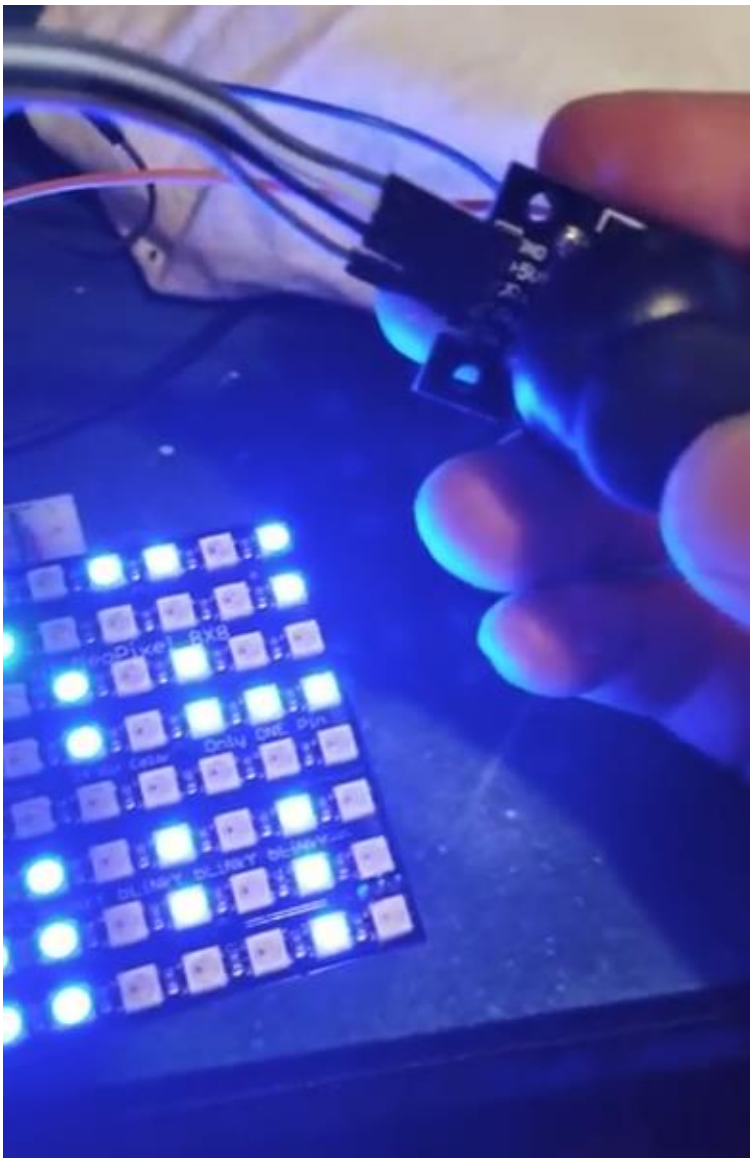


Forever Maze

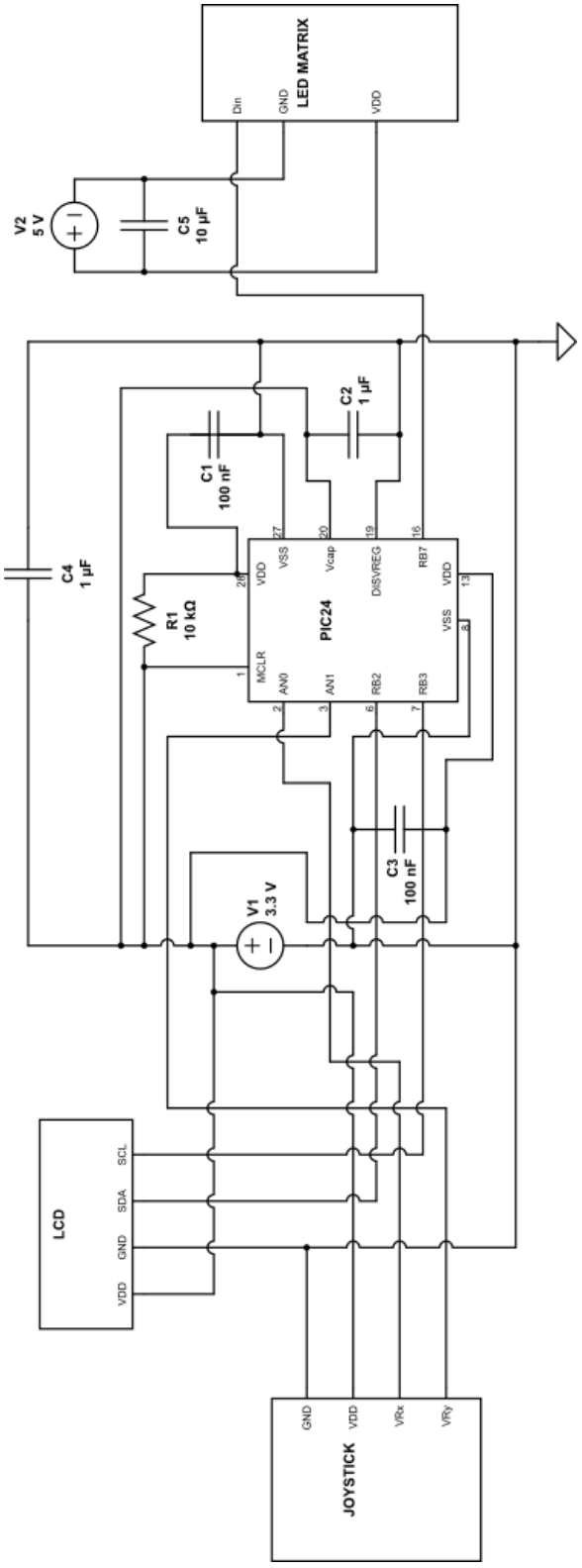
A RANDOMIZED MAZE GAME

Sayontan Roy, Jennifer Buss, Maxwell Danku, Charles Ragona | EE2361 | Final Lab Report

