyScreen(uint16_t score, uint16_t round);
static void instructionScreen(uint16_t score, uint16_t round, uint16_t
instruction);
static void goScreen(uint16_t score, uint16_t round);
static void playScreen(uint16_t score, uint8_t time, uint8_t input);
static void roundCompleteScreen(uint16_t score, uint16_t round);
static void gameCompleteScreen(void);
void demoScreen(void);
static void bitUnpack(uint16_t EventParam, uint16_t* score, uint8_t* time,
uint8_t* input);

/*----- Module Variables -----*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match that of enum in header file
static DisplayState_t CurrentState;
static uint8_t LastDisplayState;

```

```

// keep track of values needing to be written on the display
static uint16_t score;
static uint8_t time = 15;
static uint8_t input = 8;
static uint16_t round = 1;
static uint16_t instruction;
uint16_t score1;
uint16_t score2;
uint16_t score3;

// with the introduction of Gen2, we need a module level Priority var as
well
static uint8_t MyPriority;

// OLED variables
extern uint8_t u8x8_pic32_gpio_and_delay(u8x8_t *u8x8, uint8_t msg, uint8_t
arg_int, void *arg_ptr);
extern uint8_t u8x8_byte_pic32_hw_spi(u8x8_t *u8x8, uint8_t msg, uint8_t
arg_int, void *arg_ptr);
static u8g2_t u8g2;

// add a deferral queue for up to 2 pending deferrals to allow for overhead
static ES_Event_t DeferralQueue[2];

/*----- Module Code -----*/
/*****
Function
    InitDisplay

Parameters
    uint8_t : the priority of this state machine

Returns
    bool, false if error in initialization, true otherwise

Description
    Saves away the priority, sets up the initial transition and does any
    other required initialization for this state machine

Notes

Author
    A. Gin
*****/
bool InitDisplay(uint8_t Priority)
{
    ES_Event_t ThisEvent;

    MyPriority = Priority;
    //initialize deferral queue
    ES_InitDeferralQueueWith(DeferralQueue, ARRAY_SIZE(DeferralQueue));
    // put us into the Initial PseudoState
    CurrentState = DisplayInitPState;
    // post the initial transition event
    ThisEvent.EventType = ES_INIT;
    if (ES_PostToService(MyPriority, ThisEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

```

    }
}

/*****
Function
    PostDisplay

Parameters
    ES_Event_t ThisEvent , the event to post to the queue

Returns
    boolean False if the Enqueue operation failed, True otherwise

Description
    Posts an event to this state machine's queue

Notes

Author
    A. Gin
*****/
bool PostDisplay(ES_Event_t ThisEvent)
{
    return ES_PostToService(MyPriority, ThisEvent);
}

/*****
Function
    RunDisplay

Parameters
    ES_Event_t : the event to process

Returns
    ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
    displays different screens to OLED based on state

Notes
    uses nested switch/case to implement the machine.

Author
    A. Gin
*****/
ES_Event_t RunDisplay(ES_Event_t ThisEvent)
{
    ES_Event_t ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    switch (CurrentState)
    {
        /*----- DisplayInitPState*/
        case DisplayInitPState:
        {
            if (ThisEvent.EventType == ES_INIT)
            {
                SPI_Init_Display(); //initialize SPI1
                //build up the u8g2 structure with the proper values for our
display
                u8g2_Setup_ssd1306_128x64_noname_f(&u8g2, U8G2_R0,
u8x8_byte_pic32_hw_spi,

```

```

        u8x8_pic32_gpio_and_delay);
// pass all that stuff on to the display to initialize it
u8g2_InitDisplay(&u8g2);
// turn off power save so that the display will be on
u8g2_SetPowerSave(&u8g2, 0);
// choose the font. this one is mono-spaced and has reasonable size
u8g2_SetFont(&u8g2, u8g2_font_t0_18_mr);
// overwrite the background color of newly written characters
u8g2_SetFontMode(&u8g2, 0);

//set display state value for event checker
LastDisplayState = u8g2_NextPage(&u8g2);

//transition to available state
CurrentState = DisplayAvailable;
}
}
break;

/*----- DisplayAvailable*/
case DisplayAvailable:
{
    if (ThisEvent.EventType == ES_DISPLAY_WELCOME)
    {
        welcomeScreen(); // display welcome screen
        score = 0;
        CurrentState = DisplayBusy; // transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_READY)
    {
        round = ThisEvent.EventParam; // update round
        readyScreen(score, round); // display ready screen
        CurrentState = DisplayBusy; // transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_INSTRUCTION)
    {
        instruction = ThisEvent.EventParam; // update instruction
        instructionScreen(score, round, instruction); // display
instruction screen
        CurrentState = DisplayBusy; //transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_GO)
    {
        goScreen(score, round); // display go screen
        CurrentState = DisplayBusy; // transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_PLAY_UPDATE)
    {
        bitUnpack(ThisEvent.EventParam, &score, &time, &input);
        playScreen(score, time, input); // display play screen
        CurrentState = DisplayBusy; // transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_ROUNDCOMPLETE)
    {
        roundCompleteScreen(score, round); // display round complete
screen

