E155 Final Project Report - Music Rhythm Game: 8 Bit Hero

Jimmy Fernandez, Kathryn Chan, Anneka Noë

November 30, 2020

I. PROJECT INTRODUCTION

The goal of this project is to create a music rhythm game, in a similar style to *Guitar Hero*. Upon initialization, the user will be able to select between several predefined songs. During the game, an LED matrix will display scrolling information on what buttons the user should press at a certain time. If the correct buttons are pressed, the system will play the correct notes of the predefined song; if incorrect buttons or no buttons are pressed, the system will not play sound. After the game ends, the LED matrix will display the game score corresponding to the accuracy with which the user pressed the buttons.

The block diagram of the overall system is shown in Figure 1. The user inputs are fed to the input control which manipulates the inputs for ease of use in the game controller. The game controller uses these inputs in addition to a predetermined array of song data to feed appropriate values to the speaker and LED modules based on the user inputs. Figure 2 displays how the major pieces of hardware are electrically connected.

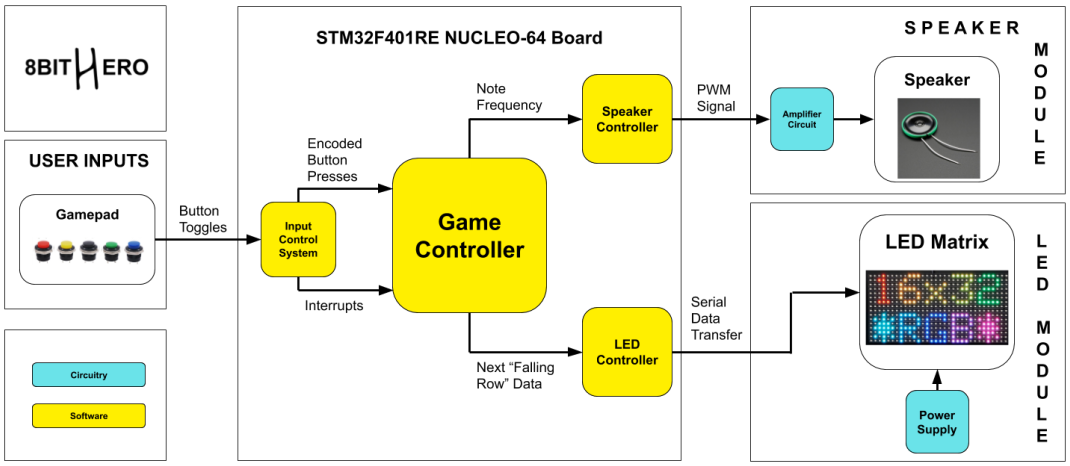

Figure 1. Block diagram of overall system

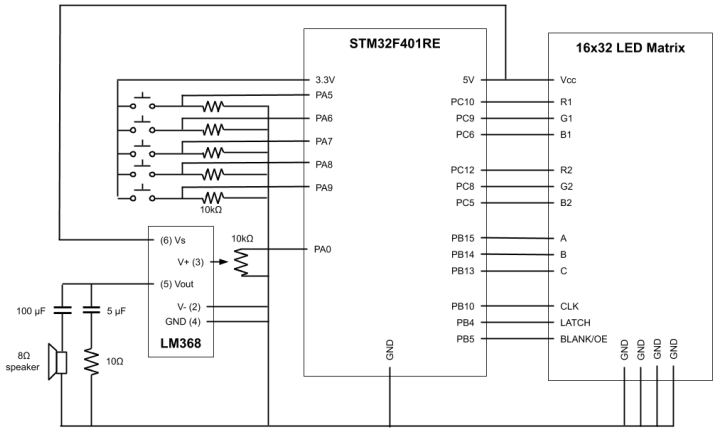

Figure 2. Circuit schematic for overall system

II. DELIVERABLES

The deliverables of this project include the following items:

- ❏ LED Matrix Display that indicates when to press specified buttons to play notes.
- ❏ A speaker setup that will play the note specified by the buttons pressed by the user.
- ❏ Maximum latency of 50 ms between button press and playing of a note.
- ❏ After a song is played, there will be an indicator of accuracy of user inputs displayed on the LED Matrix.
- ❏ At least three songs that the user can select from (these options will be designated 1, 2, and 3).

The team met all deliverables specified above. The LED Matrix Display and the speaker setup are discussed in *III.D* and *III.C*, respectively. The calculation of user accuracy score and song selection process are discussed in *III.B Game Control Subsystem*. The button-to-note latency and other deliverables are discussed in *Section V. Results*.

III. TECHNICAL DETAILS: CONTROLLER SUBSYSTEMS

A. User Input Subsystem

The user input subsystem is intended to generate interrupts when any of the user pushbuttons are toggled and to provide a five bit integer to the game controller to indicate which of the five buttons are pressed. Figure 3 displays the schematic for the User Input Subsystem. Each pushbutton is powered at one node and connected in series with a pull down resistor on the other node. The value read by the board, the node between the pushbutton and resistor, reads high when a button is pressed, and low when a button is released.

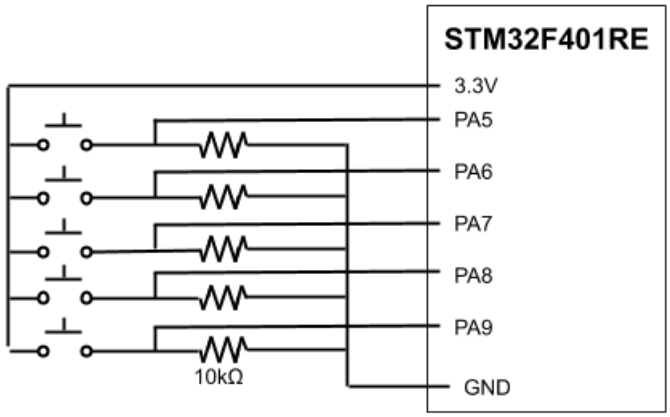

Figure 3. Schematic for User Input and Speaker Subsystems

Each of the GPIOs pins connected to the pushbuttons is configured to trigger interrupts on both rising and falling edges of button inputs which are handled by the EXTI9_5 interrupt handler. When an interrupt event occurs, the handler checks which buttons are pressed and manipulates the PWM signal sent to the speaker module appropriately. Inputs could result in changing the game screen, selecting a song level, or playing a note during gameplay. The details of these outputs are discussed in *III.B Game Control Subsystem* and *III.D LED Matrix Subsystem.*

In order to ensure that the inputs of the buttons were read properly, it was important to characterize the bouncing of the pushbutton press signal. Figure 4 shows an oscilloscope trace of the signal produced when the button is pressed and released quickly. It can be seen that the signal has no significant bouncing, which is why no debouncing algorithms have been implemented in hardware or software. However, if the project were to switch to different buttons that have significant bouncing, an RC or SR debouncing circuit could be used to debounce in hardware, or minor delays could be introduced to skip over the bouncing period before the value of the button is read in firmware [1].

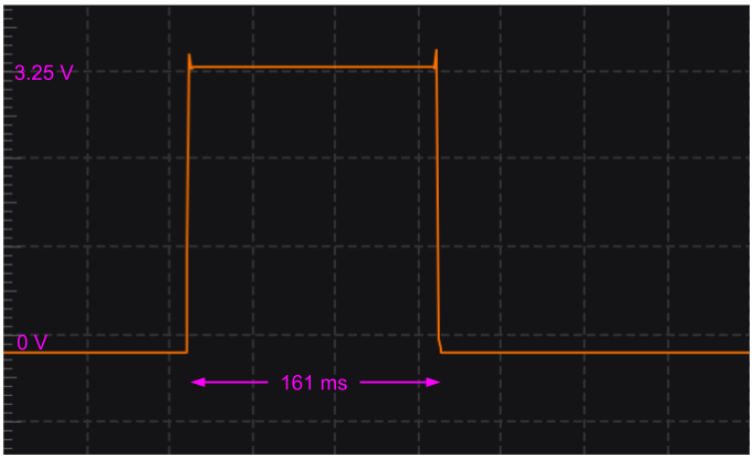

Figure 4. Oscilloscope trace of output when pushbutton is quickly pressed and released

B. Game Control Subsystem

The game controller decides what button information to display on the LED matrix and what notes to play based on user inputs. The controller is also responsible for synchronizing timing related to the scrolling rate of the board and the duration of each note and button press. Figure 5 shows a block diagram of the game control subsystem. At each scrolling refresh (i.e. each time the display screen scrolls), the game controller must index into the predefined song array in two different locations. The first index corresponds to the new row displayed at the top of the LED matrix. The second index corresponds to the buttons and notes that should be pressed and played at the current moment in time. The suffix "_0" on a variable indicates that a variable is related to the newly displaced LED row, while the suffix "_1" indicates that a variable is related to the correct current-moment button presses.

With each scrolling refresh, both indices are updated based on the duration of the new LED row buttons and the current-moment expected buttons. The function then updates the current-moment expected buttons (buttons_1), as well as a global variable for the next buttons to display on the LED screen (buttons_0), which the LED controller accesses and maps to a new LED row. The current-moment note (pitch_1) is also updated. If the current-moment index (index_1) has been updated, the game controller checks what buttons are pressed and changes the frequency of the note being played across the speaker accordingly using the checkButtonsUpdatePWM function. This ensures that the PWM is updated if a new note is supposed to be played.

The checkButtonsUpdatePWM function first checks whether the buttons that the user is currently pressing matches the expected buttons from the song array. This function needs to be executed whenever the user buttons change, which is handled by an interrupt discussed in *III.A User Input Subsystem*. This function also needs to be executed if the expected buttons change (buttons_1). The checkButtonsUpdatePWM function then determines what pitch to send to the speaker controller. If the correct buttons are pressed, the correct pitch (pitch_1) is played. If no buttons are pressed or the wrong buttons are pressed, then no note is played. This is achieved by calling the updatePWM function which updates the PWM with the determined pitch (either with 'pitch_1' or with a value of 0).

Two variables are used to calculate the user accuracy score: 'points' and 'possible_points' (where accuracy_score is the ratio of points to possible points in percentage form). To give the user some leeway, the user gets credit if the correct buttons are pushed at any point during the current LED row. For example, if the scrolling refresh rate of the LED matrix is 4 Hz, then the user must press the correct buttons that correspond to the lowest LED row anytime within the 0.25 seconds that this row is being displayed at the bottom of the LED matrix. To achieve this, the variable 'possible_points' is incremented every time the LED matrix scrolls if a note is supposed to be playing. If a note is not supposed to be playing, the user is neither awarded nor penalized any points. The variable 'points' is incremented if the following are true: the current user input buttons match the expected buttons, 'points' has not yet been incremented for this LED row, and a note is supposed to be playing.

Once the end of the song array is reached (indicated by a duration of 0 in the song array), the global ‘accuracy_score’ variable is calculated and updated, and can then be accessed by the LED controller. The variable ‘flag_end_song’ is also set to 1 to indicate that the song has ended.

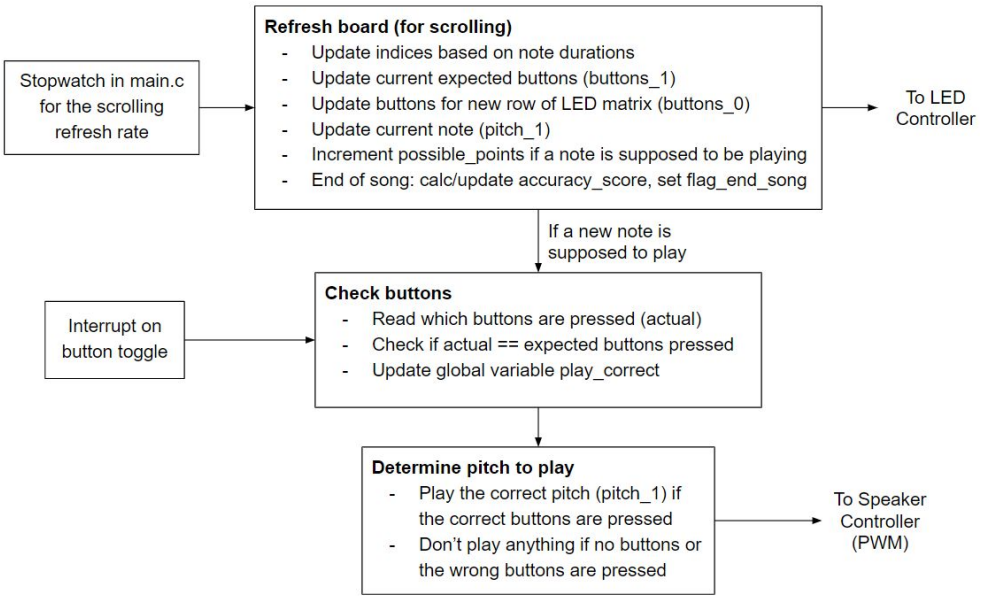

Figure 5. Block diagram of game controller subsystem

C. Speaker Subsystem

After the game controller has configured the note timer to output a signal with a certain frequency, the signal is sent through the amplifier circuit displayed in Figure 6. The circuit is designed to amplify the power of the original PWM signal so that current through and voltage across the speaker are substantial enough to create satisfactory sound. This circuit is the minimum part count application for the LM386 Low Voltage Audio Power Amplifier [2]. The potentiometer present in the circuit is used to regulate the volume of the sound outputted by the speaker. The configuration and values of capacitors and resistors in the circuit were recommended by the datasheet, though the values in Figure 6 are slightly different due to limited availability of components.

