

## GAME STATE MACHINE

State:

InitPState:

ES\_INIT:

- Read initial touch sensor state
- Initialize high scores array
- Update Display with Welcome Screen
- Update Dotstar with Random Colors
- Start DemoTimer (15 seconds)
- Change state to WelcomeScreen

WelcomeScreen:

ES\_SENSOR\_PRESSED:

- Update Display with Ready Screen
- Update Dotstar to Turn Off
- Update Sequence for First Round
- Start ReadyTimer (1 second)
- Change state to GAlLeader

ES\_TIMEOUT (from DemoTimer):

- Update Display with Demo Screen
- Update Sequence for First Round
- Start Demo Screen Timer (1 second)

GAlLeader:

ES\_GAME\_START:

- Update Display with Go Screen
- Start GoTimer (2 seconds)
- Change state to GAFollower

ES\_MASTER\_RESET:

- Complete a master reset

GAFollower:

ES\_ROUND\_COMPLETE:

- Update Display with Round Complete Screen
- Update Dotstar to flash green
- Change state to GARoundComplete

ES\_GAME\_COMPLETE:

- Update Display with Game Complete Screen
- Update Dotstar to flash green/red based on whether high score achieved

- Init Game Over Timer

- Change state to GameComplete

ES\_MASTER\_RESET:

- Complete a master reset

GARoundComplete:

ES\_SENSOR\_PRESSED:

- Update Display to Ready Screen
- Update Dotstar to turn off
- Start ReadyTimer (2 seconds)
- Change state to GAlLeader

ES\_MASTER\_RESET:

- Complete a master reset

```

GameComplete:
    ES_SENSOR_PRESSED:
        Update Display with Welcome Screen
        Update Dotstar with Random Colors
        Start DemoTimer (15 seconds)
        Change state to WelcomeScreen

    ES_TIMEOUT (from GameOverTimer):
        Update Display with Welcome Screen
        Update Dotstar with Random Colors
        Start DemoTimer (15 seconds)
        Change state to WelcomeScreen

    ES_MASTER_RESET:
        Complete a master reset

Demo:
    ES_TIMEOUT (from LastDirectionTimer):
        Complete a master reset
        Update Sequence FSM to complete a master reset.

```

---

## Query Functions

```

CheckTouchSensor:
    If in WelcomeScreen, GARoundComplete, GameComplete or Demo States:
        Read touch sensor input pin.
        If touch sensor pressed to unpressed:
            Update game state FSM of touch sensor press.
            Update master reset FSM of detected input.
        Update last sensor state to current sensor state.

    If touch sensor is changed, post ES_SENSOR_PRESSED to Reset Service
    Event checker for touch sensor from 1 (touching) to 0 (not touching)
    Posts ES_SENSOR_PRESSED when true.

```

---

## Event Checkers

```

CheckTouchSensor:
    If in WelcomeScreen, GARoundComplete or GameComplete States:
        Read touch sensor input pin.
        If sensor state has changed and is low:

    If touch sensor is changed, post ES_SENSOR_PRESSED to Reset Service
    Event checker for touch sensor from 1 (touching) to 0 (not touching)
    Posts ES_SENSOR_PRESSED when true.

```

---

## Private Functions

```

UpdateHighScore:
    Boolean function that updates high scores
    Returns true if high score achieved

```

## SEQUENCE STATE MACHINE

Initialize Module level Variables:

seq\_array, array\_len, score, seq\_idx, play\_time, round, display\_c,  
Current\_State, adcResults, Last\_Zval, Neutral axis

Initialize Sequence Service

Configure and Initialize Analog to Digital function for Joystick Input  
Transition to PseudoInit State

Run Sequence Service

State:

PseudoInit:

Read joystick X and Y values to obtain their

neutral state

Save those values to Neutral Array

Post to Service Event First Round

Transition to SequenceCreate State

SequenceCreate:

ES\_FIRST\_ROUND:

Clear sequence array values

Initialize array\_len to 4 and reset score

and round values

Append 4 random directions to seq\_array

Initialize seqIndex to 0, this will keep

track of sequence when

checking against user input

ES\_NEXT\_ROUND

Append new direction to seq\_array

increase array\_length and round by 1

reset seqIndex

ES\_TIMEOUT (from ReadyTimer):

set display counter to 0

PostToDisplay first direction from seq\_array

Start DirectionTimer 0.75 seconds

Transition to SequenceDisplay State

increment display counter

SequenceDisplay:

Use switch statement to differentiate between timers

Add a short pause between instructions

ES\_TIMEOUT (from Direction\_Pause\_Timer):

PostToDisplay next direction in seq\_array

if display\_counter is less than array\_len -1

Start DirectionTimer 0.75 seconds

if display counter = array\_len -1

Start LAST\_DIRECTION\_TIMER 0.75 seconds

increment display counter

ES\_TIMEOUT (from Direction\_Timer):

```

        PostToDisplay blank direction
        Start Direction_Pause_Timer 0.25 seconds

    ES_TIMEOUT(from GO_TIMER):
        PostToDisplay Gameplay Screen
        Start INPUT_TIMER 1 seconds
        Start InstructionTimer .1 seconds
        Initialize play_time = 0
        Transition to Sequence Input State

SequenceInput:
    ES_TIMEOUT(from INPUT_TIMER):
        if play_time is > 0
            Start INPUT_TIMER 1 second
            decrement play_time by 1

        if play_time = 0
            Transition to SequenceCreate State
            PostToDisplay to transition to gameover
screen

    ES_TIMEOUT(from INSTRUCTION_TIMER):
        Read JoyStick X and Y values
        PostToDisplay Input direction and update all screen
elements

        Start Instruction Timer 0.1 seconds

    ES_INCORRECT_INPUT
        Transition to SequenceCreate State
        PostToDisplay to transition to gameover screen

    ES_CORRECT_INPUT_F
        Post to this service ES_NEXT_ROUND
        PostToDisplay to transition to round won screen
        increment score
        Transition to SequenceCreate State

    ES_CORRECT_INPUT
        increment score
        increment seqIndex

```

-----

Event Checkers

```
bool CheckXYVal(void)
```

```
Initialize returnValue = False
Initialize currentTouchSensor
```

```
Read X and Y values from Joystick
Call Function Input_Direction
```

```
PostToMasterReset if Joystick input was detected
```

```
Minimize amount of time event cheker is running while
joystick input is not required
```

```
currentTouchSensor = read TouchSensor Input
```

```
Read sensor input while player is pressing the button and post
```

such input when the player releases the button

```
if currentTouchSensor = LastTouchSensor
    Do nothing
    returnValue = False
if currentTouchSensor = 1 and LastTouchSensor = 0
    Set LastTouchSensor to currentTouchSensor
    returnValue = true
if LastTouchSensor = 1 and currentTouchSensor = 0

    Check that sequence index is not the last one
    if direction matches input from user
        Post to this service ES_Correct Input
    if the direction does not match the player input
        Post to this service ES_Incorrect Input

    If the sequence index is the last element of array
    if direction matches input from user
        Post to this service ES_Correct Input Final
    if the direction does not match the player input
        Post to this service ES_Incorrect Input
```

-----  
Private Functions

```
InputChecker(Joystick Input)
compare the joystick input to all the possible directions
if the value matches return true
otherwise return false

Input_Direction(Joystick Input)
Assign a cardinal coordinate value to Joystick X and Y output

SequenceSM_PseudoCode.txt
Displaying SequenceSM_PseudoCode.txt.
```