```

```

        CurrentState = DisplayBusy;           // transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_GAMECOMPLETE)
    {
screen      gameCompleteScreen();           // display game complete
        CurrentState = DisplayBusy;           // transition to busy state
    }

    if (ThisEvent.EventType == ES_DISPLAY_DEMO)
    {
        demoScreen();                       // display demo screen
        CurrentState = DisplayBusy;           // transition to busy state
    }
}
break;

/*----- DisplayBusy*/
case DisplayBusy:
{
    if (ThisEvent.EventType == ES_UPDATE_COMPLETE)
    {
        //recall the deferred event
        ES_RecallEvents(MyPriority, DeferralQueue);
        //transition to available state
        CurrentState = DisplayAvailable;
    }

    if (ThisEvent.EventType == ES_DISPLAY_WELCOME)
    {
        ES_DeferEvent(DeferralQueue, ThisEvent); // defer event
    }

    if (ThisEvent.EventType == ES_DISPLAY_READY)
    {
        ES_DeferEvent(DeferralQueue, ThisEvent); // defer event
    }

    if (ThisEvent.EventType == ES_DISPLAY_INSTRUCTION)
    {
        ES_DeferEvent(DeferralQueue, ThisEvent); // defer event
    }

    if (ThisEvent.EventType == ES_DISPLAY_PLAY_UPDATE)
    {
        ES_DeferEvent(DeferralQueue, ThisEvent); // defer event
    }

    if (ThisEvent.EventType == ES_DISPLAY_ROUNDCOMPLETE)
    {
        ES_DeferEvent(DeferralQueue, ThisEvent); // defer event
    }

    if (ThisEvent.EventType == ES_DISPLAY_GAMECOMPLETE)
    {
        ES_DeferEvent(DeferralQueue, ThisEvent); // defer event
    }

    if (ThisEvent.EventType == ES_DISPLAY_DEMO)
    {

```

```

        ES_DeferEvent(DeferralQueue, ThisEvent);    // defer event
    }
}
break;

// repeat state pattern as required for other states
default:
    ;
}
// end switch on Current State
return ReturnEvent;
}

/*****
Function
    QueryDisplay

Parameters
    None

Returns
    DisplayState_t The current state of the Template state machine

Description
    returns the current state of the Display state machine

Notes

Author
    A. Gin
*****/
DisplayState_t QueryDisplay(void)
{
    return CurrentState;
}

/*****
private functions
*****/
/*****
Function
    welcomeScreen
Parameters
    Nothing
Returns
    Nothing
Description
    Creates and displays the welcome screen
Author
    A. Gin
*****/
static void welcomeScreen(void)
{
    // clear screen
    u8g2_FirstPage(&u8g2);
    // write game name to display
    u8g2_DrawStr(&u8g2, 1, 15, " KEEP COPYING ");
    u8g2_DrawStr(&u8g2, 1, 30, "  AND NOBODY ");
    u8g2_DrawStr(&u8g2, 1, 45, "   EXPLODES  ");
    // write start instructions to display
    u8g2_DrawStr(&u8g2, 1, 60, " press button");
    // set last display state to busy
    LastDisplayState = 1;
}

```

```

}

/*****
Function
    readyScreen
Parameters
    round
Returns
    Nothing
Description
    Creates and displays the ready screen
Author
    A. Gin
*****/
static void readyScreen(uint16_t score, uint16_t round)
{
    // multiply score by 10 to get actual score
    score = score * 10;
    // turn round into a string and add it to "R"
    char roundstring[4];
    sprintf(roundstring, "R%i", round);

    // turn score into a string
    char scorestring[5];
    sprintf(scorestring, "%i", score);

    // clear screen
    u8g2_FirstPage(&u8g2);
    // write READY to the display
    u8g2_DrawStr(&u8g2, 45, 40, "READY");
    // write the round number to the display
    u8g2_DrawStr(&u8g2, 1, 15, roundstring);
    // write the score to the display, align text with left side
    if (score < 10) //score is 1 number
wide
    {
        u8g2_DrawStr(&u8g2, 120, 15, scorestring);
    }
    else if ((score >= 10) && (score < 100)) //score is 2 numbers
wide
    {
        u8g2_DrawStr(&u8g2, 110, 15, scorestring);
    }
    else if ((score >= 100) && (score < 1000)) //score is 3 numbers
wide
    {
        u8g2_DrawStr(&u8g2, 100, 15, scorestring);
    }
    else //score is 4 numbers
wide
    {
        u8g2_DrawStr(&u8g2, 90, 15, scorestring);
    }
    // set last display state to busy
    LastDisplayState = 1;
}

/*****
Function
    instructionScreen
Parameters