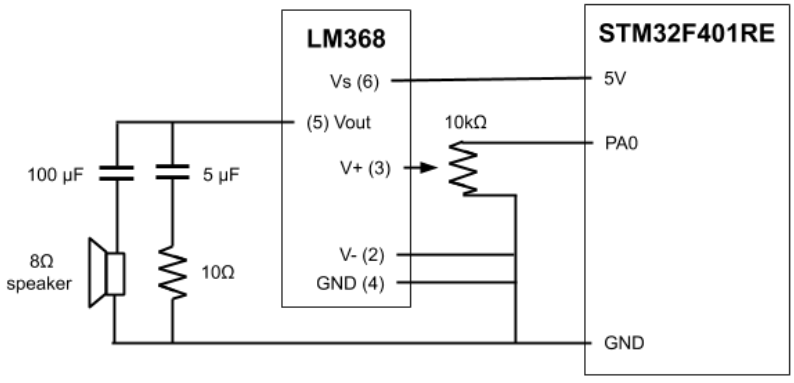

Figure 6. Speaker Electrical Schematic

D. LED Matrix Subsystem

The team purchased a 16x32 RGB LED Matrix from Adafruit for use as the game display of *8 Bit Hero*. LED matrices of this type do not have available datasheets, so interfacing with it required researching the internet and past E155 Final Project reports where similar matrices were used [1, 2, 3, 4]. The process of displaying

information to the board as the team understands it is described here in detail. The team hopes this will be helpful to future E155 students. To help with this goal, information about both the 16x32 and 32x32 Adafruit matrix will be discussed here, as there are only minor differences between the two.

Powering the matrix requires a large amount of current. The team followed Adafruit's advice and used the 5V, 2A power supply linked on the product page. In addition to this, another connector is needed for the power terminal on the back of the LED matrix. However, as described in the Adafruit tutorial, the terminal on the back can take two different forms, and the connectors used to interface with them are different [7]. For the molex style connector, a 2.1x5.1mm female DC power supply to screw adapter is needed (see parts list), while the screw post connector requires a 2.1mm male/female extension cord to be cut, stripped, and soldered on. Since there is no guarantee which connection a board will have, it is recommended that the user either waits to verify which connection they have, or buy both ahead of time. As an added benefit, if both are purchased and the connection type is the molex connector, the extension cord can still be used to extend the range of the power supply. The team recommends following the Adafruit tutorial for hooking up the power connections [7]. Also note that due to built on 74HC245 octal bus transceiver chips, the inputs are buffered, so 3.3V based logic can be safely used to drive the board despite its 5V power supply [8].

For both the 16x32 and 32x32 matrix, only two rows of the LED matrix (offset from one another by half of the number of rows on the matrix) can be displayed at a time. Therefore, in order to display an image, the display must be "scanned" repeatedly, which means that every row must be flashed in a sequence quickly enough for the image to seem static to the human eye. For displays like this, this is known as having a 1/8th duty cycle (1/16th for the 32x32 matrix).

Each LED on the matrix is actually composed of three smaller LEDs: one red, one blue, and one green. These LEDs can only be turned on and off, so to achieve a wider color range (i.e. more bits of color for red, green, and blue) a different method of driving the matrix would have to be used (see [6] for discussion of "Binary Coded Modulation"). The driving sequence described here only uses 8 colors: black, blue, green, cyan, red, purple, yellow, and white. However, this limited range of colors was substantial for the team's purposes.

To drive an LED on the board, the row needs to be selected and the LED needs to be driven by the column driver. Both the 16x32 and 32x32 variants have a total of six column LED drivers, broken into two groups of red, green, and blue drivers. Each driver consists of a 32 bit shift register - where each nth bit indicates whether or not the colored LED in the nth column is on or off - and a parallel output register. The inputs R1, G1, B1 are connected to the red, green, and blue drivers for the upper rows of the matrix. Inputs R2, G2, and B2 are the remaining three drivers, but are similarly connected to the lower rows of the matrix. A clock signal SCLK is used to indicate when data is ready to be shifted in. The parallel output driver has a latch signal LATCH, which moves the data from the shift register into the output register when enabled. The drivers also have a blank signal BLANK, which turns off all outputs when enabled. The SCLK, LATCH, and BLANK signals are shared on all drivers. A diagram of how a column driver is (probably) implemented in hardware is shown in Figure 7., found from Glen Akin's tutorial [6].

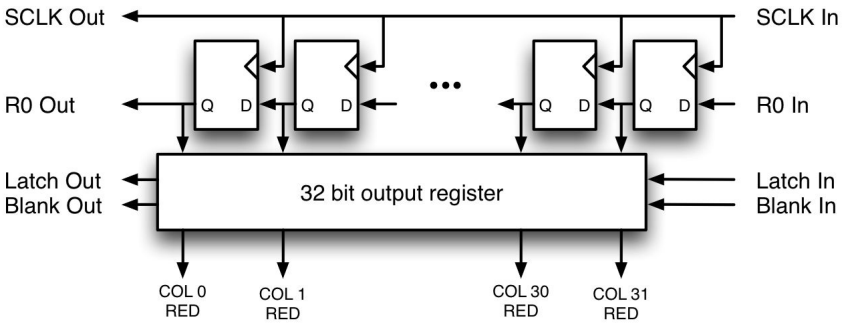

Figure 7. Red, Green, or Blue LED Column Driver [6]

To select which rows will be displayed, the matrix uses demultiplexers. In the case of the 16x32 matrix, a 3 bit address (pins A, B, and C) is used for a 3 to 8 demultiplexer. For the 32x32 matrix, a 4 bit address (pins A, B, C,

and D) is for a 4 to 16 demultiplexer. For sake of simplicity, the address signal will be referred to as ADDR from here on, where the A pin is the LSB of the address and C (or D for the 32x32 matrix) is the MSB of the address. Two demultiplexers are used for both matrices, one for the upper rows and one for the lower rows, and they share the same address inputs. This means that for a set value of ADDR, the two rows that will be displayed are ADDR and ADDR + ((Number of rows) / 2), with ADDR receiving color data from R1, G1, B1 and ADDR + ((Number of rows) / 2) receiving data from R2, G2, and B2. As a note, the ADDR is indexed from zero.

With all this in mind, it is easier to understand the process of displaying data to the matrix. As a rough overview, six groups of 32 bits of color data need to be serially written to the matrix's shift drivers synchronously (as they share the same clock). Once the data is written to the registers, the display needs to be blanked, the row needs to be selected, and then the color data can be latched into the parallel output driver. This process is then repeated for each group of rows fast enough to seem like a static image to the human eye. The specific sequence of data transmission and control signals required to display data to the matrix is described in further detail in the following list:

1. Shift the six groups of 32 bit color data into the RGB shift registers (R1, G1, and B1 for ADDR; R2, G2, B2 for ADDR + ((Number of rows) / 2)). Recall that data transfer is synchronized since the shift registers share the same clock SCLK.
    a. Update data values on the falling edge of SCLK (or when the clock is low, in our case) to ensure signal stability [3]
    b. Values are shifted into the shift register when the clock goes high
    c. NOTE: It does not matter what address is set at this point, that will be set later
    d. NOTE: Once all the data is transferred into the shift registers, do not toggle SCLK any more.
2. Assert BLANK, which turns off every LED on the screen.
3. Set the value of ADDR to be the address that is to be displayed during this cycle
    a. Current ADDR = (Previous ADDR + 1) % ((Number of Rows) / 2))
4. Assert and then deassert LATCH. This passes color data into the parallel output drivers on the display.
5. Delay for an arbitrary period of time (to reduce brightness and prevent bleeding), then deassert BLANK
    a. The team found that a delay of 1.25 ms prevented color bleed and reduced the brightness to a manageable level. Other delay values may work for different desired outputs
6. Repeat from step 1 for the next pair of rows in the sequence. Repeat for all rows on the board fast enough for the image to appear static (Glen Akins recommends ~100-200 times a second to prevent flickering [6]).

Due to the parallel transmission of the color data, driving one of these matrices is better suited to an FPGA. However, the team was able to use this sequence to display images on the LED matrix using the STM32F401RE Nucleo-64 board running at 84MHz by bit-banging the data out on GPIO data pins. However, driving the matrix on a microcontroller in this manner does require a large proportion of the processor's time, as the display needs to be continuously driven in a loop while a certain image output is desired. The team was able to handle updating the state of the game before and after each display cycle, but performing more intense operations while also driving the board may not be possible in this inefficient format. One potential way of improving the efficiency of this operation include using DMA to write all the data pins (on the same GPIO bank) and connect the display transmission to a timer interrupt, though it would be difficult to properly handle the complex sequence of control signals. Another potential way to improve performance would be to use a Real Time Operating System (RTOS), which would allow for more efficient multitasking. In any case, the inefficient bit-banged version was found to be sufficient for the team's application. The electrical schematic for how the STM microcontroller was connected to the matrix is shown in Figure 8.

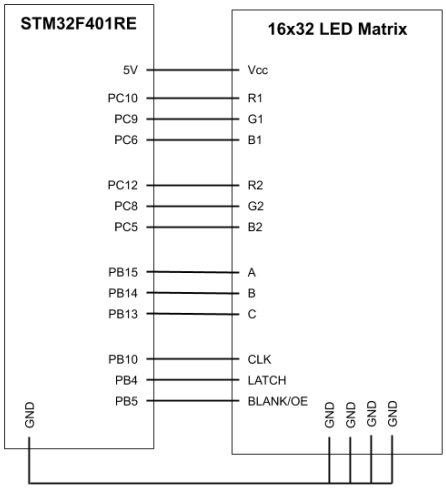

Figure 8: LED Matrix Electrical Schematic

IV. SOFTWARE DESCRIPTION

After calling various setup and initialization functions for the 84 MHz clock, note and duration timers, GPIO pins, and interrupts, main.c handles the overall flow of the game. This is done through a series of while loops in order to continuously update the LED matrix. The following new libraries were developed for this project:

The LED_MATRIX library contains functions for updating the LED matrix (LED_MATRIX.c and .h), as well as predefined game screens (game_screens.c and .h) for title, song selection, and score. Since the LED matrix is used in portrait orientation, the game screens are defined in portrait orientation. However, because the LED matrix updates horizontally in landscape orientation, the displayColorData function was written with landscape orientation in mind. Thus, the game screens must be transposed before calling this function. To display a completely new screen, the display data must be transposed and loaded into the 'display' array using the setTransposedScreen and transposeVert2StandardDisplay functions. Then, the function displayColorData is called within a while loop to continuously update the LED matrix.

The GAME_CONTROL library is responsible for the gameplay during a song, which involves figuring out the next set of buttons to be displayed and deciding what note to play based on user input. The while loop in main that runs during the song updates the LED matrix constantly, but also has a stopwatch that calls the refreshBoardScroll function at a certain board scroll rate. This board scroll rate is dependent on the song tempo and the number of LEDs used to display a quarter note beat, both of which are predefined in code for every song. The refreshBoardScroll function updates the global variable 'buttons_0', which is used for incrementing the display. To display a new row and shift all other rows down, the button data is mapped to the row to display using the convertButtons2NewCol function and the board is updated using the incrementDisplay function (this is done in main using these functions from the LED_MATRIX library). When the song is over, the refreshBoardScroll function calculates and updates the global 'accuracy_score' variable and sets the 'flag_end_song' to be 1. Once this flag is set high, main will exit the current while loop, update the 'display' array to be the score screen, and enter another while loop to display the score screen. If the user hits any buttons during the display of the score screen, the game will start again from the title screen.