Introduction

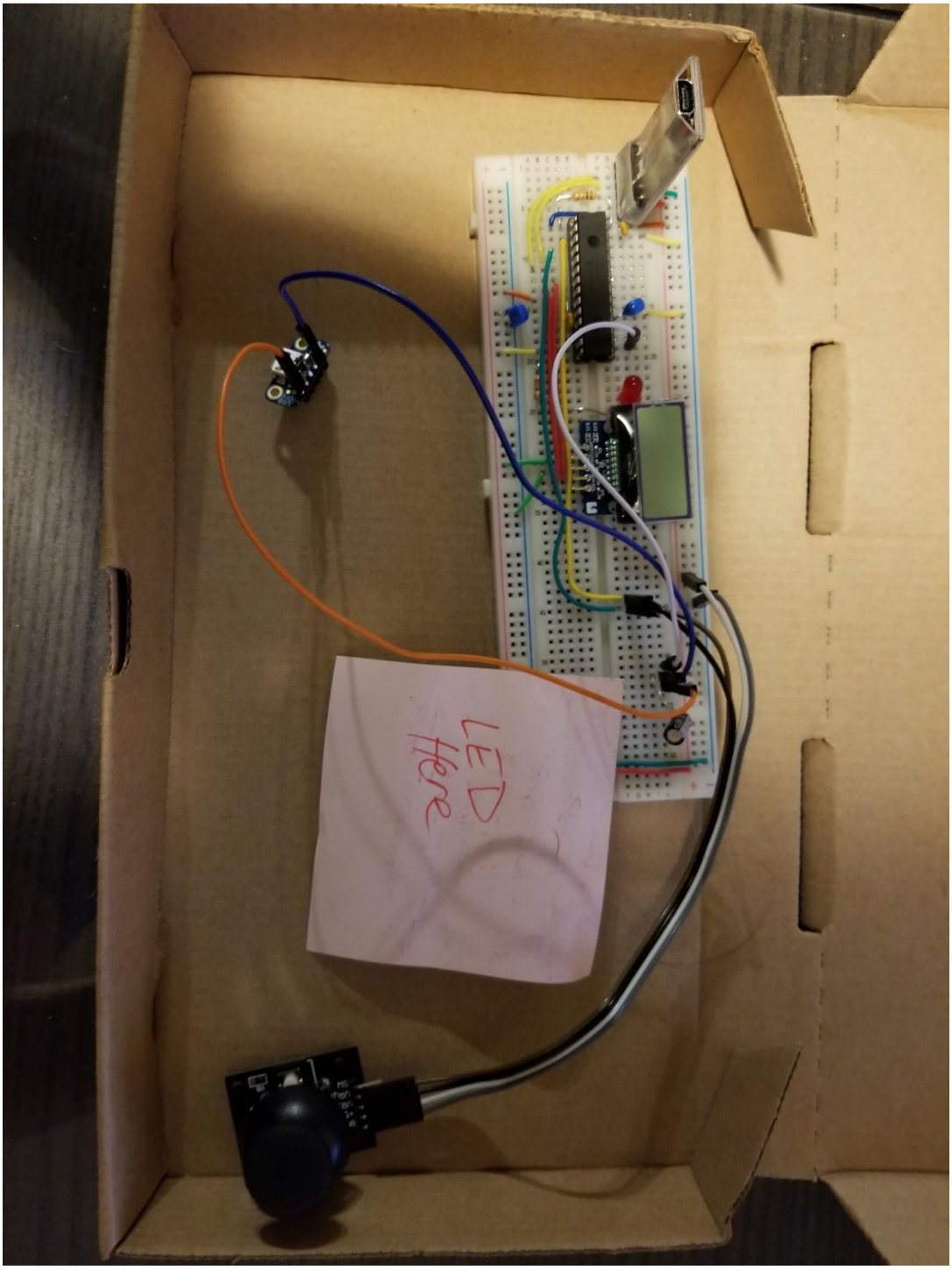
The final project was meant to demonstrate the students' knowledge of microcontrollers and hardware design in an in depth and interactive manner. An interactive Maze game was chosen to be implemented with a haptic joystick and an 8x8 LED matrix. The maze is displayed on the LED matrix and the player must navigate through the maze to get to the other side, while not bumping into the walls of the maze.

Schematic


LED MATRIX
Din
GND
VDD
V2
5 V
C5
10 µF
C2
1 µF
C1
100 nF
C4
1 µF
R1
10 kΩ
PIC24
MCLR
AN0
AN1
RB2
RB3
VSS
VDD
Vcap
DISVREG
RB7
C3
100 nF
V1
3.3 V
LCD
SCL
SDA
GND
VDD
JOYSTICK
GND
VDD
VRx
VRy

Picture of Assembled Design

(minus the LED because we turned in it)



Library Description

The lcd library was mainly derived from Labs 5 and 6. Here lcd_cmd() was used to create a write packet with a start bit, slave address, control byte, data byte, and a stop bit. Calls to lcd_cmd(char c) with 8-bit binary values passed in as parameters were used to initialize the LCD inside the lcdinit(void) function. lcdSetCursor(char row, char col)was used to set the cursor of the LCD to a value for row and column. lcdPrintChar(char c) and lcdString(char *s) had similar purposes, with the main purpose being that the former prints one single character while the latter prints an entire string being passed in.

The joystick library consisted of a few operations. In general, it would read the values from two ADC1 buffers and store them into two separate variables after the ADC1 interrupt has occurred. This was done in order to separate the voltage readings from movements in the x-direction to voltage readings from movements in the y-direction. Additionally, setup of the joystick settings were done through the joystickSetup(void) function. Moreover, the getJoystickDirection(void) function was responsible for the movement of the joystick depending upon the voltage value read in the buffer.

The ws_2812_C_lib library consisted of functions that were derived from Labs 2a & 2b. However, for the purpose of the game, only the writeColor(int r, int g, int b) was used to convert colors into binary values. The write_1() and write_0() assembly functions were called from the bitBangHead library to send out pulses of ones and zeros to produce the desired colors.

Hardware Description

- Microchip pic24 microcontroller
- Analog 2-axis thumb joystick with select button + breakout board
- Adafruit neopixel neomatrix 8x8 - 64 rgb led pixel matrix
- Lcd module
- Adafruit 5 v usb breakout

Function Description

There are a few functions that give the maze life. The first is getJoystickDirection(). This begins the sampling and returns an int as a specified direction which is passed to checkPlayer(int). The checkPlayer(int) then takes this param from getJoystickDirection and passes it through series of bound checking for walls and out of bounds play for the player movement. The player position (a global variable) is then updated accordingly. The play position is just a xy coordinate of the player within the maze matrix. The player is represented by the number 8 in the matrix. After checkPlayer is done we used updateMaze() to remove the previous 8 from the matrix and update the new player position with a new 8. All of this is the basic functionality of player movement.

We created 2 other functions that create our maze and push the pulses to the LED matrix. The function to generate the maze is mazeSetup(). This utilizes a set of 20 mazes and picks one randomly. This maze is passed to a global maze variable which we access in the function that pushes the pulses, writeMaze(). During the write maze function we scan each cell of the matrix to see if there is a wall (indicated by 1), a path (indicated by 0), and a player point (indicated by 8). This pushes the assigned color to the LED on the LED matrix. There is also a "breath" effect inside writeMaze that pulse from blue to red to green. Since writeMaze is called in our while(1) loop this is iterated of a 90 count variable.