```


score, round, instruction
Returns
Nothing
Description
Creates and displays the instruction screen
Author
A. Gin

```
*****  
static void instructionScreen(uint16_t score, uint16_t round, uint16_t  
instruction)  
{  
    // turn round into a string and add it to "R"  
    char roundstring[4];  
    sprintf(roundstring, "R%i", round);  
  
    // turn score into a string  
    score = score * 10;  
    char scorestring[5];  
    sprintf(scorestring, "%i", score);  
  
    // clear screen  
    u8g2_FirstPage(&u8g2);  
    // write the round number to the display  
    u8g2_DrawStr(&u8g2, 1, 15, roundstring);  
    // write the score to the display, align text with left side  
    if (score < 10) //score is 1 no wide  
    {  
        u8g2_DrawStr(&u8g2, 120, 15, scorestring);  
    }  
    else if ((score >= 10) && (score < 100)) // score is 2 no wide  
    {  
        u8g2_DrawStr(&u8g2, 110, 15, scorestring);  
    }  
    else if ((score >= 100) && (score < 1000)) // score is 3 no wide  
    {  
        u8g2_DrawStr(&u8g2, 100, 15, scorestring);  
    }  
    else // score is 4 no wide  
    {  
        u8g2_DrawStr(&u8g2, 90, 15, scorestring);  
    }  
  
    // write the direction to the screen  
    if (instruction == LEFT)  
    {  
        u8g2_SetFontDirection(&u8g2, 3); // font direction 270 deg  
        u8g2_DrawStr(&u8g2, 65, 15, ">"); // super up arrow  
        u8g2_DrawStr(&u8g2, 65, 25, ">"); // up arrow  
        u8g2_SetFontDirection(&u8g2, 0); // reset font direction  
  
        u8g2_DrawStr(&u8g2, 28, 40, "<"); // super left arrow  
        // highlight on  
        u8g2_SetFontMode(&u8g2, 0);  
        u8g2_SetDrawColor(&u8g2, 0);  
        u8g2_DrawStr(&u8g2, 38, 40, "<"); // left arrow  
        // highlight off  
        u8g2_SetFontMode(&u8g2, 0);  
        u8g2_SetDrawColor(&u8g2, 1);  
  
        u8g2_DrawStr(&u8g2, 75, 40, ">"); // right arrow  
        u8g2_DrawStr(&u8g2, 85, 40, ">"); // super right arrow  
    }  
}
```

```

    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");         // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");         // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
}

if (instruction == SUPERLEFT)
{
    u8g2_SetFontDirection(&u8g2, 3);         // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");         // up arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 28, 40, "<");         // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");         // left arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 75, 40, ">");         // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");         // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);         // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");         // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");         // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
}

if (instruction == RIGHT)
{
    u8g2_SetFontDirection(&u8g2, 3);         // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");         // up arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");         // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");         // left arrow

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 75, 40, ">");         // right arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);
    u8g2_DrawStr(&u8g2, 85, 40, ">");         // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);         // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");         // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");         // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
}

if (instruction == SUPERRIGHT)
{
    u8g2_SetFontDirection(&u8g2, 3);         // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow

```

```

u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
u8g2_SetFontDirection(&u8g2, 0);           // reset font direction

u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

// highlight on
u8g2_SetFontMode(&u8g2, 0);
u8g2_SetDrawColor(&u8g2, 0);
u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow
// highlight off
u8g2_SetFontMode(&u8g2, 0);
u8g2_SetDrawColor(&u8g2, 1);

u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
}

if (instruction == UP)
{
    u8g2_SetFontDirection(&u8g2, 3);       // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");       // super up arrow
    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 65, 25, ">");       // up arrow
    u8g2_SetFontDirection(&u8g2, 0);       // reset font direction
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 28, 40, "<");       // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");       // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");       // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");       // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);       // font direction 90
deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");       // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");       // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);       // reset font direction
}

if (instruction == SUPERUP)
{
    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_SetFontDirection(&u8g2, 3);       // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");       // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");       // up arrow
    u8g2_SetFontDirection(&u8g2, 0);       // reset font direction
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 28, 40, "<");       // super left arrow

```

```

    u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
}

if (instruction == DOWN)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");           // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);
    u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
}

if (instruction == SUPERDOWN)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");           // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);
}

```

```

if (instruction == BLANK)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");         // up arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");         // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");         // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");         // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");         // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);         // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");         // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");         // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
}

// set last display state to busy
LastDisplayState = 1;
}

/*****
Function
goScreen
Parameters
score, round
Returns
Nothing
Description
Creates and displays the go screen
Author
A. Gin
*****/
static void goScreen(uint16_t score, uint16_t round)
{
    // turn round into a string and add it to "R"
    score = score * 10;
    char roundstring[4];
    sprintf(roundstring, "R%i", round);

    // turn score into a string
    char scorestring[5];
    sprintf(scorestring, "%i", score);

    // clear screen
    u8g2_FirstPage(&u8g2);
    // write GO to the display
    u8g2_DrawStr(&u8g2, 55, 40, "GO!");
    // write the round number to the display
    u8g2_DrawStr(&u8g2, 1, 15, roundstring);
    // write the score to the display, align text with left side
    if (score < 10) // score is 1 number wide
    {
        u8g2_DrawStr(&u8g2, 120, 15, scorestring);
    }
    else if ((score >= 10) && (score < 100)) // score is 2 numbers wide
    {
        u8g2_DrawStr(&u8g2, 110, 15, scorestring);
    }
}