For debugging purposes, the code has a 'god_mode' variable that can be set in GAME_CONTROL.c. When set to 1, the game will always play the correct notes. This is useful when checking that the programmed songs play as expected.

The MUSIC library contains the song arrays, which have three columns. The first column contains the pitch of each note in Hz. For readability, the pitches of every possible note are defined as their respective note names (e.g. A4, C5sharp) in notes.h. The second column of the song array contains the relative duration of each note. Once again for readability, the relative durations of each note are defined in notes.h as QUARTER = 250, HALF = 500, etc. Note that these 'relative durations' are not in milliseconds, as the tempo differs between and is hardcoded for each song. The third column of the song array contains the corresponding buttons to be pressed during that note. These are designated by the colors of the buttons, and each button corresponds to a different bit in a 5-bit number. Since the buttons from left to right are red, yellow, black (displayed as white on the LED matrix), green, and blue, the following assignments were made:

- R = 0b10000
- Y = 0b01000
- K = 0b00100
- G = 0b00010
- B = 0b00001

With these definitions, it is easy to designate integers for multiple buttons pressed at the same time: simply add the desired buttons together. For example, the integer to display red and green would be R + G = 0b10010.

V. RESULTS

The team was able to successfully build *8 Bit Hero* and meet all deliverables. The first deliverable was for the LED matrix to display data to indicate the proper button presses. As described in Section IV, songs were encoded with correct button presses corresponding to each note in the sequence, and this button press was converted into a set of colored LEDs on the matrix. A sample of the display showing colored during gameplay is shown in Figure 9.

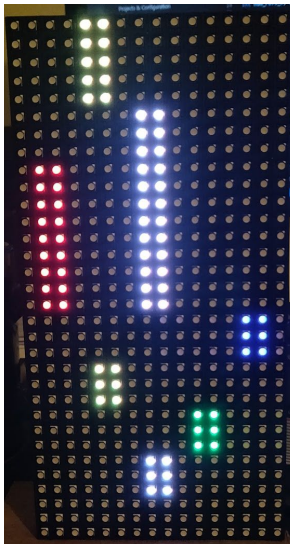

Figure 9: *8 Bit Hero* display during gameplay

Figure 10 shows an oscilloscope trace connected to the output of one of the buttons and the input of the speaker, in order to demonstrate both the fact that the speaker plays a note when a button is pressed correctly during the game as well as to demonstrate that the project meets the 50 ms latency requirement. As can be seen in Figure 10, it takes approximately 11µs for the speaker signal, shown in purple, to begin changing (PWM wave) after the button output, shown in orange, reaches a logical high value.

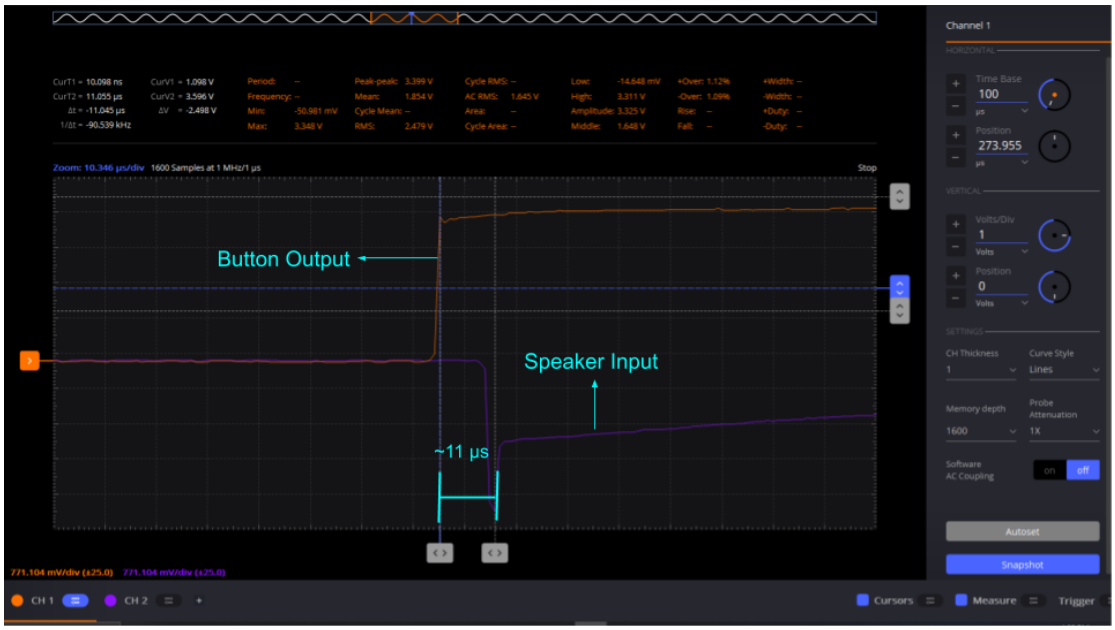

Figure 10: Oscilloscope Trace of Button/Speaker Latency

The calculation of the user score throughout the duration of a song was discussed in depth in *III.B Game Control System*. Following the completion of the song, the value of this score was used to update the LED display with a fraction out of 100 to indicate their score, as well as display a message about the players performance depending on their score bracket. A sample of these score screens is shown in Figure 11.

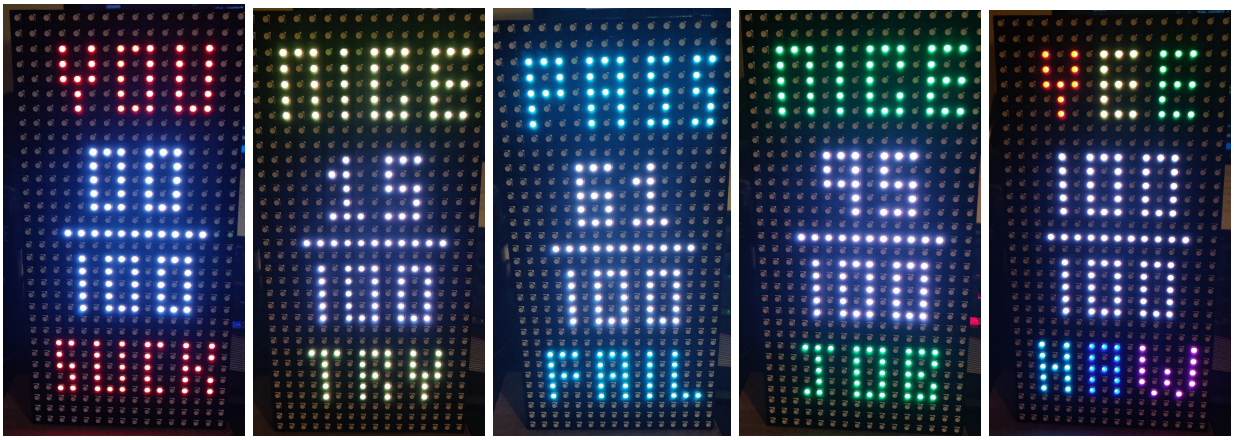

Figure 11: Sample Score Screens with Messages

The final deliverable required the team to code up three songs and label these choices as 1 - 3. The team ultimately ended up including 5 stages, one for each button input, and they were labeled 1-5. The songs include the following, in the order they appear: (1) Twinkle Twinkle Little Star, (2) The Final Countdown, (3) Star Trek: The Next Generation Main Theme, (4) Take on Me, (5) The Mandalorian Theme. After the title screen, a song select screen is displayed, where each song is represented by the color of the corresponding button, and the difficulty is represented by the length of the bar. After a song is selected, a number indicating the stage is shown, and then the game commences. The game display during this sequence is shown in Figure 12.

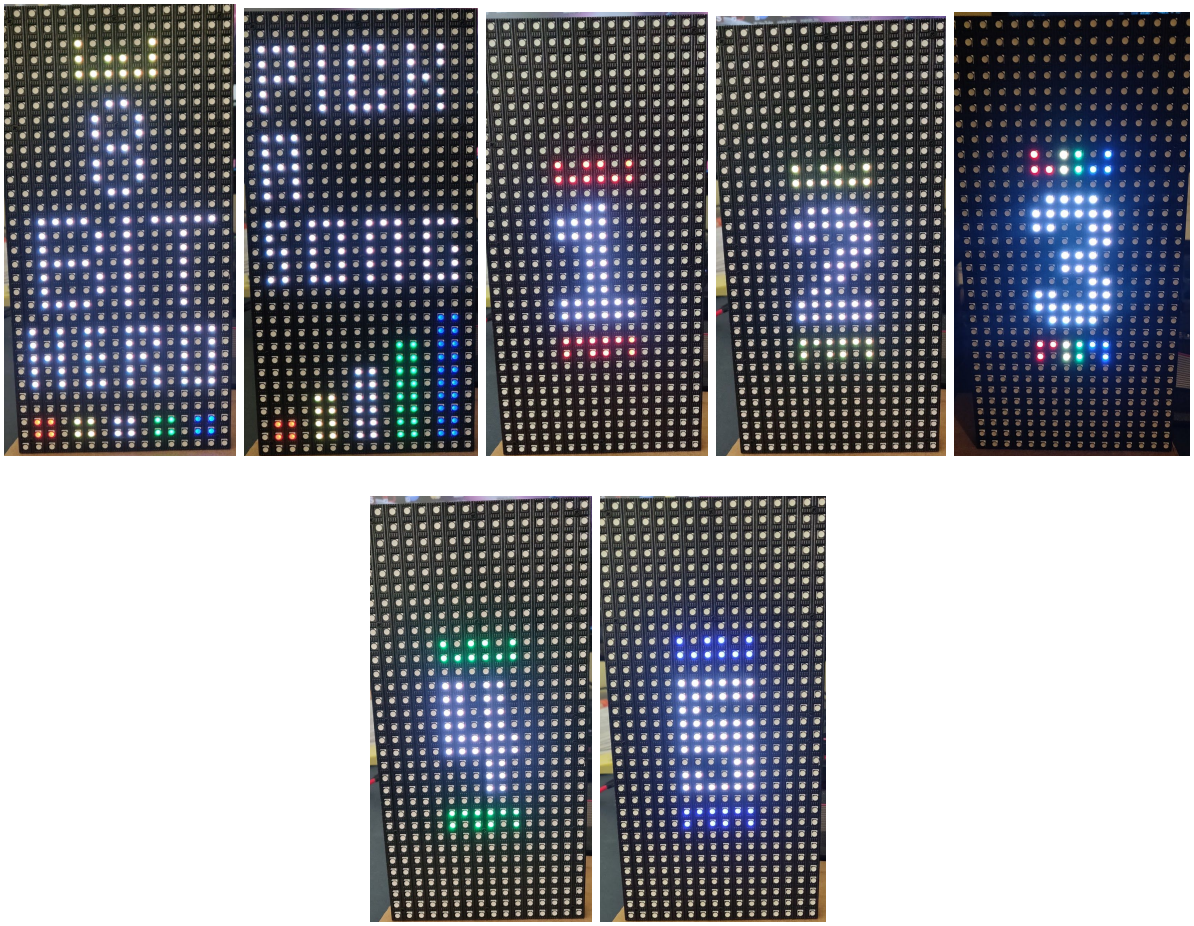

Figure 12: Display Sequence from start to playing a song. Top: (from left to right) the title screen, song select screen, level 1 screen, level 2 screen, level 3 screen. Bottom: (from left to right) level 4 screen, level 5 screen

In summary, the team was able to successfully implement a music rhythm game using an LED matrix, speaker, buttons, and an STM32F401RE Nucleo-64 board. The game had a total of 5 songs that a user could play, and upon completion the user would be presented with their score out of 100, as well as a different message based on their performance. The system was able to achieve a latency of 11µs, well below the 50 ms deliverable. The team found this project to be both enjoyable and educational, and would like to thank Prof. Brake and their classmates for their help and encouragement.

VI. PARTS LIST

Table 1 contains the list of parts used to create the setup of *8 Bit Hero*.