There are a few things we do in the main function to control everything. First we initialize our components. After this we enter the while(1) loop. Here we hold the maze is constanct joystick scan and update/write maze functions until the maze cell 7,7 is associated with our player. When the player reaches that cell the loop is exited and we pick a new maze and reset the player position back to 0,0. There are LCDstring writes that pass the direction of joystick movement for debugging purposes. We add a timer 1 1ms delay with an int variable to allow for a frame rate.

Basic usage example

We did this by utilizing the LCD library we built in lab 5. We set the joy stick up and pushed the voltages through the MPLABx debugger to find the range of voltages. There we began to decide the range and pushed the functionality of left, right, down, up to our LCD screen. This was followed up by using the cascade method described in the WS-2812 data sheet sending bits to certain LEDs on the matrix. We constructed the rest of the functionality mainly on these two debugging techniques.

Here is a link to a video recorded after we got the basic functionality working:

https://vimeo.com/268123238

This is the phase before we debugged for the correct movement of cells and adding the breath effect.

Advance usage example

Let's move into talking about the functions in detail.

joystickSetup()

The joystick is connected to the PIC24 through pins AN0 and AN1. To set these pins for input TRISA is set to 0b11. The joystick has two potentiometers so the A/D module is setup to convert the voltage readings from the potentiometers to a digital value. First, the AD1CONx registers are reset. Then selected bits of the registers of the A/D module are set.

We wanted manual sampling and auto-conversion. Thus, from the AD1CON1 register the SSRC bit is set to 0b111 so that the internal counter (auto-convert) conversion trigger source is selected. The ASAM bit was already set to zero by the resetting of the AD1CONx registers so sampling is manual.

Because the joystick is connected to the AN0 and AN1 pins, we wanted sequential scanning of these two pins and an interrupt each time samples from these two pins have been converted. Since the AD1CSSL register selects the pins that will input channels, CSSL0 and CSSL1 bits were set to 1 for sequential scanning on MUX A. And, from the AD1CON2 register, we set the CSCNA, SMPI and ALTS bits. When the CSCNA bit is set to 1, the channels specified by the AD1CSSL register are sequentially sampled. Because sampling is only available if MUX A input is selected, the ALTS bit is set to zero. The SMPI bit is set to 1 so that there is an interrupt at the completion of conversion for each 2nd sample/ convert sequence.

From the AD1CON3 register, the ADCS and SAMC bits are set. Since the minimum A/D clock period must be 75 ns, the ADCS bit is set so that the A/D conversion clock period is:

$$2 \times Tcy = 2 \times \frac{1}{Fcy} = 2 \times \frac{1}{16\ MHz} = 125\ ns$$

The SAMC bit is set to 17 T_{AD}. Actual T_{SAMP} is:

$$17 \text{ x actual } T_{AD} = 17 \text{ x } 125 \text{ ns} = 2.125\ \mu s$$

Thus, the total time to sample and convert is :

$$\text{actual } T_{SAMP} + \text{time to convert} = \text{actual } T_{SAMP} + 12\ T_{AD}$$

$$= 2.125\ \mu s + 12(125\ ns) = 2.125\ \mu s + 1.5\ \mu s = 3.625\ \mu s$$

Then the interrupt bit AD1IE is enabled (= 1), the interrupt flag AD1IF bit is cleared (= 0), and the ADON bit is set to 1 to turn on the A/D module.

The function also includes setup for the LCD display so that the direction of movement could be displayed on the LCD for debugging purposes and for use in the forever maze game. Therefore there is a I2C setup and a timer 1 setup. In the I2C setup the I2C peripheral is disabled so that the baud rate generator settings can be modified. The baud rate was selected to that the I2C would be run at 100 kHz (I2C2BRG = 0x9D). Then the I2C peripheral is enabled and the interrupt flag reset to 0. Timer 1 is setup so that the PR1 = 15999, the interrupt flag is reset and then the timer is turned on.

Finally the setup calls the lcdinit(), lcdSetCursor() and lcdString() functions.

lcdinit

This function initializes the LCD display so that two lines of characters can be viewed on the lcd. First there is a 50 ms delay which is accomplished by polling. Next is a sequence of calls to the lcd_cmd() function:

1. An 8 bit bus mode with MPU, 2 display lines and a 5x8 character font, normal instruction is selected with lcd_cmd(0b00111000).

2. An 8 bit bus mode with MPU, 2 display lines and a 5x8 character font, extension instruction selected with lcd_cmd(0b00111001).

3. The internal OSC frequency is set with lcd_cmd(0b00010100). Specifically, bias is set to 1/5, frame frequency (Hz), two line mode 183-192 for VDD = 3.0 V to VDD = 5.0 V respectively

4-5. The contrast control of the power, ICON, and contrast set of LCD display are set with lcd_cmd(0b01110000) and lcd_cmd(0b01011110). Specifically, lcd_cmd(0b01110000) sets up contrast bits C3:C0 and lcd_cmd(0b01011110) sets up the power/ICON/contrast set with Ion is set to high (ICON display on); Bon is set to high (booster circuit is turned on); and C5:C4 are set.

6. A follower control instruction that sets the internal follower circuit on, Rab2 is set to high, and Rab1:0 are set to low with lcd_cmd(0b01101100).

Then there is a 200 ms delay implemented by another for loop with polling. After the 200 ms delay, there are three more calls to the lcd_cmd() function:

1. An 8 bit bus mode with MPU, 2 display lines and a 5x8 character font, normal instruction is selected with lcd_cmd(0b00111000).

2. The display on is turned on with lcd_cmd(0b00001100).

3. The display is cleared with lcd_cmd(0b00000001).

The lcdinit function ends with a 2 ms delay implemented by polling.

lcdSetCursor(char row, char col)

The lcdSetCursor() function takes two char arguments: row and column. This function is called in main() so that the user can initially position the cursor at a desired row and column of the LCD (a frame):

As shown by the following table, each frame of the LCD has a unique address:

	Column									
	0	1	2	3	4	5	6	7		
DDRAM Address	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0	Row
	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	1	

Based on the input arguments provided for the row and column, the lcdSetCursor determines the address of the desired frame and provides it as input to the lcd_cmd() function.

The lcdSetCursor begins with the declaration of two integer variables, rowValue and calc. Then an if, else if statement sets the value of the rowValue variable. If the top row of the LCD (row 0) is chosen, rowValue is set to 0x00, otherwise if the bottom row of the LCD is chosen (row 1) the rowValue is set to 0x40. Next an equation to determine the address of the desired frame of the LCD is provided: calc = (rowValue * row) + col. Then the variable calc is masked with 0x80 to set the data bus line 7 = 1 (MSB). The masked calc variable is then provided as input to the lcd_cmd() function.

lcdString(char *s)

This function beings by taking in a parameter of a string. It then starts very similar to our printChar function. It sets the start bit and waits for it to be processed. It then sends the slave address and waits for it to be processed. This is where the variation takes places. In order to process the string, we needed to find the length of the string. Utilizing the string.h C library we used the function strlen to return an int of the amount of characters in our string. This is then used to create an iteration for all letters from s[0] to s[length-1]. During this iteration there is a control bit sent that tells the slave we are going to be sending more commands after this one. After the control bit is sent, there is an acknowledgement wait. Once that condition has been met we then send the char value of the string index i. This again is followed by an acknowledgement. After the letters from s[0] to s[length-1] have been sent we send a new and final control bit. This signals to the slave that this will be our last input from the string. We then send the last letter of the string followed by setting the stop bit. The stop bit is processed, and we exit the function.