```

```

else if ((score >= 100) && (score < 1000)) // score is 3 numbers wide
{
    u8g2_DrawStr(&u8g2, 100, 15, scorestring);
}
else // score is 4 numbers wide
{
    u8g2_DrawStr(&u8g2, 90, 15, scorestring);
}
// set last display state to busy
LastDisplayState = 1;
}

/*****
Function
    playScreen
Parameters
    score, time, input
Returns
    Nothing
Description
    Creates and displays the gameplay screen
Author
    A. Gin
*****/
static void playScreen(uint16_t score, uint8_t time, uint8_t input)
{
    // turn round into a string and add it to "R"
    char roundstring[4];
    sprintf(roundstring, "R%i", round);

    // multiply score by 10 and turn score into a string
    score = score * 10;
    char scorestring[5];
    sprintf(scorestring, "%i", score);

    // turn time into a string
    char timestring[3];
    sprintf(timestring, "%i", time);

    // clear screen
    u8g2_FirstPage(&u8g2);
    // write the round number to the display
    u8g2_DrawStr(&u8g2, 1, 15, roundstring);

    // write the time to the display
    if (time < 10) // time is 1 number wide
    {
        u8g2_DrawStr(&u8g2, 120, 60, timestring);
    }
    if ((time >= 10) && (time <= 15)) // time is 2 numbers wide
    {
        u8g2_DrawStr(&u8g2, 110, 60, timestring);
    }

    // write the score to the display, align text with left side
    if (score < 10) //score is 1 number wide
    {
        u8g2_DrawStr(&u8g2, 120, 15, scorestring);
    }
    else if ((score >= 10) && (score < 100)) // score is 2 numbers wide
    {

```

```

    u8g2_DrawStr(&u8g2, 110, 15, scorestring);
}
else if ((score >= 100) && (score < 1000)) // score is 3 numbers wide
{
    u8g2_DrawStr(&u8g2, 100, 15, scorestring);
}
else // score is 4 numbers wide
{
    u8g2_DrawStr(&u8g2, 90, 15, scorestring);
}

// write the direction to the screen
if (input == LEFT)
{
    u8g2_SetFontDirection(&u8g2, 3); // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">"); // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">"); // up arrow
    u8g2_SetFontDirection(&u8g2, 0); // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<"); // super left arrow
    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 38, 40, "<"); // left arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 75, 40, ">"); // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">"); // super right arrow

    u8g2_SetFontDirection(&u8g2, 1); // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">"); // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">"); // super down arrow
    u8g2_SetFontDirection(&u8g2, 0); // reset font direction
}

if (input == SUPERLEFT)
{
    u8g2_SetFontDirection(&u8g2, 3); // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">"); // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">"); // up arrow
    u8g2_SetFontDirection(&u8g2, 0); // reset font direction

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 28, 40, "<"); // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<"); // left arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 75, 40, ">"); // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">"); // super right arrow

    u8g2_SetFontDirection(&u8g2, 1); // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">"); // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">"); // super down arrow
    u8g2_SetFontDirection(&u8g2, 0); // reset font direction
}

```

```

if (input == RIGHT)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");         // up arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");         // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");         // left arrow

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 75, 40, ">");         // right arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);
    u8g2_DrawStr(&u8g2, 85, 40, ">");         // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);         // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");         // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");         // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
}

if (input == SUPERRIGHT)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");         // up arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");         // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");         // left arrow

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 75, 40, ">");         // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");         // super right arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_SetFontDirection(&u8g2, 1);         // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");         // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");         // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
}

if (input == UP)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");         // super up arrow
    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_DrawStr(&u8g2, 65, 25, ">");         // up arrow
    u8g2_SetFontDirection(&u8g2, 0);         // reset font direction
    // highlight off

```



```

    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
}

if (input == SUPERUP)
{
    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");           // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);

    u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
    u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
}

if (input == DOWN)
{
    u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
    u8g2_DrawStr(&u8g2, 65, 15, ">");           // super up arrow
    u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
    u8g2_SetFontDirection(&u8g2, 0);           // reset font direction

    u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
    u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

    u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
    u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

    // highlight on
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 0);
    u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
    u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
    // highlight off
    u8g2_SetFontMode(&u8g2, 0);
    u8g2_SetDrawColor(&u8g2, 1);
}

```

```

        u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
        u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
    }

    if (input == SUPERDOWN)
    {
        u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
        u8g2_DrawStr(&u8g2, 65, 15, ">");           // super up arrow
        u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
        u8g2_SetFontDirection(&u8g2, 0);           // reset font direction

        u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
        u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

        u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
        u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

        // highlight on
        u8g2_SetFontMode(&u8g2, 0);
        u8g2_SetDrawColor(&u8g2, 0);
        u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
        u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
        u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
        u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
        // highlight off
        u8g2_SetFontMode(&u8g2, 0);
        u8g2_SetDrawColor(&u8g2, 1);
    }

    if (input == BLANK)
    {
        u8g2_SetFontDirection(&u8g2, 3);           // font direction 270 deg
        u8g2_DrawStr(&u8g2, 65, 15, ">");           // super up arrow
        u8g2_DrawStr(&u8g2, 65, 25, ">");           // up arrow
        u8g2_SetFontDirection(&u8g2, 0);           // reset font direction

        u8g2_DrawStr(&u8g2, 28, 40, "<");           // super left arrow
        u8g2_DrawStr(&u8g2, 38, 40, "<");           // left arrow

        u8g2_DrawStr(&u8g2, 75, 40, ">");           // right arrow
        u8g2_DrawStr(&u8g2, 85, 40, ">");           // super right arrow

        u8g2_SetFontDirection(&u8g2, 1);           // font direction 90 deg
        u8g2_DrawStr(&u8g2, 55, 45, ">");           // down arrow
        u8g2_DrawStr(&u8g2, 55, 55, ">");           // super down arrow
        u8g2_SetFontDirection(&u8g2, 0);           // reset font direction
    }

    // set last display state to busy
    LastDisplayState = 1;
}

/*****
Function
    roundCompleteScreen
Parameters
    score, round
Returns
    Nothing
Description
    Creates and displays the round complete screen
*****/