Table 1. Parts List

Item	**Source**	**Unit Price**	**Quantity**	**Total Cost**
Medium 16x32 RGB LED matrix[1]	Adafruit	$24.95	1	$24.95
5V 2A Switching Power Supply[1]	Adafruit	$7.95	1	$7.95
2.1 x 5.1 mm Male/Female DC supply screw adaptors[2]	Amazon	$5.99	1	$5.99

2.1mm male/female barrel extension cord (1.5m)[1, 2, 3]	Adafruit	$2.95	1	$2.95
Twidec Colored Pushed Buttons[4]	Amazon	$7.99	1	$7.99
Speaker - 0.5 W, 8Ω	E155 Lab Kit	N/A	1	N/A
LM386N Audio Amplifier	E155 Lab Kit	N/A	1	N/A
Pushbuttons	E155 Lab Kit	N/A	5	N/A
Assorted Resistors	E155 Lab Kit	N/A	N/A	N/A
Assorted Capacitors	E155 Lab Kit	N/A	N/A	N/A
Assorted wires	E155 Lab Kit	N/A	N/A	N/A
Total				$49.83[5]

1: These components were covered with the funds allocated for the project.
2: Only one of these components is required to interface with the matrix, but both were purchased since the matrix had the opposite power. connector than the one that would have worked with the cord initially purchased.
3: Cheaper single adaptors could be found, but this pack was purchased due to the speed of Amazon Prime shipping.
4: These buttons were purchased to make a more interesting gamepad, which unfortunately did not make it to the final design (yet).
5: This total reflects the total cost of components, not the total cost reimbursed by HMC.

VII. REFERENCES

[1] J. Ganssle. "A Guide to Debouncing, or, How to Debounce a Contact in Two Easy Pages," *The Ganssle Group*. Reisterstown, MD. Available: http://www.ganssle.com/debouncing.pdf

[2] "LM386 Low Voltage Audio Power Amplifier," *Texas Instruments*. May, 2017. Available: https://www.ti.com/lit/ds/symlink/lm386.pdf

[3] J. Liang and D. Sobek. "Final Project Report: Bead Maze with LED Matrix and Accelerometer," Claremont, CA, 2019. Available: http://pages.hmc.edu/harris/class/e155/projects19/Sobek_Liang.pdf

[4] R. Alkhamis and S. Griffith. "LED Visual Art," Claremont, CA, 2019. Available: http://pages.hmc.edu/harris/class/e155/projects19/Alkhamis_Griffith.pdf

[5] S. Malpani and M. Yao "LED Snake with Tilt Controls,"Claremont, CA, 2019. Available: http://pages.hmc.edu/harris/class/e155/projects19/Malpani_Yao.pdf

[6] G. Akins. "RGB LED Panel Driver Tutorial." bikerglen.com. Available: https://bikerglen.com/projects/lighting/led-panel-1up/

[7] P. Burgess. "32x16 and 32x32 RGB LED Matrix." adafruit.com. Available: https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/overview

[8] R. Logic. "Adafruit RGB LED Matrix." rayslogic.com. Available: http://rayslogic.com/propeller/Programming/AdafruitRGB/AdafruitRGB.htm

VIII. APPENDICES

A. Main files

The following header file main.h was used to define the list of all include statements needed in main.c

```
#ifndef MAIN_H
#define MAIN_H

#include <stdint.h>

// Include other peripheral libraries

#include "STM32F401RE_GPIO.h"
#include "STM32F401RE_FLASH.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_SPI.h"
#include "STM32F401RE_TIM.h"
#include "GAME_CONTROL.h"
#include "interrupts.h"
#include "LED_Matrix.h"
#include "notes.h"
#include "game_screens.h"
#include <stdio.h>

#endif
```

The following main file main.c was used to run *8 Bit Hero* on the STM32F401RE board.

```
// GPIO_control main function.c

#include "main.h"



uint8_t transposed_display[32][16] = {0};

uint8_t display[16][32] = {0};

int song[MAX_SONG_LEN][3] = {0};

// Variables that the LED controller code needs to access
int buttons_0 = 0;                    // for what LEDs to display
next
int flag_new_row_available = 0;  // for checking whether we need to
update the LED row
```

```
    // for reporting score
    int accuracy_score;
    int flag_end_song = 1;
    uint8_t button_state;


    int main(void) {
        // configure the clock (84MHz using PLL) and FLASH
        configureFlash();
        configureClock();

        // enable clock to get to NOTE TIMER (TIM2) and DELAY TIMER
(TIM5) and DURATION TIMER (TIM3)
        RCC->APB1ENR.TIMER2 = 1;
        RCC->APB1ENR.TIMER3 = 1;
        RCC->APB1ENR.TIMER5 = 1;

        //enable clock to SYSFG
        RCC->APB2ENR |= (1<<14); // set SYSFGEN bit

        // enable clock to BUTTON PINS (GPIOA and GPIOC)
        RCC->AHB1ENR.GPIOAEN = 1;
        RCC->AHB1ENR.GPIOBEN = 1;
        RCC->AHB1ENR.GPIOCEN = 1;

        // configure TIM2 to PWM mode
        setupPWM(NOTE_TIM);
        initTIM(DELAY_TIM);
        initDurationTimer(DUR_TIM);

        // set NOTE PIN as an alternate function
        pinMode(NOTE_GPIO, NOTE_PIN, GPIO_ALT);
        // configure PA5 (NOTE PIN) as AF1 to get TIM2_CH1
        GPIOA->AFRL |= (1 << NOTE_PIN*4);

        // set button pins as inputs
        pinMode(RED_GPIO, RED_BUTTON_PIN, GPIO_INPUT);
        pinMode(YELLOW_GPIO, YELLOW_BUTTON_PIN, GPIO_INPUT);
        pinMode(BLACK_GPIO, BLACK_BUTTON_PIN, GPIO_INPUT);
```

```
    pinMode(GREEN_GPIO, GREEN_BUTTON_PIN, GPIO_INPUT);
    pinMode(BLUE_GPIO, BLUE_BUTTON_PIN, GPIO_INPUT);


    //intialize the matrix pins
    initMatrixPins();

    // Set EXTICR  register for PA5-9
        // clear the bits
        *SYSCFG_EXTICR2 &= ~(0b1111 << 4); // EXTI 5
        *SYSCFG_EXTICR2 &= ~(0b1111 << 8); // EXTI 6
        *SYSCFG_EXTICR2 &= ~(0b1111 << 12); // EXTI 7
        *SYSCFG_EXTICR3 &= ~(0b1111 << 0); // EXTI 8
        *SYSCFG_EXTICR3 &= ~(0b1111 << 4); // EXTI 9
        // configure EXTI9_5 to GPIOA
            //0b0010 = GPIOC
            //0b0000 = GPIOA
        *SYSCFG_EXTICR2 |= (0b0000 << 4);  // EXTI 5
        *SYSCFG_EXTICR2 |= (0b0000 << 8);  // EXTI 6
        *SYSCFG_EXTICR2 |= (0b0000 << 12);  // EXTI 7
        *SYSCFG_EXTICR3 |= (0b0000 << 0);  // EXTI 8
        *SYSCFG_EXTICR3 |= (0b0000 << 4);  // EXTI 9

    // Enable interrupts globally
    __enable_irq();

    // Configure interrupt for falling edge of GPIO PA5-9
        // Configure mask bit
        EXTI->IMR |= (1 << RED_BUTTON_PIN);
        EXTI->IMR |= (1 << YELLOW_BUTTON_PIN);
        EXTI->IMR |= (1 << BLACK_BUTTON_PIN);
        EXTI->IMR |= (1 << GREEN_BUTTON_PIN);
        EXTI->IMR |= (1 << BLUE_BUTTON_PIN);
        // Enable rising edge trigger
        EXTI->RTSR |= (1 << RED_BUTTON_PIN);
        EXTI->RTSR |= (1 << YELLOW_BUTTON_PIN);
        EXTI->RTSR |= (1 << BLACK_BUTTON_PIN);
        EXTI->RTSR |= (1 << GREEN_BUTTON_PIN);
        EXTI->RTSR |= (1 << BLUE_BUTTON_PIN);
        // Enable falling edge trigger
```

```
            EXTI->FTSR |= (1 << RED_BUTTON_PIN);
            EXTI->FTSR |= (1 << YELLOW_BUTTON_PIN);
            EXTI->FTSR |= (1 << BLACK_BUTTON_PIN);
            EXTI->FTSR |= (1 << GREEN_BUTTON_PIN);
            EXTI->FTSR |= (1 << BLUE_BUTTON_PIN);
            // Turn on EXTI9_5 interrupt in NVIC_ISER0
                // see main.h for bits that corresponds to other EXTI
lines
            *NVIC_ISER0 |= (1 << 23); //bit 23 of the ISRE

        //define color data arrays
        uint8_t R1[32] = {0};
        uint8_t G1[32] = {0};
        uint8_t B1[32] = {0};
        uint8_t R2[32] = {0};
        uint8_t G2[32] = {0};
        uint8_t B2[32] = {0};

        //define time_elapsed variable;
        volatile double time_elapsed = 0;

        //reference to starting point in code. Use to jump back to the
beginning after the song has been played
        start:

        //set the display to the start screen
        setTransposedScreen(transposed_display, HOME_SCREEN);
        transposeVert2StandardDisplay(transposed_display, display);

        //START SCREEN
        //display the start screen until any button is pressed
        while(button_state == 0){

            displayColorData(display, R1, G1, B1, R2, G2, B2);

        }

        //delay one second in between the screens
        delay_millis(DELAY_TIM, 1000);
```

```
        //SONG SELECT SCREEN
        setTransposedScreen(transposed_display, SONG_SELECT_SCREEN);
        transposeVert2StandardDisplay(transposed_display, display);

        //display the song select screen while no buttons are pressed
        while(button_state == 0){

            displayColorData(display, R1, G1, B1, R2, G2, B2);

        }

        //save the user's selection to another variable
        volatile uint8_t user_selection = button_state;

        //initialize variables that relate to the song and how it's
displayed
        uint8_t  song_selection; //indicates which song to use
        uint8_t leds_per_beats; //indicates how long a quarter note is
on the display
        volatile uint32_t refreshRate; //indicates the rate at which we
need to increment the display data

        //setup the song based on the user input
        switch(user_selection){

            //Level 1: Twinkle twinkle Little Star
            case 0b10000:
                song_selection = SONG_TWINKLE_TWINKLE;
                leds_per_beats = 4;
                refreshRate = calcRefreshRate(100,leds_per_beats);
                setTransposedScreen(transposed_display, LEVEL_1);
                transposeVert2StandardDisplay(transposed_display,
display);
                break;

            //Level 2: The Final Countdown
            case 0b01000:
                song_selection = SONG_FINALCOUNTDOWN;
                leds_per_beats = 4;
```

```
                refreshRate = calcRefreshRate(128*2, leds_per_beats);
                setTransposedScreen(transposed_display, LEVEL_2);
                transposeVert2StandardDisplay(transposed_display,
display);
                break;

            //Level 3: Star Trek the Next Generation Main Theme
            case 0b00100:
                song_selection = SONG_STARTREK;
                leds_per_beats = 6;
                refreshRate = calcRefreshRate(120, leds_per_beats);
                setTransposedScreen(transposed_display, LEVEL_3);
                transposeVert2StandardDisplay(transposed_display,
display);
                break;

            //Song 4: Take on Me
            case 0b00010:
                song_selection = SONG_TAKEONME;
                leds_per_beats = 4;
                refreshRate = calcRefreshRate(169, leds_per_beats);
                setTransposedScreen(transposed_display, LEVEL_4);
                transposeVert2StandardDisplay(transposed_display,
display);
                break;

            //Song 5: The Mandalorian Theme
            case 0b00001:
                song_selection = SONG_MANDO;
                leds_per_beats = 12;
                refreshRate = calcRefreshRate(84, leds_per_beats);
                setTransposedScreen(transposed_display, LEVEL_5);
                transposeVert2StandardDisplay(transposed_display,
display);
                break;

            //Default case: Test Song
            default:
                song_selection = SONG_TEST;
                leds_per_beats = 4;
```

```
                refreshRate = 250;
                setTransposedScreen(transposed_display, LEVEL_5);
                transposeVert2StandardDisplay(transposed_display,
display);
                break;

        }

        //display the song selection screen for 5 seconds
        time_elapsed = 0;
        resetDurationTimer(DUR_TIM);
        while(time_elapsed < 5000){

            displayColorData(display, R1, G1, B1, R2, G2, B2);

            time_elapsed = getElapsedTime(DUR_TIM);

        }

        //prepare to play the actual song
        getSongData(song, song_selection);
        initNewGame((int*)song, leds_per_beats);
        uint8_t newCol[16] = {0};
        flag_end_song = 0;
        flag_new_row_available = 0;
        resetDurationTimer(DUR_TIM);

        //play the song while the end song flag is low
        while(flag_end_song != 1){

           time_elapsed = getElapsedTime(DUR_TIM);

            //update board every x milliseconds
            if(time_elapsed > refreshRate){

                //update the button state
                refreshBoardScroll(&accuracy_score, &flag_end_song,
&buttons_0, &flag_new_row_available);

                //convert the button data into a display row
```

```
                convertButtons2NewCol(newCol, buttons_0);

                //increment the display
                incrementDisplay(display, newCol);

                resetDurationTimer(DUR_TIM);
            }

            displayColorData(display, R1, G1, B1, R2, G2, B2);

            //check the button press to see if it matches the current
display, update the score if so
            checkScore();

        }

        //wait a second before displaying the score
        delay_millis(DELAY_TIM, 1000);

        //set the display with the user's score to be displayed
        setTransposedScreen(transposed_display, SCORE_SCREEN);
        updateScoreScreen(accuracy_score, transposed_display);
        transposeVert2StandardDisplay(transposed_display, display);

        //display the score for the user until they want to play again
        while(1){

            displayColorData(display, R1, G1, B1, R2, G2, B2);

            //restart the game if the user hits any buttons
            if(button_state != 0){
                delay_millis(DELAY_TIM, 1000);
                goto start;
            }

        }

    }
```

```
        // this function is specifc to EXTI line 1 (GPIOx1)
    void EXTI9_5_IRQHandler(void){

        // Check that the button EXTI was what triggered our interrupt
        if (EXTI->PR & (0b11111<<RED_BUTTON_PIN)){

            // If so, clear the interrupt
            EXTI->PR |= (0b11111 << RED_BUTTON_PIN);

            //run the button interrupt function
            button_IRQ();
        }
    }

    //button press interrupt function
    void button_IRQ(void){

        //set the global button variable to whatever buttons are being
pressed
        button_state = readButtons();

        //if the song is currently playing, check if the correct buttons
are pressed
        //and update the note output
        if(flag_end_song == 0){

            checkButtonsUpdatePWM();

        }

    }
```