_ADC1Interrupt()

First, the interrupt flag is reset (_AD1IF = 0). Then the converted sample for the x and y directions of the joystick are assigned to variables xBuf and yBuf respectively.

getJoystickDirection()

This function determines the direction the joystick is pointing. First, the variable direction is declared. Then sampling begins with the ASAM bit being set to 1. Polling is used to wait for the time to convert the sample from an analog value to a digital value. Next sampling is shut off (ASAM = 0) so that conversion can begin. Subsequently there is a sequence of if / else if statements to determine which direction the joystick is pointing. The joystick uses the 3.3 V supplied by the microcontroller, so the digital values range from 0 to 1024. For each axis, the digital range was split into a first range (left, down) where the digital value is greater than 550, a second range (right, up) where the digital value is less than 400, and a center range where the digital value is a value that was not assigned to the previous directions. Each direction (left, right, up, down, center) are assigned a different number which is returned by the function. Then to reset the direction to center a digital value of 500 is assigned to the xBuf and yBuf variables.

void mazeSetup(void)

This function's main purpose is to randomly choose a maze from a set of 20 premade mazes. Here, each of the 20 premade mazes are made using double arrays, with zeros representing a clear path and one's representing a wall. The two for loops following each maze are used to copy the paths and walls from the premade maze to the maze used during the game. Each individual maze is also assigned a different number from 0 to 19. The rand () % 20 operation is used to randomly select a number from 0 to 19 and assigned to the variable r. Depending upon the value of r, one of the 20 mazes will be made and used for the game. For example, if r=0, the first maze of the list of premade mazes will be used for one iteration of the game. The program ensures that after the player finishes one iteration of the game, the next maze used is not the same maze that the player just finished. This was accomplished with the while (r == prevMaze) loop, which selects another random number from 0 to 19 if the next maze is the same as the previous maze. Finally, the maze[playerRow][playerCol] = 8 was used to light up the starting location of the player at the beginning of the maze. Because, both playerRow and playerCol were initialized to 0 at the beginning of the program, the user will always see a green light on the top left corner of the maze to indicate the starting position.

void updateMaze(int direction)

This function updates the position of the player on the maze. First, it calls checkPlayer(direction), which compares the direction that the user chooses to go with the joystick and decides if the player can move in that direction based upon parameters of the maze such as a wall, out of bounds move, or path. After making that decision, it turns off led of the previous position of the player with the maze[playerPrevCol][playerPrevRow] = 0 command, and sets the new position of the player to green using the maze[playerCol][playerRow] = 8 command.

void writeColor (int r, int g, int b)

This function takes in color parameters of red, green, and blue, then converts them into the respective 8-bit binary values for each color and sends out pulses of ones and zeros. First, the function checks if the parameter values are between 0 and 255 since there is a maximum of 8 bits and so the maximum value for each color can be (2^8)-1 = 255. Once it passes the check, the function takes the MSB of each color, shifts it right, and uses the mod operator to convert it to a corresponding binary value. Depending upon the binary value, the function will call either the write_1() or write_0() assembly subroutines. These subroutines generate a combination of high and low pulses with varying number of cycles to either write a 1 or 0. write_1() uses the bset LATB, #7 along with repeat and nop commands to set RB7 high for 12 cycles and bclr LATB, #7 with nops and return to clear RB7 for 8 cycles. write_0() sets RB7 high for 6 cycles using the bset, repeat, and nop commands and clears RB7 using bclr, repeat, nop, and return for 14 cycles. The count and shift variables were post decremented in order to make sure that we traverse through all 8 bits for each color and shift them to the proper amount of places as a means of converting them to their binary values.

mazeWrite(void)

During this function we call our maze matrix to be checked cell by cell. This is done by two for loops, i indicated the row, and j indicating column. During the for-loop iteration we first check the cell to see what number is inside. Our maze was designed as follows: 1 is a wall, 0 is a path, and 8 is a player. We also set the point 7,7 of the matrix to a pink color to indicate to the player that this is the end/reset point. If the cell being checked is 1 we send the wall color, if 0 we send all low pulses, if 8 we send the player color green. The pulse sending is done in a cascading method utilizing the writeColor function from lab 2b to send the pulses. Prior to this scanning and pulses sending there are 3 variables to indicate what color we want to send, red, green, blue. These are initialized as static at 0. There is a counter called breath that is used to iterate and increment these colors. The first

call of mazeWrite is called in the while(1) thus all variables mentioned are 0. After the cell color send we increment the red, green, blue color dependent on what the breath count is. Breath starting from 0 increments 90 times during 90 calls in main then resets. For breath 0 through 14 we increment the blue color. From 15 through 30 we decrement it. For 31 through 45 we increment red, and 46 through 60 we decrement red. For a breath count of 61 through 75 we increment green, and 76 through 90 we decrement green. Breath count is checked at the end of the function to see if 90 has reached, if so set it back to 0. As you can see each color reaches an "intensity" of 15 then fades back to 0. I say intensity even though it is a color value because anything higher than these colors being wrote to the LED was causing retina burns similar to staring at the sun.

checkPlayer(int direction)

This function begins by setting two global variables playerPrevRow/Col to the current player position. It takes in an int value passed by getjoystickdirection which is then passed into a switch statement. The switch checks the direction int and chooses the case. Case 1 indicates a right movement is trying to be done. This is followed by determining where the player currently is and seeing if that player is at column 7 of the matrix. If so, we know the player can not move right. If this statement is ignored (we aren't at column 7) we move to checking if the movement to the right is a wall. This statement simply checks the maze cell of the player's current position at row, column + 1. If both statements fail, we know we are allowed to move and thus we update the global variable playerCol. Case 2 utilizes the same logic with the exception of that we are checking for the player to move up. If the playerRow is 0 we move to checking if the cell above the player position of playerRow – 1 is a wall, else we can move. Case 3 is again the same except we are moving left. This checks if we are at playerCol 0, if so move to checking if playerCol – 1 in the maze is a wall, else we can move. Case 4 wraps up checking and updating the player movement. We check if we are at row 7, if so move to checking if the row below the player is a wall (playerRow + 1) in the maze, else we move the player. This function was built as a big boy helper which we utilize in updateMaze.