```

Author

A. Gin

```
*****/  
static void roundCompleteScreen(uint16_t score, uint16_t round)
```

```
{  
    // turn round into a string and add it to "R"  
    char roundstring[4];  
    sprintf(roundstring, "R%i", round);  
  
    // multiply score by 10 and turn score into a string  
    score = score * 10;  
    char scorestring[5];  
    sprintf(scorestring, "%i", score);  
  
    // clear screen  
    u8g2_FirstPage(&u8g2);  
    // write BOMB DEFUSED! to the display  
    u8g2_DrawStr(&u8g2, 7, 40, "BOMB DEFUSED!");  
    // write the round number to the display  
    u8g2_DrawStr(&u8g2, 1, 15, roundstring);  
    // write the score to the display, align text with left side  
    if (score < 10) // score is 1 number wide  
    {  
        u8g2_DrawStr(&u8g2, 120, 15, scorestring);  
    }  
    else if ((score >= 10) && (score < 100)) // score is 2 numbers wide  
    {  
        u8g2_DrawStr(&u8g2, 110, 15, scorestring);  
    }  
    else if ((score >= 100) && (score < 1000)) // score is 3 numbers wide  
    {  
        u8g2_DrawStr(&u8g2, 100, 15, scorestring);  
    }  
    else // score is 4 numbers wide  
    {  
        u8g2_DrawStr(&u8g2, 90, 15, scorestring);  
    }  
    // write press button to the display  
    u8g2_DrawStr(&u8g2, 1, 60, "press button");  
    // set last display state to busy  
    LastDisplayState = 1;  
}
```

```
*****
```

Function

gameCompleteScreen

Parameters

Nothing

Returns

Nothing

Description

Creates and displays the game complete screen

Author

A. Gin

```
*****/  
static void gameCompleteScreen(void)
```

```
{  
    // clear screen  
    u8g2_FirstPage(&u8g2);  
    // write GAME OVER to the display  
    u8g2_DrawStr(&u8g2, 85, 35, "GAME");  
}
```

```

    u8g2_DrawStr(&u8g2, 85, 50, "OVER");
    // write High Scores to the display
    u8g2_DrawStr(&u8g2, 1, 12, "High Scores");

    //get high score values and turn into strings
    queryHighScores(&score1, &score2, &score3);
    score1 = score1 * 10;
    score2 = score2 * 10;
    score3 = score3 * 10;
    char score1string[8];
    sprintf(score1string, "1. %i", score1);
    char score2string[8];
    sprintf(score2string, "2. %i", score2);
    char score3string[8];
    sprintf(score3string, "3. %i", score3);

    // write high score values to display
    u8g2_DrawStr(&u8g2, 1, 30, score1string);
    u8g2_DrawStr(&u8g2, 1, 45, score2string);
    u8g2_DrawStr(&u8g2, 1, 60, score3string);

    // set last display state to busy
    LastDisplayState = 1;
}

/*****
Function
    demoScreen
Parameters
    Nothing
Returns
    Nothing
Description
    Creates and displays the demo screen
Author
    A. Gin
*****/
static void demoScreen(void)
{
    // clear screen
    u8g2_FirstPage(&u8g2);
    // write DEMO to display
    u8g2_DrawStr(&u8g2, 4, 40, "    DEMO    ");
    // set last display state to busy
    LastDisplayState = 1;
}

/*****
Function
    bitUnpack
Parameters
    EventParam, score, time, input by reference
Returns
    Nothing
Description
    Updates values for score, time, input
Notes
    Note which params are uint16_t vs uint8_t - will need to initialize
    correctly
    Needs to pass score, time and input by reference.
Author