B. Game Controller Library

The following header file GAME_CONTROL.h was used to define values for the game controller.

```
// STM32F401RE_GPIO.h
// Header for GPIO functions

#ifndef GAME_CONTROL_H
#define GAME_CONTROL_H

#include <stdint.h> // Include stdint header

//////////////////////////////////////////////////////////////////////////////
// Definitions
//////////////////////////////////////////////////////////////////////////////

#define RED_GPIO GPIOA
#define RED_BUTTON_PIN 5
#define YELLOW_GPIO GPIOA
#define YELLOW_BUTTON_PIN 6
#define BLACK_GPIO GPIOA
#define BLACK_BUTTON_PIN 7
#define GREEN_GPIO GPIOA
#define GREEN_BUTTON_PIN 8
#define BLUE_GPIO GPIOA
#define BLUE_BUTTON_PIN 9

// // Variables that the LED controller code needs to access
// int buttons_0;                    // for what LEDs to display next
// int flag_new_row_available = 0;  // for checking whether we need
to update the LED row

// // for reporting score
// int accuracy_score;
// int flag_end_song;

//////////////////////////////////////////////////////////////////////////////
// Bitfield structs
```

```
    ////////////////////////////////////////////////////////////////////////
//////////



    ////////////////////////////////////////////////////////////////////////
//////////
    // Function prototypes
    ////////////////////////////////////////////////////////////////////////
//////////

    uint8_t readButtons(void);
    /* read buttons does a digital read of each of the pins
    defined for each user input button above and combines
    this information into one 5 bit binary number
    from MSB to LSB: red, yellow, black, green, blue */

    void initNewGame(int *notes, int leds_per_beat);
    /* initializes song counters and indices, also copies the
    value of the notes pointer and leds_per_beat to variables
    that the rest of the game control functions can access */

    void checkButtonsUpdatePWM(void);
    /* checks whether actual buttons == expected buttons, updates
    PWM accordingly as well as the play_correct variable */

    void refreshBoardScroll(int *accuracy_score, int *flag_end_song, int
*buttons_0, int *flag_new_row_available);
    /* called whenever the board should scroll, updates the global
    buttons_0 variable for the new LED row to display, also updates
    current buttons_1 and pitch_1 */

    void checkScore(void);
    /* checks and updates the points variable if points should be and
    have not yet been assigned for a particular LED row */

    int calcRefreshRate(int tempo, int leds_per_beats);
    /* calcRefreshRate takes in the tempo of song and the
    number of LEDs to play per quarter note and outputs
    the appropriate rate at which the board should
```

```
      scroll to ensure that the song is played at the correct
      tempo */

      #endif
```

The following library file GAME_CONTROL.c implemented game control functionality.

```
      #include "GAME_CONTROL.h"
      #include "STM32F401RE_GPIO.h"
      #include "STM32F401RE_TIM.h"
      #include "notes.h"

      // for debugging, automatically plays all the right notes
      int god_mode = 0;

      // indices for the arrays
      int i_0, i_1;

      //int prev_pitch_1;
      int prev_i_1;
      int points, possible_points;
      int points_assigned;
      // ex. scroll_rate_ms = 1000 * 60 / tempoBPM / leds_per_beat;     //
in ms
      // int scroll_rate_ms; // unsure if need this here or wherever we
have that interrupt

      // counters for figuring out when to increment indices
      // Note: cnt_1 starts at -31 because current notes don't start right
away
      int cnt_0, cnt_1;
      int cnt_past_blank;
      int num_refs_0, num_refs_1;

      int play_correct = 1;
      int buttons_1 = 0b00000;    // buttons that are currently supposed
(expected) to be pressed
      int pitch_1 = 0;            // current note that is supposed to be
played
```

```
    int *notes_ptr, leds_pb;

    uint8_t readButtons(void){
        uint8_t RED = digitalRead(RED_GPIO, RED_BUTTON_PIN);
        uint8_t YELLOW = digitalRead(YELLOW_GPIO, YELLOW_BUTTON_PIN);
        uint8_t BLACK = digitalRead(BLACK_GPIO,BLACK_BUTTON_PIN);
        uint8_t GREEN = digitalRead(GREEN_GPIO,GREEN_BUTTON_PIN);
        uint8_t BLUE = digitalRead(BLUE_GPIO,BLUE_BUTTON_PIN);

        uint8_t buttons = 0;
        buttons |= (RED << 4)
                |  (YELLOW << 3)
                |  (BLACK << 2)
                |  (GREEN << 1)
                |  (BLUE << 0);
        return buttons;
    }

    void initNewGame(int *notes, int leds_per_beat) {
        // indices for the arrays
        i_0 = 0, i_1 = 0;

        //int prev_pitch_1;

        points = 0;
        possible_points = 0;

        // counters for figuring out when to increment indices
        // Note: cnt_1 starts at -31 because current notes don't start
right away
        cnt_0 = 0;
        cnt_1 = -31;
        cnt_past_blank = -31;

        notes_ptr = notes;
        leds_pb = leds_per_beat;

        //int num_refs_0 = notes[0][1] / QUARTER * leds_per_beat;
        //int num_refs_1 = notes[0][1] / QUARTER * leds_per_beat;
```

```
          num_refs_0 = (*(notes_ptr + i_0*3 + 1) * (leds_pb)) /
(float)QUARTER;
          num_refs_1 = (*(notes_ptr + i_1*3 + 1) * (leds_pb)) /
(float)QUARTER;
      }

      int calcRefreshRate(int tempo, int leds_per_beats){

          int refreshRate = ((60.0/tempo)/leds_per_beats)*1000;

          return refreshRate;

      }

      void checkScore(void){
          // if points haven't been assigned for this particular row
              // and if the expected buttons aren't 0
          if ((points_assigned == 0) && (buttons_1 != 0) && play_correct
== 1) {
              points++;
              points_assigned = 1;
          }

      }

      // checks which buttons are currently pressed with what buttons
SHOULD be pressed
      // updates play_correct accordingly (play_correct is true when
actual == expected)
      // checks what note should be played if play_correct is true
      // calls setPWMFreq to play this note if play_correct is true,
      // otherwise calls setPWMFreq to play nothing or a wrong note
      void checkButtonsUpdatePWM(void) {
          // figure out which buttons are currently pressed
          // can handle this here or in another function
          int actual_buttons = readButtons();          // i.e. 0b01001 or
something

          // figure out which buttons should be pressed at this moment
```

```
        int expected_buttons = buttons_1;     // probably don't need a
separate variable

        // simplified version:
        if (actual_buttons == expected_buttons || god_mode == 1) {
            play_correct = 1;   // needs to be a global variable
            updatePWM(NOTE_TIM, pitch_1);

        }
        else {
            play_correct = 0;
            if (actual_buttons == 0) {
                // if no buttons are pressed, don't play anything
                updatePWM(NOTE_TIM, 0);
            }
            else {
                // if the wrong buttons are pressed
                // currently set to not play anything
                updatePWM(NOTE_TIM, 0);

            }
        }
    }

    // a[i][j] = *(ptr + ( i * numofcols) + j)
    // this gets called when the interrupt for the scrolling rate gets
triggered
    void refreshBoardScroll(int *accuracy_score, int *flag_end_song, int
*buttons_0, int *flag_new_row_available) {

        // Update indices if needed
        int calc_score = 0;

        // for whether to call checkButtonsUpdatePWM
        prev_i_1 = i_1;

        // increment i_0 if the led duration stuff has passed
        if (cnt_0 == num_refs_0 ) {
            i_0++;
            //num_refs_0 = notes_ptr[i_0][1] / QUARTER * (*leds_pb_ptr);
```

```
            num_refs_0 = (*(notes_ptr + i_0*3 + 1) * (leds_pb)) /
(float)QUARTER;
            cnt_0 = 0;

            if (num_refs_0 == 0) num_refs_0 = -1;   // once we get to
the end of the array, stop incrementing
        }

        // increment i_1 if the current duration has passed
        if (cnt_1 == num_refs_1) {
            i_1++;
            //num_refs_1 = notes_ptr[i_1][1] / QUARTER * (*leds_pb_ptr);
            num_refs_1 = (*(notes_ptr + i_1*3 + 1) * (leds_pb)) /
(float)QUARTER;
            cnt_1 = 0;

            // exit game if there are no more notes left (indicated by a
duration of 0)
            if (num_refs_1 == 0) {
                // report score

                calc_score = (int)(100 *
(float)points/(float)possible_points);

                if (calc_score < 0) {
                    calc_score = 0;
                }
                *accuracy_score = calc_score;
                // set flag for the end of song
                *flag_end_song = 1;
            }
        }

        // update global variable for expected buttons pressed
        //buttons_1 = notes_ptr[i_1][2];
        buttons_1 = *(notes_ptr + i_1*3 + 2);
        // update which buttons should be displayed next on the LED
        //buttons_0 = notes_ptr[i_0][2];
        *buttons_0 = *(notes_ptr + i_0*3 + 2);
        volatile check = *buttons_0;
```

```
          *flag_new_row_available = 1;  // let jimmy know to update the
LED

          // update global variable for current pitch that is supposed to
be played
          //prev_pitch_1 = pitch_1;
          //pitch_1 = notes_ptr[i_1][0];
          pitch_1 = *(notes_ptr + i_1*3);

          //call update if a new pitch needs to be played
          //if (prev_pitch_1 != pitch_1) {
          if (prev_i_1 != i_1) {
              checkButtonsUpdatePWM();
          }

          // increment possible points if expected buttons aren't 0
          if (buttons_1 != 0) {
              possible_points++;
          }

          cnt_0++;
          cnt_1++;
          cnt_past_blank++;
          points_assigned = 0;
      }
```