A few things about our Randomization

Pseudorandom number generator is a function of any high-level programming language that is capable of picking near random numbers following a random

generation algorithm created by the creator of the programming language. Calling the rand() function in C will create a pseudo randomly generated number from 0 to 1. The function can however be augmented to produce and create whatever range of random numbers the programmer is interested in. To produce an even more randomized result, you would need to seed the random number generator. Depending on the compiler, this need only be done once, in some cases however, it may be needed multiple times, as in the case of MPLAB. The random generator can be seeded with any value the programmer chooses but it has become common practice to seed the random generator with the system's null time. The value of time when motherboard/cpu was first activated.

Appendix A: Codes

Libraries/Headers:

- xc.h
- p24Fxxxx.h
- stdint.h
- stdlib.h
- lcdlib.
- joysticklib.h
- ws2812lib.h
- bitBangHead.h

Codes of Libraries/Headers:

- bigBangASM.s
- ForeverMaze.c
- Joystick.c
- lcd.c
- ws2812_C_lib.c

LCD

```
#ifndef LCD_H
#defineLCD_H "lcdlib.h"

void lcd_cmd(char c);
void lcdinit(void);
void lcdSetCursor(char row, char col);
void lcdPrintChar(char c);
void lcdString(char *s);

#endif



/*
 * File:   lcd.c
 * Author: crago
 *
 * Created on March 26, 2018, 10:26 AM
 */

#include "xc.h"
#include <p24Fxxxx.h>
#include <stdio.h>
#include <string.h>
#include "lcdlib.h"
```

```
void lcd_cmd(char c)
{

   I2C2CONbits.SEN = 1; //send start signal

   while(I2C2CONbits.SEN); //wait for start to be processed

   I2C2TRN = 0x7C;  //send the address to

   while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2TRN = 0x0; //send control bit

    while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2TRN = c; //send control bit

    while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2CONbits.PEN = 1;
   while(I2C2CONbits.PEN); //wait for stop to be processed
}

void lcdinit(void)
{
```

```
int i;

//50 ms delay
for(i = 0; i < 50; i++){
   while (!IFSobits.T1IF) ;
   IFSobits.T1IF = 0;
}

lcd_cmd(0b00111000); //pass
lcd_cmd(0b00111001); //pass
lcd_cmd(0b00010100); //passes now
lcd_cmd(0b01110000); //contrast c3-c0
lcd_cmd(0b01011110); //c5 - c4 - ion - bon
lcd_cmd(0b01101100); //control

//200ms delay
for(i = 0; i < 200; i++){
   T1CONbits.TON = 1;
   while (!IFSobits.T1IF) ;
   IFSobits.T1IF = 0;
}

lcd_cmd(0b00111000);
lcd_cmd(0b00001100);
lcd_cmd(0b00000001);
```

```
    //2ms delay
    for(i = 0; i < 2; i++)
    {
        T1CONbits.TON = 1;
        while (!IFSobits.T1IF) ;
        IFSobits.T1IF = 0;
    }
}

void lcdSetCursor(char row, char col)
{
    int rowValue, calc;

    //find our first variable of the formula
    if(row == 0)
        rowValue = 0x00;
    else if(row == 1)
        rowValue = 0x40;

    calc = (rowValue * row) + col;

    calc |= 0x80; //set MSB to 1

    lcd_cmd(calc); //send away to the lcd
}
```

```
void lcdPrintChar(char c)
{
   I2C2CONbits.SEN = 1; //send start signal

   while(I2C2CONbits.SEN); //wait for start to be processed

   I2C2TRN = 0x7C;  //send the address to

   while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2TRN = 0x40; //send control bit

   while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2TRN = c; //send c

   while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2CONbits.PEN = 1;

   while(I2C2CONbits.PEN); //wait for stop to be processed
}

void lcdString(char *s)
{
   I2C2CONbits.SEN = 1; //send start signal
```

```
while(I2C2CONbits.SEN); //wait for start to be processed

I2C2TRN = 0x7C;  //send the address to

while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

int i;

int stop = strlen(s);

//send every letter but last in
for(i = 0;  i < stop-1; i++)
{
   I2C2TRN = 0xC0; //send control bit

   while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

   I2C2TRN = s[i];

   while(I2C2STATbits.TRSTAT); //wait for interrupt/ack
}

//send last letter in
I2C2TRN = 0x40; //send control bit
```

```
    while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

    I2C2TRN = s[i];

    while(I2C2STATbits.TRSTAT); //wait for interrupt/ack

    I2C2CONbits.PEN = 1; //stop

    while(I2C2CONbits.PEN); //wait for stop to be processed
}
```

Joystick

```
#ifndef joystick_H
#definejoystick_H "joysticklib.h"

void __attribute__((__interrupt__,__auto_psv__)) _ADC1Interrupt(void);
void joystickSetup(void);
int getJoystickDirection(void);

#endif



#include <p24Fxxxx.h>
#include "xc.h"
#include "lcdlib.h"
#include "ws2812lib.h"



//x will be an0, assign ADC1BUF 0
//y will be an1, assign ADC1BUF 1
volatile int xBuf, yBuf;

//add buffer to variable for checking
void __attribute__((__interrupt__,__auto_psv__)) _ADC1Interrupt(void)
{
```

```
    _AD1IF = 0;
    xBuf = ADC1BUF0;
    yBuf = ADC1BUF1;
}

//gather samples, pass direction via if-else-if, set xyBuf back to center
int getJoystickDirection(void)
{
    //left = 1, up = 2, right = 3, down = 4, center = 5
    int direction;

    //auto sample start - TAD based conversion
    AD1CON1bits.ASAM = 1; //start sample
    while(!IFS0bits.AD1IF){}; //wait until conversion time
    AD1CON1bits.ASAM = 0; //shut off sampling to start conversion

    if(xBuf > 550) //x-right
        direction = 1;
    else if(xBuf < 400) //x-left
        direction = 3;
    else if(yBuf > 550) //y-down
        direction = 4;
    else if(yBuf < 400) //y-up
        direction = 2;
    else
        direction = 5;
```

```
    //recenter x and y Bufs
    xBuf = 500;
    yBuf = 500;

    return direction;
}

void joystickSetup(void)
{
    TRISA = 0b11;

    //Setting up ADC
    AD1CON1 = 0;
    AD1CON2 = 0;
    AD1CON3 = 0;

    //Set ANx to scan
    AD1CON2bits.CSCNA = 1; //enable scanning
    AD1CON2bits.SMPI = 1;  //take 2 sample before interrupt
    AD1CON2bits.ALTS = 0; //set to use for MUX A
    AD1CON1bits.SSRC = 0b111;

    AD1CSSLbits.CSSL0 = 1; //AN0
    AD1CSSLbits.CSSL1 = 1; //AN1
```

```
    //Setup for TAD
    AD1CON3bits.ADCS = 1; //TAD = 2TCy = 125ns
    AD1CON3bits.SAMC = 17; //sample time =

    //set interrupt flags
    _AD1IE = 1; //enable ad int
    _AD1IF = 0; //clear AD IF int

    _ADON = 1;

     //i2c setup
    I2C2CONbits.I2CEN = 0; //disable
    I2C2BRG = 0x9D; //100 kHz
    I2C2CONbits.I2CEN = 1; //enable
    _MI2C2IF = 0;

    //TIMER SETUP
    T1CON = 0;
    PR1 = 15999;
    TMR1 = 0;
    IFSobits.T1IF = 0;
    T1CONbits.TON = 1;
    lcdinit();
    lcdSetCursor(0,0);
    lcdString("test");
}
```