```

```

K. Cao
*****/
static void bitUnpack(const uint16_t EventParam, uint16_t* score, uint8_t*
time, uint8_t* input){
    uint8_t fourBitMask = ((BIT0HI) | (BIT1HI) | (BIT2HI) | (BIT3HI));
    *input = EventParam & fourBitMask;
    *time = (EventParam >> 4) & fourBitMask;
    *score = EventParam >> 8;
}

/*****
event checkers
*****/
/*****
Function
    Check4WriteDone
Parameters
    Nothing
Returns
    bool
Description
    Event checker that checks if display is done writing to screen, then
returns
    true
Author
    A. Gin
*****/
bool Check4WriteDone(void)
{
    uint8_t        CurrentDisplayState;
    bool           ReturnVal = false;

    if (CurrentState == DisplayBusy){ // only check when display is busy
        CurrentDisplayState = u8g2_NextPage(&u8g2);

        // check for display done AND different from last time
        if ((LastDisplayState != CurrentDisplayState) &&(CurrentDisplayState
== 0))
        {
            // Post event to Display Service
            ES_Event_t ThisEvent;
            ThisEvent.EventType = ES_UPDATE_COMPLETE;
            ThisEvent.EventParam = 1;
            ES_PostToService(MyPriority, ThisEvent);
            ReturnVal = true;
        }
        LastDisplayState = CurrentDisplayState; // update the state for next
time
    }
    return ReturnVal;
}

```

DOTSTAR STATE MACHINE

```

/*****
Header file for the Dotstar State Machine

based on the Gen2 Events and Services Framework
*****/

#ifndef Dotstar_H
#define Dotstar_H

// Event Definitions
#include "ES_Configure.h" /* gets us event definitions */
#include "ES_Types.h"     /* gets bool type for returns */

// typedefs for the states
// State definitions for use with the query function
typedef enum
{
    DotstarInitPState, DotstarRed, DotstarGreen, DotstarRandom, DotstarOff
}DotstarState_t;

// Public Function Prototypes

bool InitDotstar(uint8_t Priority);
bool PostDotstar(ES_Event_t ThisEvent);
ES_Event_t RunDotstar(ES_Event_t ThisEvent);
DotstarState_t QueryDotstar(void);
static void dotStar_Write(const uint8_t Bright1, const uint8_t Red1, const
uint8_t Blue1,
    const uint8_t Green1, const uint8_t Bright2, const uint8_t Red2,
const uint8_t Blue2,
    const uint8_t Green2);

#endif /* Dotstar_H */

/*****
Module
    Dotstar.c

Revision
    1.0.0

Description
    This is a state machine that creates the different flashing patterns
for the
    Dotstar

Notes

History
When          Who          What/Why
-----
10/30/20 01:46 acg          first pass
10/31/20 02:43 acg          restructured RunDotstar
*****/

```

```

/*----- Include Files -----*/
*/
/* include header files for this state machine as well as any machines at
the
next lower level in the hierarchy that are sub-machines to this machine
*/
// Hardware
#include <xc.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <proc/p32mx170f256b.h>
#include <sys/attrs.h> // for ISR macros

// Event & Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_DeferRecall.h"
#include "ES_ShortTimer.h"
#include "ES_Port.h"
#include "EventCheckers.h"

// My Modules
#include "Dotstar.h"
#include "GameState.h"
#include "spi_master.h"

/*----- Module Defines -----*/
// these times assume a 10.000mS/tick timing
#define ONE_SEC 1000
#define QUARTER_SEC (ONE_SEC / 4)

/*----- Module Functions -----*/
/* prototypes for private functions for this machine.They should be
functions
relevant to the behavior of this state machine
*/
void dotStar_Write(uint8_t Bright1, uint8_t Red1, uint8_t Blue1, uint8_t
Green1,
uint8_t Bright2, uint8_t Red2, uint8_t Blue2, uint8_t Green2);

/*----- Module Variables -----*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match that of enum in header file
static DotstarState_t CurrentState;

// with the introduction of Gen2, we need a module level Priority var as
well
static uint8_t MyPriority;

/*----- Module Code -----*/
/*****
Function
InitDotstar

Parameters
uint8_t : the priority of this service

Returns
bool, false if error in initialization, true otherwise
*****/

```

Description

Saves away the priority, sets up the initial transition and does any other required initialization for this state machine

Notes

Author

A. Gin

```
*****/
```

```
bool InitDotstar(uint8_t Priority)
```

```
{
    ES_Event_t ThisEvent;

    MyPriority = Priority;
    // put us into the Initial PseudoState
    CurrentState = DotstarInitPState;
    // post the initial transition event
    ThisEvent.EventType = ES_INIT;
    if (ES_PostToService(MyPriority, ThisEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
*****/
```

Function

PostDotstar

Parameters

ES_Event_t ThisEvent , the event to post to the queue

Returns

boolean False if the Enqueue operation failed, True otherwise

Description

Posts an event to this state machine's queue

Notes

Author

A. Gin

```
*****/
```

```
bool PostDotstar(ES_Event_t ThisEvent)
```

```
{
    return ES_PostToService(MyPriority, ThisEvent);
}
```

```
*****/
```