C. LED Matrix Library

The following header file LED_MATRIX.h was used to define values for the LED matrix drivers.

```
#ifndef LED_MAT_DEF
#define LED_MAT_DEF

#include "STM32F401RE_GPIO.h"
#include "STM32F401RE_TIM.h"


////////////////////////////////////////
////// LED MATRIX GPIO PIN DEFS ////////
////////////////////////////////////////


//NOTE: for simplicity sake, using GPIO_PAx defn even for pins from
different GPIOs

//address pins
#define MATRIX_A_GPIO GPIOB
#define MATRIX_A_PIN GPIO_PA15

#define MATRIX_B_GPIO GPIOB
#define MATRIX_B_PIN GPIO_PA14

#define MATRIX_C_GPIO GPIOB
#define MATRIX_C_PIN GPIO_PA13

//Control Pins

#define MATRIX_CLK_GPIO GPIOB
#define MATRIX_CLK_PIN GPIO_PA10

#define MATRIX_LAT_GPIO GPIOB
#define MATRIX_LAT_PIN GPIO_PA4

#define MATRIX_BLANK_GPIO GPIOB
#define MATRIX_BLANK_PIN GPIO_PA5

//Data Pins
#define MATRIX_DATA_GPIO GPIOC
```

```
#define MATRIX_R1_GPIO MATRIX_DATA_GPIO
#define MATRIX_R1_PIN GPIO_PA10

#define MATRIX_R2_GPIO MATRIX_DATA_GPIO
#define MATRIX_R2_PIN GPIO_PA12

#define MATRIX_G1_GPIO MATRIX_DATA_GPIO
#define MATRIX_G1_PIN GPIO_PA9

#define MATRIX_G2_GPIO MATRIX_DATA_GPIO
#define MATRIX_G2_PIN GPIO_PA8

#define MATRIX_B1_GPIO MATRIX_DATA_GPIO
#define MATRIX_B1_PIN GPIO_PA6

#define MATRIX_B2_GPIO MATRIX_DATA_GPIO
#define MATRIX_B2_PIN GPIO_PA5


////////////////////////////////////////
////// LED MATRIX COLOR REFERENCES//////
////////////////////////////////////////

#define K 0b000 //black
#define B 0b001 //blue
#define G 0b010 //green
#define C 0b011 //cyan
#define R 0b100 //red
#define P 0b101 //purple
#define Y 0b110 //yellow
#define W 0b111 //white

////////////////////////////////////////
////// LED MATRIX FUNCTION DEFNS///////
////////////////////////////////////////

void initMatrixPins();
/* */
```

```
      void setAddr(uint8_t addr);
      /* */

      void calcColorData(uint8_t display[16][32], uint8_t currRow, uint8_t
R1_data[], uint8_t G1_data[], uint8_t B1_data[], uint8_t R2_data[],
uint8_t G2_data[], uint8_t B2_data[]);
      /* */

      void incrementDisplay(uint8_t display[16][32], uint8_t newRow[32]);
      /* */

      void displayColorData(uint8_t display[16][32], uint8_t R1_data[],
uint8_t G1_data[], uint8_t B1_data[], uint8_t R2_data[], uint8_t
G2_data[], uint8_t B2_data[]);
      /* */

      void setColorDataPins(uint8_t R1_val, uint8_t G1_val, uint8_t
B1_val, uint8_t R2_val, uint8_t G2_val, uint8_t B2_val);
      /* */

      void transposeVert2StandardDisplay(uint8_t design[32][16], uint8_t
display[16][32]);
      /* */

      void convertButtons2NewCol(uint8_t newCol[16], uint8_t
buttons_to_press);
      /* */

      #endif
```

The following library file LED_MATRIX.c was used to display data to the LED matrix.

```
      #include "LED_Matrix.h"



      void initMatrixPins(){

          //function: initMatrixPins
          //author: Jimmy Fernandez (jpfernandez@g.hmc.edu)
          //purpose: initialize all the pins used to write to the LED
matrix
```

```
        //address pins
        pinMode(MATRIX_A_GPIO, MATRIX_A_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_B_GPIO, MATRIX_B_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_C_GPIO, MATRIX_C_PIN, GPIO_OUTPUT);

        //control pins
        pinMode(MATRIX_CLK_GPIO, MATRIX_CLK_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_BLANK_GPIO, MATRIX_BLANK_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_LAT_GPIO, MATRIX_LAT_PIN, GPIO_OUTPUT);

        //data pins
        pinMode(MATRIX_R1_GPIO, MATRIX_R1_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_R2_GPIO, MATRIX_R2_PIN, GPIO_OUTPUT);

        pinMode(MATRIX_G1_GPIO, MATRIX_G1_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_G2_GPIO, MATRIX_G2_PIN, GPIO_OUTPUT);

        pinMode(MATRIX_B1_GPIO, MATRIX_B1_PIN, GPIO_OUTPUT);
        pinMode(MATRIX_B2_GPIO, MATRIX_B2_PIN, GPIO_OUTPUT);

    }

    void setAddr(uint8_t addr){

        //function: setAddr
        //author: Jimmy Fernandez (jpfernandez@g.hmc.edu)
        //Purpose: set the address pins A, B and C based on the value of
addr
        //Inputs:
            //addr - the current address to set the pins to


        //NOTE: might be a more optimal method of setting if we put all
these address in the same GPIO bank
        //assuming addr is formated such that bit 0 = A, bit 1 = B, bit
2 = C
        digitalWrite(MATRIX_A_GPIO, MATRIX_A_PIN, (addr >> 0) & 1);
        digitalWrite(MATRIX_B_GPIO, MATRIX_B_PIN, (addr >> 1) & 1);
        digitalWrite(MATRIX_C_GPIO, MATRIX_C_PIN, (addr >> 2) & 1);
```

```
    }


    void calcColorData(uint8_t display[16][32], uint8_t currRow, uint8_t
R1_data[], uint8_t G1_data[], uint8_t B1_data[], uint8_t R2_data[],
uint8_t G2_data[], uint8_t B2_data[]){
        //Function: calcColorData
        //author: Jimmy Fernandez (jpfernandez@g.hmc.edu)
        //Purpose: to write the color data to for rows "currRow" and
"currRow" + 8 to the provided RGB data arrays
        //Inputs:
            //display - a 16x32 uint8_t array that contains the display
data
            //currRow - a uint8_t representing the current value of the
address ADDR we want data for
            //R1_data - a uint8_t 32 element array representing the red
led dat for row ADDR
            //G1_data - a uint8_t 32 element array representing the
green led dat for row ADDR
            //B1_data - a uint8_t 32 element array representing the blue
led dat for row ADDR
            //R2_data - a uint8_t 32 element array representing the red
led dat for row ADDR + 8
            //G2_data - a uint8_t 32 element array representing the
green led dat for row ADDR + 8
            //B2_data - a uint8_t 32 element array representing the blue
led dat for row ADDR + 8


        uint8_t row1_color;
        uint8_t row2_color;
        for(uint8_t col = 0; col < 32; col++){

            //retrieve the row colors
            row1_color = display[currRow][col];
            row2_color = display[currRow + 8][col];

            R1_data[col] = (row1_color >> 2) & 1;
            G1_data[col] = (row1_color >> 1) & 1;
            B1_data[col] = (row1_color >> 0) & 1;
```

```
            R2_data[col] = (row2_color >> 2) & 1;
            G2_data[col] = (row2_color >> 1) & 1;
            B2_data[col] = (row2_color >> 0) & 1;

        }
    }

    void incrementDisplay(uint8_t display[16][32], uint8_t newCol[16]){

        //Function: incrementDisplay
        //author: Jimmy Fernandez (jpfernandez@.g.hmc.edu)
        //Purpose: to have the game data "fall" in the direction of the
positive columns, and add a new passed in row to the top
        //Inputs:
            //display - a 16x32 uint8_t array that contains the display
data
            //newCol - a 16 element uint8_t array that contains the new
"column" of data to append onto the top (shifting everything else down)

        //increment over all the columns, starting with the second to
last one
        for(int col = 30; col >= 0; col--){
            for(int row = 0; row < 16; row++){

                display[row][col + 1] = display[row][col];
            }
        }


        //add the last column
        for(int row = 0; row < 16; row++){

                display[row][0] = newCol[row];

                }
    }
```

```
     void setColorDataPins(uint8_t R1_val, uint8_t G1_val, uint8_t
B1_val, uint8_t R2_val, uint8_t G2_val, uint8_t B2_val){

         //function: setColorDataPins
         //author: Jimmy Fernandez
         //purpose: function that sets all the data pins to a value
simulatenously. Did not end up seeing use due to
         //no noticeable improvements in efficiency
         //Inputs:
             //R1_val - a uint8_t that represents whether or note the R1
data is one or zero
             //G1_val - a uint8_t that represents whether or note the G1
data is one or zero
             //B1_val - a uint8_t that represents whether or note the B1
data is one or zero
             //R2_val - a uint8_t that represents whether or note the R2
data is one or zero
             //G2_val - a uint8_t that represents whether or note the G2
data is one or zero
             //B2_val - a uint8_t that represents whether or note the B2
data is one or zero


         //If the value is a 1, we want to set the pin, which is in the
lower 16 bits
         //if the value is a 0, we want to right to the reset pin, which
is in the upper
         //by adding 16*(!value), we add 16 to the pin when value is zero
(reset is upper 16) and
         //add 0 to the pin when we want to set it
         MATRIX_DATA_GPIO->BSRR = (1 << (MATRIX_R1_PIN + 16*(!R1_val))) |
                                  (1 << (MATRIX_G1_PIN + 16*(!G1_val))) |
                                  (1 << (MATRIX_B1_PIN + 16*(!B1_val))) |
                                  (1 << (MATRIX_R2_PIN + 16*(!R2_val))) |
                                  (1 << (MATRIX_G2_PIN + 16*(!G2_val))) |
                                  (1 << (MATRIX_B2_PIN + 16*(!B2_val)));

     }
```

```
     void transposeVert2StandardDisplay(uint8_t design[32][16], uint8_t
display[16][32]){

         //function: transposeVert2StandardDisplay
         //author: Jimmy Fernandez (jpfernandez@g.hmc.edu)
         //purpose: To take a display in the 32x16 format that we expect
to see while playing the game and tranpose it into the 16x32
         //         design used to transmit the data. Created for ease of
designing more complicated static screens for use in the game.
         //Inputs:
             //design - a 32x16 uint8_t array that contains the color
data we want to display
             //display - a 16x32 uint8_t array that represents the data
to be sent to the display
         for(int row = 0; row < 16; row++){
             for(int col = 0; col < 32; col++){
                 display[row][col] = design[col][15-row];
             }
         }
     }

     void displayColorData(uint8_t display[16][32], uint8_t *R1, uint8_t
*G1, uint8_t *B1, uint8_t *R2, uint8_t *G2, uint8_t *B2){

         //Function: displayColorData
         //Author: Jimmy Fernandez (jpfernandez@g.hmc.edu)
         //Purpose: Goes through the sequence of displaying each of the
LED Matrix rows (one sequence of 16 rows)
         //Inputs:
             //display - a 16x32 uint8_t matrix that represents what the
display should look lik
             //R1 - a 32 element array that contains boolean data
indicating if the red LED at each column of ADDR should be turned on
             //G1 - a 32 element array that contains boolean data
indicating if the blue LED at each column of ADDR should be turned on
             //B1 - a 32 element array that contains boolean data
indicating if the green LED at each column of ADDR should be turned on
             //R2 - a 32 element array that contains boolean data
indicating if the red LED at each column of ADDR + 8 should be turned on
```

```
            //G2 - a 32 element array that contains boolean data
indicating if the blue LED at each column of ADDR + 8 should be turned on
            //B2 - a 32 element array that contains boolean data
indicating if the green LED at each column of ADDR + 8 should be turned on

        uint8_t addr;
        addr = 0; //reset to row 0

        //Iterate through every row
        for(uint8_t row = 0; row < 8; row ++){

            //calculate the LED values we need to send to the display
and store them in R1, G1, B1, R2, G2, B2
            calcColorData(display, row, R1, G1, B1, R2, G2, B2);

            //iterate through every column, shifting in the color data
to the shift registers
            for(uint8_t col = 0; col < 32; col++){

                //turn the clock off
                digitalWrite(MATRIX_CLK_GPIO, MATRIX_CLK_PIN, GPIO_LOW);

                    //set all the data pins to the values (using
inefficient method)

                    //row ADDR values
                    digitalWrite(MATRIX_R1_GPIO, MATRIX_R1_PIN,
R1[col]);
                    digitalWrite(MATRIX_G1_GPIO, MATRIX_G1_PIN,
G1[col]);
                    digitalWrite(MATRIX_B1_GPIO, MATRIX_B1_PIN,
B1[col]);

                    //row ADDR + 8 values
                    digitalWrite(MATRIX_R2_GPIO, MATRIX_R2_PIN,
R2[col]);
                    digitalWrite(MATRIX_G2_GPIO, MATRIX_G2_PIN,
G2[col]);
                    digitalWrite(MATRIX_B2_GPIO, MATRIX_B2_PIN,
B2[col]);
```

```
                    //turn the clock on to shift the values into the
shift register
                    digitalWrite(MATRIX_CLK_GPIO, MATRIX_CLK_PIN,
GPIO_HIGH);

                }

                //assert blank
                digitalWrite(MATRIX_BLANK_GPIO, MATRIX_BLANK_PIN,
GPIO_HIGH);

                //set the address that should currently be displayed
                setAddr(addr);

                //increment the address to the next one to be displayed
                addr = (addr + 1) % 8;

                //assert then deassert latch
                digitalWrite(MATRIX_LAT_GPIO, MATRIX_LAT_PIN,
GPIO_HIGH);
                digitalWrite(MATRIX_LAT_GPIO, MATRIX_LAT_PIN, GPIO_LOW);

                //delay for an arbitrary period
                delay_micros(DELAY_TIM, 1250); //lowers brightness and
prevents bleeding

                //deassert blank
                digitalWrite(MATRIX_BLANK_GPIO, MATRIX_BLANK_PIN,
GPIO_LOW);

            }
        }

    void convertButtons2NewCol(uint8_t newCol[16], uint8_t
buttons_to_press){
        //Function: convertButtons2NewCol
        //author: Jimmy Fernandez (jpfernandez@g.hmc.edu) and Kathryn
Chan (klchan@g.hmc.edu)
```

```
        //Purpose: Take in an 8 bit integer indicating what button
presses should appear on the display
        //and convert this data into an array that contains the properly
formatted color data for this column
        //Inputs:
            //newCol - a uint8_t array 16 elements long that contains
color data for the new column to display on the
                //next board refresh
            //buttons_to_press - a uint8_t that describes the button
presses to be done on the next display refresh

        //retrieve the values of each button to set
        uint8_t red_val = (buttons_to_press >> 4) & 1;
        uint8_t yellow_val = (buttons_to_press >> 3) & 1;
        uint8_t black_val = (buttons_to_press >> 2) & 1;
        uint8_t green_val = (buttons_to_press >> 1) & 1;
        uint8_t blue_val = (buttons_to_press >> 0) & 1;

        //set the red col
        newCol[14] = R*red_val;
        newCol[13] = R*red_val;

        //set the yellow col
        newCol[11] = Y*yellow_val;
        newCol[10] = Y*yellow_val;

        //set the "black" col
        newCol[8] = W*black_val;
        newCol[7] = W*black_val;

        //set the green col
        newCol[5] = G*green_val;
        newCol[4] = G*green_val;

        //set the blue col
        newCol[2] = B*blue_val;
        newCol[1] = B*blue_val;
    }
```