## DISPLAY SERVICE

```
create module-level variables: score1, score2, score3
create module-level static variables: score, time, input, round,
DeferralQueue
set time to 15
set input to 8
set round to 1

InitializeOLED()
    Initialize MyPriority variable
    initialize deferral queue
    set state to pseudostate DisplayInitPState
    post the initial transition event ES_INIT

PostToOLED()
    return ES_PostToService function

RunOLED()
    switch CurrentState
    case DisplayInitPState:
        if event is ES_INIT
            initialize SPI
            build up the u8g2 structure with the proper values for our display
            pass all that stuff on to the display to initialize it
            turn off power save so that the display will be on
            choose the font
            overwrite the background color of newly written characters
            set display state value for event checker
            transition to DisplayAvailable state

    case DisplayAvailable:
        if event is ES_DISPLAY_WELCOME
            call welcomeScreen() to display welcome screen
            set score to 0
            transition to DisplayBusy state

        if event is ES_DISPLAY_READY
            update round to event param
            call readyScreen() to display ready screen
            transition to DisplayBusy state

        if event is ES_DISPLAY_INSTRUCTION
            update instruction to event param
            call instructionScreen() to display instruction screen
            transition to DisplayBusy state

        if event is ES_DISPLAY_GO
            call goScreen() to display go screen
            transition to DisplayBusy state

        if event is ES_DISPLAY_PLAY_INITIAL
            call bitUnpack to update score, time, input values
            call playScreen() to display play screen
            transition to DisplayBusy state

        if event is ES_DISPLAY_PLAY_UPDATE
            update score = ThisEvent.EventParam
            update time = ThisEvent.EventParamTime
            update input = ThisEvent.EventParamInput
            playScreen(score, time, input)
```

```

        transition to DisplayBusy state

    if event is ES_DISPLAY_ROUNDCOMPLETE
        call roundCompleteScreen() to display round complete screen
        transition to DisplayBusy state

    if event is ES_DISPLAY_GAMECOMPLETE
        call gameCompleteScreen() to display game complete screen
        transition to DisplayBusy state

    if event = ES_DISPLAY_DEMO
        call demoScreen() to display demo screen
        transition to DisplayBusy state

case DisplayBusy:
    if event is ES_UPDATE_COMPLETE
        recall the deferred event
        transition to DisplayAvailable state

    if event is ES_DISPLAY_WELCOME
        defer event

    if event is ES_DISPLAY_READY
        defer event

    if event is ES_DISPLAY_INSTRUCTION
        defer event

    if event is ES_DISPLAY_PLAY_UPDATE
        defer event

    if event is ES_DISPLAY_ROUNDCOMPLETE
        defer event

    if event is ES_DISPLAY_GAMECOMPLETE
        defer event

    if event is ES_DISPLAY_DEMO
        defer event

QueryDisplay()
    return current state of service

----- Private Functions -----
welcomeScreen()
    clear screen
    write game name to display
    write start instructions to display
    set last display state to busy

readyScreen(score, round)
    multiply score by 10 to get actual score
    turn round into a string and add it to "R"
    turn score into a string
    clear screen
    write READY to the display
    write the round number to the display
    write the score to the display, align text with left side
    set last display state to busy

```

```

instructionScreen(score, round, instruction)
    turn round into a string and add it to "R"
    turn score into a string
    clear screen
    write the round number to the display
    write the score to the display, align text with left side
    if instruction = 0
        write left arrow to the screen
    if instruction = 1
        write super left arrow to the screen
    if instruction = 2
        write right arrow to the screen
    if instruction = 3
        write super right arrow to the screen
    if instruction = 4
        write up arrow to the screen
    if instruction = 5
        write super up arrow to the screen
    if instruction = 6
        write down arrow to the screen
    if instruction = 7
        write super down arrow to the screen
    if instruction = 8
        write blank arrows to the screen
    set last display state to the busy

```

```

goScreen(score, round)
    turn round into a string and add it to "R"
    turn score into a string
    clear screen
    write GO to the display
    write the round number to the display
    write the score to the display, align text with left side
    set last display state to busy

```

```

playScreen(score, time, input)
    turn round into a string and add it to "R"
    multiple score by 10 and turn score into a string
    turn time into a string
    clear screen
    write the round number to the display
    write the time to the display
    write the score to the display, align text with left side
    if input = 0
        write left arrow to the screen

    if input = 1
        write super left arrow to the screen

    if input = 2
        write right arrow to the screen

    if input = 3
        write super right arrow to the screen

    if input = 4
        write up arrow to the screen

    if input = 5
        write super up arrow to the screen