Function

RunDotstar

Parameters

ES_Event_t : the event to process

Returns

ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description

Flashes red, green, or random lights on the dotstar

Notes

uses nested switch/case to implement the machine.


```

Author
  A. Gin
*****/
ES_Event_t RunDotstar(ES_Event_t ThisEvent)
{
  static uint16_t flipflop; // incremented every timeout and checked if
  even or odd to implement different code within same Event to produce
  flashing LED effect
  uint8_t red1, green1, blue1, red2, green2, blue2;
  ES_Event_t ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  switch (CurrentState)
  {
    /*----- DotstarInitPState*/
    case DotstarInitPState:
    {
      if (ThisEvent.EventType == ES_INIT)
      {
        //initialize SPI for dotstar
        SPI_Init_Dotstar();
        //transition to next case
        CurrentState = DotstarOff;
        // turn off LEDs
        dotStar_Write(0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00);
      }
    }
    break;

    /*----- DotstarRed*/
    case DotstarRed:
    {
      if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam ==
DOTSTAR_TIMER))
      {
        // increment flipflop
        flipflop = flipflop + 1;
        // if flipflop reaches 256, reset to 0
        if (flipflop > 255)
        {
          flipflop = 0;
        }

        if (flipflop%2 == 0) // if flipflop is divisible by 2
        {
          // write LED1 red
          dotStar_Write(0xFF, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0x00,
0x00);
          // set Dotstar timer
          ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
        }
        else // else flipflop is not divisible by 2
        {
          // write LED2 red
          dotStar_Write(0xFF, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x00,
0x00);
          // set Dotstar timer
          ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
        }
      }
    }
  }
}

```

```

if (ThisEvent.EventType == ES_OFF)
{
    // transition to DotstarOff state
    CurrentState = DotstarOff;
    // turn off LEDs
    dotStar_Write(0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00);
}

if (ThisEvent.EventType == ES_GREEN)
{
    // transition to DotstarGreen state
    CurrentState = DotstarGreen;
    // set flipflop to 0
    flipflop = 0;
    // set Dotstar Timer
    ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
}

if (ThisEvent.EventType == ES_RANDOM)
{
    // transition to DotstarRandom state
    CurrentState = DotstarRandom;
    // set flipflop to 0
    flipflop = 0;
    // set Dotstar Timer
    ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
}
}
break;

/*----- DotstarGreen*/
case DotstarGreen:
{
    if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam ==
DOTSTAR_TIMER))
    {
        // increment flipflop
        flipflop = flipflop + 1;
        // if flipflop reaches 256, reset to 0
        if (flipflop > 255)
        {
            flipflop = 0;
        }

        if (flipflop % 2 == 0)    // if flipflop is divisible by 2
        {
            // write LED1 green
            dotStar_Write(0xFF, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00,
0x00);

            // set Dotstar timer
            ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
        }
        else    // else flipflop is not divisible by 2
        {
            // write LED2 green
            dotStar_Write(0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00,
0xFF);

            // set Dotstar timer
            ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
        }
    }
}
}

```

```

if (ThisEvent.EventType == ES_OFF)
{
    // transition to DotstarOff state
    CurrentState = DotstarOff;
    // turn off LEDs
    dotStar_Write(0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00);
}

if (ThisEvent.EventType == ES_RED)
{
    // transition to DotstarRed state
    CurrentState = DotstarRed;
    // set flipflop to 0
    flipflop = 0;
    // set Dotstar Timer
    ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
}

if (ThisEvent.EventType == ES_RANDOM)
{
    // transition to DotstarRandom state
    CurrentState = DotstarRandom;
    // set flipflop to 0
    flipflop = 0;
    // set Dotstar Timer
    ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
}
}
break;

/*----- DotstarRandom*/
case DotstarRandom:
{
    if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam ==
DOTSTAR_TIMER))
    {
        // generate random color values
        red1 = rand();    // LED1
        blue1 = rand();
        green1 = rand();
        red2 = rand();    // LED2
        blue2 = rand();
        green2 = rand();
        // write LED1 and LED2 random colors
        dotStar_Write(0xFF, red1, green1, blue1, 0xFF, red2, green2,
blue2);
        // set Dotstar timer
        ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
    }

if (ThisEvent.EventType == ES_OFF)
{
    // transition to DotstarOff state
    CurrentState = DotstarOff;
    // turn off LEDs
    dotStar_Write(0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00);
}

if (ThisEvent.EventType == ES_RED)
{

```

```

        // transition to DotstarRed state
        CurrentState = DotstarRed;
        // set flipflop to 0
        flipflop = 0;
        // set Dotstar Timer
        ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
    }

    if (ThisEvent.EventType == ES_GREEN)
    {
        // transition to DotstarGreen state
        CurrentState = DotstarGreen;
        // set flipflop to 0
        flipflop = 0;
        // set Dotstar Timer
        ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
    }
}
break;

/*----- DotstarOff*/
case DotstarOff:
{
    if (ThisEvent.EventType == ES_RED)
    {
        // transition to DotstarRed state
        CurrentState = DotstarRed;
        // set flipflop to 0
        flipflop = 0;
        // set Dotstar Timer
        ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
    }

    if (ThisEvent.EventType == ES_GREEN)
    {
        // transition to DotstarGreen state
        CurrentState = DotstarGreen;
        // set flipflop to 0
        flipflop = 0;
        // set Dotstar Timer
        ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
    }

    if (ThisEvent.EventType == ES_RANDOM)
    {
        // transition to DotstarRandom state
        CurrentState = DotstarRandom;
        // set flipflop to 0
        flipflop = 0;
        // set Dotstar Timer
        ES_Timer_InitTimer(DOTSTAR_TIMER, QUARTER_SEC);
    }
}
break;

default:
    ;
}
return ReturnEvent;
}