BitBangHead

```
#ifndef bitBang_H

#definebitBang_H "bitBangHead.h"


void write_0(void);

void write_1(void);

void oneMilliSec(void);


#endif



.include "xc.inc"

.text                   ;BP (put the following data in ROM(program memory))

 ; This is a library, thus it can *not* contain a _main function: the C file will

 ; define main().  However, we ; we will need a .global statement to make available ASM functions to C code.

 ; All functions utilized outside of this file will need to have a leading

 ; underscore (_) and be included in a comment delimited list below.

.global _write_1, _write_0, _oneMilliSec



;1 code total: 62.5 ns * 20 cycles = 1.25 us

;T1H: 62.5 ns * 12 cycles = 75.0 us

;20 cycles - 12 high 8 low

_write_1:
```

```
    ;call 2
    bset LATB, #7           ;high 1
    repeat #9       ;high 1
    nop             ;high 10
    bclr LATB, #7           ;low 1
    nop             ;low 1
    nop             ;low 1
    return          ;low 3 + 2 for call

;0 code total: 62.5 ns * 20 cycles = 1.25 us
;ToH: 62.5 ns * 6 cycles = .375 us
_write_0:
    ;call 2
    bset LATB, #7           ;high 1
    repeat #3       ;1
    nop             ;4
    bclr LATB, #7           ;low 1
    repeat #6       ;low 1
    nop             ;low 7
    return          ;low 3 + 2 for call

_oneMilliSec:
                    ;if needed reduce 2 cycle on repeat for bra
                    ;2 cycles call
    repeat #15993   ;1 cycle
    nop             ;15996 cycle
```

```
return              ;3 cycle
.end
```

WS2812

```
#ifndef ws2812_H
#definews2812_H "ws2812lib.h"

void writeColor(int r, int g, int b);
void drawFrame(unsigned char frame);
unsigned long int packColor(unsigned char r, unsigned char g, unsigned char b);
unsigned char getR(unsigned long int rgb);
unsigned char getG(unsigned long int rgb);
unsigned char getB(unsigned long int rgb);
void writePacked(unsigned long int packed);
unsigned long int wheel(unsigned char pos);
void delay(int delay);

#endif



#include "bitBangHead.h"
//bit shifts right 1 bit, takes remainder of that which is our binary bit starting
//from the most significant position, checks this bit and writes in the order of
//rbg colors from R --> G --> B
void writeColor(int r, int g, int b)
{
   if(r >= 0 && r <= 255 && g >= 0 && g <= 255 && b >= 0 && b <= 255)
```

```
{
    int shift = 7;
    int count = 8;
    while(count > 0)
    {
        if( (r >> shift) % 2 == 1)
            write_1();
        else
            write_0();
        count--;
        shift--;
    }

    shift = 7;
    count = 8;
    while(count > 0)
    {
        if( (g >> shift) % 2 == 1)
            write_1();
        else
            write_0();
        count--;
        shift--;
    }

    shift = 7;
```

```
        count = 8;
        while(count > 0)
        {
            if( (b >> shift) % 2 == 1)
                write_1();
            else
                write_0();
            count--;
            shift--;
        }

        //oneMilliSec();
    }
}

void drawFrame(unsigned char frame)
{
    int byteRed = frame;
    int byteBlue = 255 - frame;
    writeColor(byteRed, 0, byteBlue);
}

unsigned long int packColor(unsigned char r, unsigned char g, unsigned char b)
{
    return (((long)r << 16) | ((long)g << 8) | ((long)b));
}
```

```
unsigned char getR(unsigned long int rgb)
{
   return ((unsigned char) (rgb >> 16));
}

unsigned char getG(unsigned long int rgb)
{
   return ((unsigned char) (rgb >> 8));
}

unsigned char getB(unsigned long int rgb)
{
   return ((unsigned char) (rgb >> 0));
}

void writePacked(unsigned long int packed)
{
   writeColor(getR(packed), getG(packed), getB(packed));
}

unsigned long int wheel(unsigned char pos)
{
   pos = 255 - pos;
   if(pos < 85)
   {
```

```
        return (packColor(255 - pos * 3, 0, pos * 3));
    }
    if(pos < 170)
    {
        pos -=85;
        return (packColor(0, pos * 3, 255 - pos * 3));
    }
    pos -= 170;
    return (packColor(pos * 3, 255 - pos * 3, 0));
}

void delay(int delay)
{
    while(delay > 0)
    {
        oneMilliSec();
        delay--;
    }
}
```