```



```

if input = 6
    write down arrow to the screen

if input = 7
    write super down arrow to the screen

if input = 8
    write blank arrows to the screen
    set last display state to busy

roundCompleteScreen(score, round)
    turn round into a string and add it to "R"
    multiple score by 10 and turn score into a string
    clear screen
    write BOMB DEFUSED! to the display
    write the round number to the display
    write the score to the display, align text with left side
    write press button to the display
    set last display state to busy

gameCompleteScreen()
    clear screen
    write GAME OVER to the display
    write High Scores to the display
    call queryHighScores() to get high score values and turn into strings
    write high score values to display
    set last display state to busy

demoScreen()
    clear screen
    write DEMO to display
    set last display state to busy

bitUnpack(EventParam, *score, *time, *input)
    get value for input
    get value for time
    get value for score

Check4WriteDone()
    define variable CurrentDisplayState
    define variable ReturnVal, set to false
    if CurrentState = DisplayBusy
        set CurrentDisplayState to next page value
        if display is done and different from last state
            post event ES_UPDATE_COMPLETE to Display service
            set ReturnVal to true
        set LastDisplayState = CurrentDisplayState
    return ReturnVal

```

## DOTSTAR SERVICE

```
InitializeDotstar()
    Initialize MyPriority variable
    Set state to pseudostate DotstarInitPState
    Post the initial transition event ES_INIT

PostToDotstar()
    return ES_PostToService function

RunDotstar()
    define static variable flipflop
    define variables red1, green1, blue1, red2, green2, blue2
    switch CurrentState
        case DotstarInitPState:
            if event is ES_INIT
                initialize SPI for dotstar
                transition to DotstarOff state
                turn off LEDs

        case DotstarRed:
            if event is ES_TIMEOUT from Dotstar Timer
                increment flipflop
                if flipflop reaches 256, reset to 0
                if flipflop is divisible by 2
                    write LED1 red
                    set Dotstar Timer to 0.25s
                else
                    write LED2 red
                    set Dotstar Timer to 0.25s

            if event is ES_OFF
                transition to DotstarOff state
                turn off LEDs

            if event is ES_GREEN
                transition to DotstarGreen state
                set flipflop to 0
                set Dotstar Timer

            if event is ES_RANDOM
                transition to DotstarRandom state
                set flipflop to 0
                set Dotstar Timer

        case DotstarGreen:
            if event is ES_TIMEOUT from Dotstar Timer
                increment flipflop
                if flipflop reaches 256, reset to 0
                if flipflop is divisible by 2
                    write LED1 green
                    set Dotstar Timer to 0.25s
                else
                    write LED2 green
                    set Dotstar Timer to 0.25s

            if event is ES_OFF
                transition to DotstarOff state
                turn off LEDs

            if event is ES_RED
```

```

        transition to DotstarRed state
        set flipflop to 0
        set Dotstar Timer

    if event is ES_RANDOM
        transition to DotstarRandom state
        set flipflop to 0
        set Dotstar Timer

    case DotstarRandom:
        if event is ES_TIMEOUT from Dotstar Timer
            increment flipflop
            if flipflop reaches 256, reset to 0
            if flipflop is divisible by 2
                generate random color values for red1, blue1, green1, red2,
blue2, green2
                write LED1 random
                set Dotstar Timer to 0.25s
            else
                generate random color values for red1, blue1, green1, red2,
blue2, green2
                write LED2 random
                set Dotstar Timer to 0.25s

        if event is ES_OFF
            transition to DotstarOff state
            turn off LEDs

        if event is ES_RED
            transition to DotstarRed state
            set flipflop to 0
            set Dotstar Timer

        if event is ES_GREEN
            transition to DotstarGreen state
            set flipflop to 0
            set Dotstar Timer

```

```

    case DotstarOff
        if event is ES_RED
            transition to DotstarRed state
            set flipflop to 0
            set Dotstar Timer
        if event is ES_GREEN
            transition to DotstarGreen state
            set flipflop to 0
            set Dotstar Timer
        if event is ES_RANDOM
            transition to DotstarRandom state
            set flipflop to 0
            set Dotstar Timer

```

```

QueryDotstar()

```

```

    return current state of service

```

```

----- Private Functions -----

```

```

dotstarWrite(Bright1, Red1, Blue1, Green1, Bright2, Red2, Blue2, Green2)
    write start frame
    write first LED
    write second LED
    write reset frame
    write end frame

```

## MASTER RESET SERVICE

```
InitializeReset()  
    Initialize MyPriority variable  
    Set state to pseudostate  
    Post the initial transition event
```

```
PostToReset()  
    return ES_PostToService function
```

```
RunReset()  
    If Input Detected:  
        Reset IdleTimer  
  
    If IdleTimer Expired:  
        Update Game FSM  
        Update Sequence FSM
```