```

```

/*****
Function
    QueryDotstar

Parameters
    None

Returns
    DotstarState_t The current state of the Template state machine

Description
    returns the current state of the Display state machine

Notes

Author
    A. Gin
*****/
DotstarState_t QueryDotstar(void)
{
    return CurrentState;
}

/*****
private functions
*****/
/*****
Function
    dotStar_Write
Parameters
    Bright1, Red1, Blue1, Green1, Bright2, Red2, Blue2, Green2
Returns
    Nothing
Description
    writes specified brightness and RGB values to dotstar
Author
    A. Gin
*****/
static void dotStar_Write(const uint8_t Bright1, const uint8_t Red1,
    const uint8_t Blue1, const uint8_t Green1, const uint8_t Bright2,
    const uint8_t Red2, const uint8_t Blue2, const uint8_t Green2){
    uint32_t data;
    //write start frame
    SPI_Write(0);

    //write first LED
    data = ((Bright1 << 24) | (Blue1 << 16) | (Green1 << 8) | (Red1));
    SPI_Write(data);

    //write second LED
    data = ((Bright2 << 24) | (Blue2 << 16) | (Green2 << 8) | (Red2));
    SPI_Write(data);

    //write reset frame
    SPI_Write(0);

    //write end frame
    SPI_Write(0);
}

```

MASTER RESET SERVICE

```

/*****

Header file for the Master Reset Service

based on the Gen2 Events and Services Framework

*****/

#ifndef MasterReset_H
#define MasterReset_H

// Event Definitions
#include "ES_Configure.h" /* gets us event definitions */
#include "ES_Types.h" /* gets bool type for returns */

// typedefs for the states
// State definitions for use with the query function
typedef enum
{
    InitMR, Waiting
} MasterResetState_t;

// Public Function Prototypes

bool InitMasterReset(uint8_t Priority);
bool PostMasterReset(ES_Event_t ThisEvent);
ES_Event_t RunMasterReset(ES_Event_t ThisEvent);

#endif /* MasterReset_H */

/*****
Module
    MasterReset.c

Revision
    1.0.0

Description
    MasterReset is a file that resets the game once no inputs have been
    detected for 30 seconds.

Notes

History
When      Who      What/Why
-----
10/30/20   kcao   Creation of file
*****/
/*----- Include Files -----*/
/* include header files for this state machine as well as any machines at
the
next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "MasterReset.h"
#include "GameState.h"
#include "Seq.h"

```

```

/*----- Module Defines -----*/

/*----- Module Functions -----*/
/* prototypes for private functions for this machine.They should be
functions
relevant to the behavior of this state machine
*/

/*----- Module Variables -----*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match that of enum in header file
static MasterResetState_t CurrentState;

// with the introduction of Gen2, we need a module level Priority var as
well
static uint8_t MyPriority;

/*----- Module Code -----*/
/*****
Function
    InitMasterReset

Parameters
    uint8_t : the priority of this service

Returns
    bool, false if error in initialization, true otherwise

Description
    Saves away the priority, sets up the initial transition and does any
    other required initialization for this state machine

Notes

Author
    K. Cao
*****/
bool InitMasterReset(uint8_t Priority)
{
    ES_Event_t InitEvent;
    MyPriority = Priority;
    CurrentState = InitMR;
    InitEvent.EventType = ES_INIT;
    if (ES_PostToService(MyPriority, InitEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/*****
Function
    PostMasterReset

Parameters
    EF_Event_t ThisEvent , the event to post to the queue

Returns
    boolean False if the Enqueue operation failed, True otherwise
*****/

```

Description

Posts an event to this state machine's queue

Notes

Author

K. Cao

```
*****/
```

```
bool PostMasterReset(ES_Event_t ThisEvent)
{
    return ES_PostToService(MyPriority, ThisEvent);
}
```

```
*****/
```

Function

RunMasterReset

Parameters

ES_Event_t : the event to process

Returns

ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description

Implementation of master reset function when 30 seconds without an input expires. If an input is detected, then the timer resets.

Notes

uses nested switch/case to implement the machine.

Author

K. Cao

```
*****/
```

```
ES_Event_t RunMasterReset(ES_Event_t ThisEvent)
{
    ES_Event_t ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT;

    switch (CurrentState)
    {
        case InitMR:
        {
            if (ThisEvent.EventType == ES_INIT)
            {
                ES_Timer_InitTimer(IDLE_TIMER, 30000);
                CurrentState = Waiting;
            }
        }
        break;

        case Waiting:
        {
            switch (ThisEvent.EventType)
            {
                case ES_INPUT_DETECTED:
                {
                    // Restart timer
                    ES_Timer_InitTimer(IDLE_TIMER, 30000);
                }
                break;
            }
        }
    }
}
```



```
case ES_TIMEOUT:
{
    if (ThisEvent.EventParam == IDLE_TIMER)
    {
        // Update Game State and Sequence FSMs
        ES_Event_t ResetEvent;
        ResetEvent.EventType = ES_MASTER_RESET;
        PostGameState(ResetEvent);
        PostSequence(ResetEvent);
    }
}
break;

default:
    ;
}
}
break;

default:
    ;
}
return ReturnEvent;
}
```