Main

```
#include "xc.h"

#include <p24Fxxxx.h>

#include <stdint.h>

#include <stdlib.h>

#include "lcdlib.h"

#include "joysticklib.h"

#include "ws2812lib.h"



// CW1: FLASH CONFIGURATION WORD 1 (see PIC24 Family Reference Manual 24.1)

#pragma config ICS = PGx1 // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared with PGC1/PGD1)

#pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog Timer is disabled)

#pragma config GWRP = OFF // General Code Segment Write Protect (Writes to program memory are allowed)

#pragma config GCP = OFF // General Code Segment Code Protect (Code protection is disabled)

#pragma config JTAGEN = OFF // JTAG Port Enable (JTAG port is disabled)



// CW2: FLASH CONFIGURATION WORD 2 (see PIC24 Family Reference Manual 24.1)

#pragma config I2C1SEL = PRI // I2C1 Pin Location Select (Use default SCL1/SDA1 pins)

#pragma config IOL1WAY = OFF // IOLOCK Protection (IOLOCK may be changed via unlocking seq)

#pragma config OSCIOFNC = ON // Primary Oscillator I/O Function (CLKO/RC15 functions as I/O pin)
```

```
#pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled,



// Fail-Safe Clock Monitor is enabled)

#pragma config FNOSC = FRCPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))


#define PLAYERSPEED 60 //sets framerate of movement / led pulse sending. In ms.


volatile static int playerPrevRow = 0, playerPrevCol = 0, playerRow = 0, playerCol = 0;

volatile static int maze[8][8];


void setup(void)

{

   CLKDIVbits.RCDIV = 0;

   AD1PCFG = 0xFFFC; //set up AN0, AN1; Other pins digA

   TRISB = 0x0011;     //RA0 output

   joystickSetup();

}


void mazeSetup(void)

{

   int i,j;

   static int prevMaze;

   int r = rand() % 20; //generates random number b/w 0-19
```

```
//maze sure we dont get 2 of the same maze in a row
while(r == prevMaze)
{
   r = rand() % 20;
}

prevMaze = r;

if(r == 0)
{
   volatile int premade[8][8] = {
         {0,1,1,1,1,1,1,1},
                                    {0,1,1,1,1,1,1,1},
                                    {0,0,0,0,0,0,1,1},
                                    {1,1,1,1,1,0,0,0},
                                    {1,1,1,1,1,1,1,0},
                                    {1,1,1,1,1,1,1,0},
                                    {1,1,1,1,1,1,1,0},
                                    {1,1,1,1,1,1,1,0}
   };

   for(i = 0; i < 8; i++)
   {
      for(j = 0; j < 8; j++)
      {
```

```
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 1)
{
    volatile int premade[8][8] = {
            {0,0,0,0,0,0,0,0},
                                    {1,1,0,1,1,1,1,0},
                                    {1,1,0,1,1,1,1,0},
                                    {1,1,0,1,1,1,1,0},
                                    {1,1,0,1,1,1,1,0},
                                    {1,1,0,1,1,1,1,0},
                                    {1,1,0,0,0,0,1,0},
                                    {1,1,1,1,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}
```

```
else if(r == 2)
{
    volatile int premade[8][8] = {
            {0,1,1,1,1,1,1,0},
                                        {0,1,1,1,1,1,1,1},
                                        {0,1,1,1,1,1,1,1},
                                        {0,0,0,0,0,0,0,0},
                                        {1,1,1,0,1,1,1,0},
                                        {0,0,0,0,1,1,1,0},
                                        {0,1,0,0,1,1,1,0},
                                        {0,0,0,0,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 3)
{
    volatile int premade[8][8] = {
```

```
        {0,1,1,1,1,1,1,1},
                                {0,0,1,1,1,1,1,1},
                                {1,0,1,1,1,1,1,1},
                                {1,0,0,0,0,0,0,0},
                                {0,1,1,1,1,1,1,0},
                                {0,1,1,1,1,1,1,0},
                                {0,1,1,1,1,1,1,0},
                                {0,1,1,1,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 4)
{
    volatile int premade[8][8] = {
        {0,0,1,0,1,0,1,0},
                                {0,0,1,0,1,0,0,0},
                                {0,0,1,0,1,0,1,0},
                                {0,0,1,0,1,0,0,0},
```

```
                                    {0,0,0,0,0,0,1,0},
                                    {1,0,1,1,1,0,0,0},
                                    {1,0,1,1,1,0,0,0},
                                    {1,0,1,1,1,0,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 5)
{
    volatile int premade[8][8] = {
            {0,0,0,0,1,1,1,1},
                                    {0,1,1,1,0,1,1,1},
                                    {0,0,0,0,0,0,1,1},
                                    {0,0,0,1,0,1,0,1},
                                    {0,0,0,0,0,0,1,0},
                                    {0,1,1,1,0,1,0,1},
                                    {0,0,0,0,0,0,1,1},
                                    {1,1,0,0,0,0,0,0}
```

```
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 6)
{
    volatile int premade[8][8] = {
            {0,0,0,0,0,0,1,1},
            {1,1,0,1,1,0,0,0},
            {0,0,0,1,1,0,1,1},
            {0,1,1,1,0,0,0,1},
            {0,0,1,1,0,1,0,0},
            {1,0,0,1,0,1,0,1},
            {1,1,0,1,0,1,0,1},
            {1,1,0,0,0,1,0,0}
    };

    for(i = 0; i < 8; i++)
    {
```

```
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 7)
{
    volatile int premade[8][8] = {
            {0,1,0,0,0,1,0,1},
            {0,1,0,1,0,0,0,1},
            {0,0,0,1,0,1,0,1},
            {1,1,1,1,0,1,0,0},
            {0,0,0,0,0,1,0,1},
            {0,1,1,1,1,1,1,1},
            {0,1,1,0,0,0,1,1},
            {0,0,0,0,1,0,0,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
```

```
    }
}

else if(r == 8)
{
    volatile int premade[8][8] = {
            {0,0,1,1,0,0,0,0},
            {1,0,0,1,1,0,1,0},
            {1,0,1,1,0,0,1,0},
            {0,0,1,0,0,1,1,0},
            {0,1,1,0,1,1,0,0},
            {0,1,0,0,1,0,0,1},
            {0,1,1,0,1,1,0,1},
            {0,0,0,0,1,1,0,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 9)
```

```
{
    volatile int premade[8][8] = {
            {0,1,1,0,0,0,0,0},
            {0,1,1,0,1,0,1,0},
            {0,0,0,0,1,0,1,0},
            {1,1,1,1,1,0,1,0},
            {1,0,0,0,0,0,1,1},
            {1,0,1,1,1,1,0,0},
            {0,0,1,0,0,0,0,0},
            {1,0,0,0,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 10)
{
    volatile int premade[8][8] = {
            {0,1,1,0,0,0,0,0},
            {0,1,1,0,1,0,1,0},
```

```
            {0,0,0,0,1,0,1,0},
            {1,1,1,1,1,0,1,0},
            {1,0,0,0,0,0,1,1},
            {1,0,1,1,1,1,0,0},
            {0,0,1,0,0,0,0,0},
            {1,0,0,0,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 11)
{
    volatile int premade[8][8] = {
            {0,1,1,0,0,0,1,0},
            {0,0,1,0,1,0,1,0},