D. Timer Library Modifications

Minor additions were added to the timer library to create a duration timer that could be used to update the game display. The updated header file for this library is shown below.

```
// STM32F401RE_TIM.h
// Header for TIM functions

#ifndef STM32F4_TIM_H
#define STM32F4_TIM_H

#include <stdint.h> // Include stdint header

//////////////////////////////////////////////////////////////////////////////
/////
// Definitions
//////////////////////////////////////////////////////////////////////////////
/////
// which timers correspond to what
#define NOTE_TIM TIM2
#define DELAY_TIM TIM5
#define DUR_TIM TIM3

///////


#define __IO volatile

uint32_t SystemCoreClock;     // Updated by configureClock()
#define HSE_VALUE 8000000     // Value of external input to OSC from
ST-LINK


// timer counting
#define DIR_UPCOUNT 0
#define DIR_DOWNCOUNT 1

#define PERIPH_BASE           0x40000000UL /*!< Peripheral base address in
the alias region                           */
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x00010000UL)
```

```
#define TIM1_BASE             (APB2PERIPH_BASE + 0x0000UL)
#define TIM2_BASE             (APB1PERIPH_BASE + 0x0000UL)
#define TIM3_BASE             (APB1PERIPH_BASE + 0x0400UL)
#define TIM4_BASE             (APB1PERIPH_BASE + 0x0800UL)
#define TIM5_BASE             (APB1PERIPH_BASE + 0x0C00UL)
#define TIM9_BASE             (APB2PERIPH_BASE + 0x4000UL)
#define TIM10_BASE            (APB2PERIPH_BASE + 0x4400UL)
#define TIM11_BASE            (APB2PERIPH_BASE + 0x4800UL)

/////////////////////////////////////////////////////////////////////////////
/////
// Bitfield structs
/////////////////////////////////////////////////////////////////////////////
/////


typedef struct{
    volatile uint32_t CEN   :1;
    volatile uint32_t UDIS  :1;
    volatile uint32_t URS   :1;
    volatile uint32_t OPM   :1;
    volatile uint32_t DIR   :1;
    volatile uint32_t CMS   :2;
    volatile uint32_t ARPE  :1;
    volatile uint32_t CKD   :2;
    volatile uint32_t       :22;
} CR1_bits;

typedef struct{
    volatile uint32_t SMS   :3;
    volatile uint32_t       :1;
    volatile uint32_t TS    :3;
    volatile uint32_t MSM   :1;
    volatile uint32_t ETF   :4;
    volatile uint32_t ETPS  :2;
    volatile uint32_t ECE   :1;
    volatile uint32_t ETP   :1;
    volatile uint32_t       :16;
} SMCR_bits;
```

```
typedef struct{
    volatile uint32_t UIF   :1;
    volatile uint32_t CC1IF :1;
    volatile uint32_t CC2IF :1;
    volatile uint32_t CC3IF :1;
    volatile uint32_t CC4IF :1;
    volatile uint32_t COMIF :1;
    volatile uint32_t TIF   :1;
    volatile uint32_t BIF   :1;
    volatile uint32_t       :1;
    volatile uint32_t CC1OF :1;
    volatile uint32_t CC2OF :1;
    volatile uint32_t CC3OF :1;
    volatile uint32_t CC4OF :1;
    volatile uint32_t       :19;
} SR_bits;

typedef struct{
    volatile uint32_t UG    :1;
    volatile uint32_t CC1G  :1;
    volatile uint32_t CC2G  :1;
    volatile uint32_t CC3G  :1;
    volatile uint32_t CC4G  :1;
    volatile uint32_t COMG  :1;
    volatile uint32_t TG    :1;
    volatile uint32_t BG    :1;
    volatile uint32_t       :24;
} EGR_bits;

typedef struct{
    volatile uint32_t CC1S  :2;
    volatile uint32_t OC1FE :1;
    volatile uint32_t OC1PE :1;
    volatile uint32_t OC1M  :3;
    volatile uint32_t OC1CE :1;
    volatile uint32_t CC2S  :2;
    volatile uint32_t OC2FE :1;
    volatile uint32_t OC2PE :1;
    volatile uint32_t OC2M  :3;
    volatile uint32_t OC2CE :1;
```

```
    volatile uint32_t       :16;
} CCMR_bits;

typedef struct{
    volatile uint32_t CC1E  :1;
    volatile uint32_t CC1P  :1;
    volatile uint32_t       :1;
    volatile uint32_t CC1NP :1;
    volatile uint32_t CC2E  :1;
    volatile uint32_t CC2P  :1;
    volatile uint32_t       :1;
    volatile uint32_t CC2NP :1;
    volatile uint32_t CC3E  :1;
    volatile uint32_t CC3P  :1;
    volatile uint32_t       :1;
    volatile uint32_t CC3NP :1;
    volatile uint32_t CC4E  :1;
    volatile uint32_t CC4P  :1;
    volatile uint32_t       :1;
    volatile uint32_t CC4NP :1;
} CCER_bits;

typedef struct {
    volatile uint32_t all      :32;
} ARR_bits;

typedef struct {
  __IO CR1_bits CR1;
  __IO uint32_t CR2;
  __IO SMCR_bits SMCR;
  __IO uint32_t DIER;
  __IO SR_bits SR;
  __IO EGR_bits EGR;
  __IO CCMR_bits CCMR1;
  __IO CCMR_bits CCMR2;
  __IO CCER_bits CCER;
  __IO uint32_t CNT;
  __IO uint32_t PSC;
  __IO uint32_t ARR;
  __IO uint32_t RCR;
```

```
  __IO uint32_t CCR1;
  __IO uint32_t CCR2;
  __IO uint32_t CCR3;
  __IO uint32_t CCR4;
  __IO uint32_t BDTR;
  __IO uint32_t DCR;
  __IO uint32_t DMAR;
} TIM_TypeDef;

// Pointers to appropriately-sized chunks of memory for each peripheral
#define TIM1  ((TIM_TypeDef *) TIM1_BASE)
#define TIM2  ((TIM_TypeDef *) TIM2_BASE)
#define TIM3  ((TIM_TypeDef *) TIM3_BASE)
#define TIM4  ((TIM_TypeDef *) TIM4_BASE)
#define TIM5  ((TIM_TypeDef *) TIM5_BASE)
#define TIM9  ((TIM_TypeDef *) TIM9_BASE)
#define TIM10 ((TIM_TypeDef *) TIM10_BASE)
#define TIM11 ((TIM_TypeDef *) TIM11_BASE)

////////////////////////////////////////////////////////////////////////////////
/////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////
/////

void initTIM(TIM_TypeDef * TIMx);

void delay_millis(TIM_TypeDef * TIMx, uint32_t ms);
void delay_micros(TIM_TypeDef * TIMx, uint32_t us);

void setupPWM(TIM_TypeDef * TIMx);
void updatePWM(TIM_TypeDef * TIMx,int note_freq);

void initDurationTimer(TIM_TypeDef * TIMx);
void resetDurationTimer(TIM_TypeDef * TIMx);
double getElapsedTime(TIM_TypeDef * TIMx);

#endif
```

The updated timer library file is shown below.

```c
// STM32F401RE_TIM.c
// TIM functions

#include "STM32F401RE_SPI.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_GPIO.h"
#include "STM32F401RE_TIM.h"


void initTIM(TIM_TypeDef * TIMx){
  uint32_t psc_div = (uint32_t) ((SystemCoreClock/1e6)-1);

  // Set prescaler division factor
  TIMx->PSC = psc_div ;
  // Generate an update event to update prescaler value
  TIMx->EGR.UG = 1;
  // Enable counter
  TIMx->CR1.CEN = 1; // Set CEN = 1
}


void delay_millis(TIM_TypeDef * TIMx, uint32_t ms){
  TIMx->ARR = ms*1000;// Set timer max count
  TIMx->EGR.UG = 1;     // Force update
  TIMx->SR.UIF = 0; // Clear UIF
  TIMx->CNT = 0;      // Reset count

  while(!(TIMx->SR.UIF & 1)); // Wait for UIF to go high
}

void delay_micros(TIM_TypeDef * TIMx, uint32_t us){
  TIMx->ARR = us;// Set timer max count
  TIMx->EGR.UG = 1;     // Force update
  TIMx->SR.UIF = 0; // Clear UIF
  TIMx->CNT = 0;      // Reset count

  while(!(TIMx->SR.UIF & 1)); // Wait for UIF to go high
}
```

```
void setupPWM(TIM_TypeDef * TIMx) {
    initTIM(TIM2);

    // set to edge aligned mode
    TIMx->CR1.CMS = 0;

    // set to upcounting
    TIMx->CR1.DIR = DIR_UPCOUNT;

    // disable slave controller, internal clock
    TIMx->SMCR.SMS = 0;

    //  set to PWM mode 1
    TIMx->CCMR1.OC1M = 0b110;

    // enable preload register
    TIMx->CCMR1.OC1PE = 1;

    //auto-reload preload register
    TIMx->CR1.ARPE = 1;
    // I am note sure if I want this to happen?

    //set polarity of active low so signal is low then high
    TIMx->CCER.CC1P = 1;

    //signal is output to GPIOA PIN 0 (timer2 channel 1)
    TIMx->CCER.CC1E = 1;

    // generate a UEV, reset the counter
    TIMx->EGR.UG = 1;

    /* 110: PWM mode 1 - In upcounting, channel 1 is active as long as
        TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is
        inactive (OC1REF=‘0) as long as TIMx_CNT>TIMx_CCR1 else active
(OC1REF=1).
    */
}

void updatePWM(TIM_TypeDef * TIMx, int note_freq){
    // obtain a value to load into ARR
```

```
    if (note_freq == 0){
      TIMx->CCR1 = 0;
    }
    else{
    int period_micros = (int)((1.0f/note_freq*1000000)-1);  // period in
seconds
    int half_period_micros = period_micros/2;   //half the period in
microseconds

    //load half period in ARR
    TIMx->ARR = period_micros;
    TIMx->CCR1 = half_period_micros;
    }
    TIMx->EGR.UG = 1; // intiate UEV to update values

}


void initDurationTimer(TIM_TypeDef * TIMx){
  uint32_t psc_div = (uint32_t) ((SystemCoreClock/2e3)-1);

  // Set prescaler division factor
  TIMx->PSC = psc_div ;
  // Generate an update event to update prescaler value
  TIMx->EGR.UG = 1;
  // Enable counter
  TIMx->CR1.CEN = 1; // Set CEN = 1
}

void resetDurationTimer(TIM_TypeDef * TIMx){

  //diable the timer
  TIMx->CR1.CEN = 0;

  //genrate an update event on the duration timer
  TIMx->EGR.UG = 1;

  //re-enable duration timer
  TIMx->CR1.CEN = 1;
```

```
}

double getElapsedTime(TIM_TypeDef * TIMx){

  //function: getElapsedTime
  //author: Jimmy Fernandez (jpfernandez@g.hmc.edu) and Kathryn Chan
(klchan@g.hmc.edu)
  //purpose: returns the time passed since the last reset of the duration
timer in ms (as a double)

  double ms_elapsed = (double)TIMx->CNT*0.5;

  return ms_elapsed;

}
```