            {0,1,1,0,1,0,0,0},
            {0,1,0,0,1,1,1,0},
            {0,1,0,1,1,0,0,0},
            {0,1,0,1,0,0,1,1},
```

```
        {0,1,0,1,0,1,1,1},
        {0,0,0,1,0,0,0,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 12)
{
    volatile int premade[8][8] = {
        {0,0,0,0,0,0,1,1},
        {1,0,0,1,0,0,0,0},
        {1,0,0,1,1,1,1,1},
        {0,0,1,1,0,0,0,0},
        {0,0,0,0,1,0,1,0},
        {0,1,0,1,0,1,1,0},
        {1,1,0,1,0,0,0,1},
        {0,1,0,0,0,1,0,0}
    };
```

```
    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 13)
{
    volatile int premade[8][8] = {
            {0,1,1,1,1,1,1,1},
            {0,0,0,1,0,0,0,0},
            {1,0,0,1,0,1,1,0},
            {1,0,1,1,0,1,0,0},
            {1,0,1,1,0,1,0,1},
            {1,0,1,0,0,1,0,0},
            {1,0,0,0,0,0,1,0},
            {1,1,1,1,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
```

```
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 14)
{
    volatile int premade[8][8] = {
            {0,0,0,1,1,0,0,1},
            {1,1,0,0,0,0,1,0},
            {0,1,0,0,1,1,1,1},
            {1,0,1,0,0,0,0,0},
            {1,0,0,0,1,1,0,1},
            {1,0,0,0,0,1,0,0},
            {1,1,1,0,0,0,1,0},
            {0,1,0,0,1,1,0,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}
```

```
else if(r == 15)
{
    volatile int premade[8][8] = {
            {0,1,1,0,0,0,0,1},
            {0,0,0,0,1,1,0,1},
            {1,1,0,0,1,0,0,0},
            {0,1,1,0,0,1,1,0},
            {0,0,0,1,0,0,1,0},
            {0,1,0,0,0,1,0,0},
            {0,1,0,1,0,1,0,1},
            {0,0,0,1,1,1,0,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 16)
{
    volatile int premade[8][8] = {
```

```
            {0,0,0,0,1,1,1,0},
            {1,0,1,0,0,1,1,1},
            {0,1,1,1,0,0,0,0},
            {0,0,1,0,0,0,1,0},
            {1,0,1,0,1,0,1,0},
            {0,1,1,0,0,1,0,0},
            {0,1,1,1,0,1,0,1},
            {0,0,0,0,1,1,0,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 17)
{
    volatile int premade[8][8] = {
            {0,1,1,1,1,1,1,1},
            {0,0,0,0,1,1,1,1},
            {1,1,0,0,0,0,0,1},
            {0,1,0,1,1,1,0,1},
```

```
            {0,0,0,1,1,1,0,1},
            {0,0,1,1,1,1,0,1},
            {1,0,1,1,1,1,0,0},
            {1,0,1,1,1,1,1,0}
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 18)
{
    volatile int premade[8][8] = {
            {0,0,0,0,0,0,0,0},
            {1,1,1,1,1,1,1,0},
            {0,0,0,0,0,0,0,0},
            {0,1,1,1,1,1,1,1},
            {0,0,0,0,0,0,0,0},
            {1,1,1,1,1,1,1,0},
            {1,1,1,1,1,1,1,0},
            {1,1,1,1,1,1,1,0}
```

```
    };

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            maze[i][j] = premade[i][j];
        }
    }
}

else if(r == 19)
{
    volatile int premade[8][8] = {
            {0,1,0,0,0,0,1,1},
            {0,1,0,1,1,0,1,1},
            {0,1,0,1,1,0,1,1},
            {0,1,0,1,1,0,0,0},
            {0,1,0,1,1,1,1,0},
            {0,1,0,0,0,0,1,0},
            {0,1,1,1,1,0,1,0},
            {0,0,0,0,0,0,1,0}
    };

    for(i = 0; i < 8; i++)
    {
```

```
            for(j = 0; j < 8; j++)
            {
                maze[i][j] = premade[i][j];
            }
        }
    }

    maze[playerRow][playerCol] = 8;
}

//this scans the double array and as it is scanning it sends the correct color
//for the intended item in the array. Breath effect for wall, green for player,
//pink for end zone, and no color for path.

//the breath effect moves from blue, green, red, then resets
void writeMaze(void)
{
    int i, j;

    static int red = 0, blue = 0, green = 0, breath = 0;

    for(i = 0; i < 8; i++)
    {
        for(j = 0; j < 8; j++)
        {
            if(maze[i][j] == 1) //walls
```

```
            writeColor(green, red, blue);
        else if(maze[i][j] == 8) //this is the player
            writeColor(22,0,0);
        else if(i == 7 && j == 7) //end zone
            writeColor(0,22,22);
        else
            writeColor(0, 0, 0);//path
    }
}

if(breath <= 30) //blue in and out
{
    if(breath > 15)
        blue--;
    else
        blue++;
}
else if(breath <= 60 && breath > 30)
{
    if(breath > 45)
        red--;
    else
        red++;
}
else if(breath > 60 && breath <= 90)
{
```

```
        if(breath > 75)
            green--;
        else
            green++;
    }

    if(breath == 90)
        breath = 0;

    breath++;

    delay(1); //set hold/reset for maze
}

//checks the player value against the direction sent by the joystick sampling
//has bound checking for walls and out of bounds
void checkPlayer(int direction)
{
    //left = 3, up = 2, right = 1, down = 4, center = 5
    playerPrevRow = playerRow; //pass position to previous
    playerPrevCol = playerCol;

    switch(direction)
    {
        case 1://right
            if(playerCol == 7)//out of bounds
```

```
        break;
    else if(maze[playerRow][playerCol + 1] == 1) //wall
        break;
    else
        playerCol++; //move right
    break;

case 2://up
if(playerRow == 0)//out of bounds move
    break;
else if(maze[playerRow - 1][playerCol] == 1)//wall
    break;
else
    playerRow--; //move up
return;

case 3://left
    if(playerCol == 0)//out of bounds move
        break;
    else if(maze[playerRow][playerCol - 1] == 1)//wall
        break;
    else
        playerCol--; //move left
    break;

case 4://down
```

```
            if(playerRow == 7)
                break;
            else if(maze[playerRow + 1][playerCol] == 1)
                break;
            else
                playerRow++; //move down
            break;

        default: //5 - center
            break;
    }
}

//updates the player position
//it then clears the last position and updates the new one
void updateMaze(int direction)
{
    checkPlayer(direction);
    maze[playerPrevRow][playerPrevCol] = 0;
    maze[playerRow][playerCol] = 8;
}

int main(void) {
    setup();
    static int playerDirection;
```

```
while(1)
{
    //new maze has started, place player at start
    playerRow = 0; //start row
    playerCol = 0; //start col
    srand(time(NULL)); //change seed every pass
    mazeSetup();

    //stay in current maze until player has solved it
    //7,7 is the maze end
    while(maze[7][7] != 8)
    {
        playerDirection = getJoystickDirection();
        updateMaze(playerDirection);
        writeMaze();

        //used mainly for debugging
        if(playerDirection == 3) //x-left
            lcdString("left");
        else if(playerDirection == 1) //x-right
            lcdString("right");
        else if(playerDirection == 4) //y-down
            lcdString("down");
        else if(playerDirection == 2) //y-up
            lcdString("up");
        else
```

```
            lcdString("no move");


        delay(PLAYERSPEED);


        lcd_cmd(1); //clear display
      }


  }
  return 0;
}
```