E. Game Screens Library

The following header file game_screens.h was used to define values related to predefined game screens.

```
#ifndef GAME_SCREENS_DEF
#define GAME_SCREENS_DEF

#include "LED_Matrix.h"

////////////////////////////////////////
////////////// Screen Variables //////////////
////////////////////////////////////////

#define HOME_SCREEN 0
#define SONG_SELECT_SCREEN 1
#define LEVEL_1 2
#define LEVEL_2 3
#define LEVEL_3 4
#define LEVEL_4 5
#define LEVEL_5 6
#define SCORE_SCREEN 7

////////////////////////////////////////
////////////// PROTOTYPES //////////////
////////////////////////////////////////

void updateScoreScreen(int score, uint8_t score_screen[32][16]);
/* updateScoreScreen maps an integer score out of 100 onto an array
so that it can be displayed on the LED matrix
it also adds a message depending on the bracket of performance the
user
score falls under*/

void clearScoreScreen(uint8_t score_screen[32][16]);
/* clearScoreScreen clears the spots in the array on score screen
where a number should be displayed */

void setTransposedScreen(uint8_t screenContainer[32][16], int
screenSelection);
/* */

void copyToScreen(uint8_t src[32][16], uint8_t dest[32][16]);
```

```
    /* copyToScreen takes in a src screen and copys it onto a
destination screen */
    #endif
```

The following library file game_screens.c was used to define functions to define and copy game screen data into a container for use in the main function.

```
    #include "game_screens.h"
    #include "LED_Matrix.h"

    void copyToScreen(uint8_t src[32][16], uint8_t dest[32][16]){

        for(int row = 0; row < 32; row++){
            for(int col = 0; col < 16; col++){

                dest[row][col] = src[row][col];
            }
        }
    }

    void setTransposedScreen(uint8_t screenContainer[32][16], int
screenSelection){

        if(screenSelection == HOME_SCREEN){

                uint8_t home_screen[32][16] = {
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, Y, 0, 0, 0, 0, Y, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, Y, 0, Y, Y, 0, Y, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, Y, Y, Y, Y, Y, Y, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, W, 0, 0, W, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, W, 0, 0, W, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, W, 0, 0, W, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, W, 0, 0, W, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, W, W, W, W, 0, 0, W, 0, W, W, W, W, W, 0},
```

```
        {0, 0, W, 0, 0, 0, W, 0, W, 0, 0, 0, W, 0, 0, 0},
        {0, 0, W, 0, 0, 0, W, 0, W, 0, 0, 0, W, 0, 0, 0},
        {0, 0, W, W, W, W, 0, 0, W, 0, 0, 0, W, 0, 0, 0},
        {0, 0, W, 0, 0, 0, W, 0, W, 0, 0, 0, W, 0, 0, 0},
        {0, 0, W, 0, 0, 0, W, 0, W, 0, 0, 0, W, 0, 0, 0},
        {0, 0, W, W, W, W, 0, 0, W, 0, 0, 0, W, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, W, 0, W, 0, W, W, 0, W, W, W, 0, W, W, W, 0},
        {0, W, 0, W, 0, W, 0, 0, W, 0, W, 0, W, 0, W, 0},
        {0, W, W, W, 0, W, W, 0, W, W, 0, 0, W, 0, W, 0},
        {0, W, 0, W, 0, W, 0, 0, W, 0, W, 0, W, 0, W, 0},
        {0, W, 0, W, 0, W, W, 0, W, 0, W, 0, W, W, W, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, R, R, 0, Y, Y, 0, W, W, 0, G, G, 0, B, B, 0},
        {0, R, R, 0, Y, Y, 0, W, W, 0, G, G, 0, B, B, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    };

            copyToScreen(home_screen, screenContainer);

    }


    else if (screenSelection == SONG_SELECT_SCREEN){


            uint8_t song_select_screen[32][16] = {
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, W, W, W, 0, W, 0, W, W, W, 0, W, 0, W, 0, 0},
        {0, W, 0, W, 0, W, 0, W, 0, W, 0, W, 0, W, 0, 0},
        {0, W, W, W, 0, W, 0, W, 0, 0, 0, W, W, 0, 0, 0},
        {0, W, 0, 0, 0, W, 0, W, 0, W, 0, W, 0, W, 0, 0},
        {0, W, 0, 0, 0, W, 0, W, W, W, 0, W, 0, W, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, W, W, W, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, W, 0, W, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, W, W, W, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, W, 0, W, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, W, 0, W, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, W, W, 0, W, W, W, 0, W, W, W, 0, W, W, W, 0},
    {0, W, 0, 0, W, 0, W, 0, W, 0, W, 0, W, 0, 0, 0},
    {0, W, W, 0, W, 0, W, 0, W, 0, W, 0, W, 0, W, 0},
    {0, 0, W, 0, W, 0, W, 0, W, 0, W, 0, W, 0, W, 0},
    {0, W, W, 0, W, W, W, 0, W, 0, W, 0, W, W, W, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, B, B, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, B, B, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, G, G, 0, B, B, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, G, G, 0, B, B, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, G, G, 0, B, B, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, G, G, 0, B, B, 0},
    {0, 0, 0, 0, Y, Y, 0, W, W, 0, G, G, 0, B, B, 0},
    {0, 0, 0, 0, Y, Y, 0, W, W, 0, G, G, 0, B, B, 0},
    {0, R, R, 0, Y, Y, 0, W, W, 0, G, G, 0, B, B, 0},
    {0, R, R, 0, Y, Y, 0, W, W, 0, G, G, 0, B, B, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};

        copyToScreen(song_select_screen, screenContainer);

    }

    else if (screenSelection == LEVEL_1){

        uint8_t level_1[32][16] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, R, 0, R, R, 0, R, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, R, R, R, R, R, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, R, R, R, R, R, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, 0, R, R, 0, R, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};

            copyToScreen(level_1, screenContainer);
    }

    else if (screenSelection == LEVEL_2){


            uint8_t level_2[32][16] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, Y, 0, Y, Y, 0, Y, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, Y, Y, Y, Y, Y, Y, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, W, W, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, Y, Y, Y, Y, Y, Y, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, Y, 0, Y, Y, 0, Y, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};


        copyToScreen(level_2, screenContainer);
    }

    else if (screenSelection == LEVEL_3){


        uint8_t level_3[32][16] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, 0, Y, G, 0, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, R, Y, G, B, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, W, W, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, W, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, W, W, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, R, Y, G, B, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, R, 0, Y, G, 0, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};

        copyToScreen(level_3, screenContainer);

    }

    else if (screenSelection == LEVEL_4){


        uint8_t level_4[32][16] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, G, 0, G, G, 0, G, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, G, G, G, G, G, G, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, G, G, G, G, G, G, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, G, 0, G, G, 0, G, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};

        copyToScreen(level_4, screenContainer);

    }

    else if (screenSelection == LEVEL_5){


        uint8_t level_5[32][16] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, B, 0, B, B, 0, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, B, B, B, B, B, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, 0, 0, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, W, W, W, W, W, W, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, B, B, B, B, B, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, B, 0, B, B, 0, B, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};


        copyToScreen(level_5, screenContainer);

    }

    else if (screenSelection == SCORE_SCREEN){
```

```
            uint8_t score_screen[32][16] = {
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, C, C, C, 0, C, C, C, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, C, C, C, 0, C, C, C, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, C, C, C, 0, C, C, C, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, C, C, C, 0, C, C, C, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, C, C, C, 0, C, C, C, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, W, W, W, W, W, W, W, W, W, W, W, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, W, 0, W, W, W, 0, W, W, W, 0, 0, 0},
     {0, 0, 0, 0, W, 0, W, 0, W, 0, W, 0, W, 0, 0, 0},
     {0, 0, 0, 0, W, 0, W, 0, W, 0, W, 0, W, 0, 0, 0},
     {0, 0, 0, 0, W, 0, W, 0, W, 0, W, 0, W, 0, 0, 0},
     {0, 0, 0, 0, W, 0, W, W, W, 0, W, W, W, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};
            copyToScreen(score_screen, screenContainer);

     }

     else {
```

```
            uint8_t blank_screen[32][16] = {0};

            copyToScreen(blank_screen, screenContainer);

        }


    }



    void clearScoreScreen(uint8_t score_screen[32][16]){
        score_screen[9][4] = 0;
        score_screen[9][5] = 0;
        score_screen[9][6] = 0;
        score_screen[9][7] = 0;
        score_screen[9][8] = 0;
        score_screen[9][9] = 0;
        score_screen[9][10] = 0;
        score_screen[9][11] = 0;
        score_screen[9][12] = 0;

        score_screen[10][4] = 0;
        score_screen[10][5] = 0;
        score_screen[10][6] = 0;
        score_screen[10][7] = 0;
        score_screen[10][8] = 0;
        score_screen[10][9] = 0;
        score_screen[10][10] = 0;
        score_screen[10][11] = 0;
        score_screen[10][12] = 0;

        score_screen[11][4] = 0;
        score_screen[11][5] = 0;
        score_screen[11][6] = 0;
        score_screen[11][7] = 0;
        score_screen[11][8] = 0;
        score_screen[11][9] = 0;
        score_screen[11][10] = 0;
        score_screen[11][11] = 0;
```

```
        score_screen[11][12] = 0;

        score_screen[12][4] = 0;
        score_screen[12][5] = 0;
        score_screen[12][6] = 0;
        score_screen[12][7] = 0;
        score_screen[12][8] = 0;
        score_screen[12][9] = 0;
        score_screen[12][10] = 0;
        score_screen[12][11] = 0;
        score_screen[12][12] = 0;

        score_screen[13][4] = 0;
        score_screen[13][5] = 0;
        score_screen[13][6] = 0;
        score_screen[13][7] = 0;
        score_screen[13][8] = 0;
        score_screen[13][9] = 0;
        score_screen[13][10] = 0;
        score_screen[13][11] = 0;
        score_screen[13][12] = 0;
    }

    void updateScoreScreen(int score, uint8_t score_screen[32][16]){
        uint8_t ones_digit = score%10;
        uint8_t tens_digit = (score-ones_digit)/10;

        if(score == 100){
            score_screen[9][4] = W;
            score_screen[9][5] = 0;
            score_screen[9][6] = W;
            score_screen[9][7] = W;
            score_screen[9][8] = W;
            score_screen[9][9] = 0;
            score_screen[9][10] = W;
            score_screen[9][11] = W;
            score_screen[9][12] = W;

            score_screen[10][4] = W;
            score_screen[10][5] = 0;
```

```
            score_screen[10][6] = W;
            score_screen[10][7] = 0;
            score_screen[10][8] = W;
            score_screen[10][9] = 0;
            score_screen[10][10] = W;
            score_screen[10][11] = 0;
            score_screen[10][12] = W;

            score_screen[11][4] = W;
            score_screen[11][5] = 0;
            score_screen[11][6] = W;
            score_screen[11][7] = 0;
            score_screen[11][8] = W;
            score_screen[11][9] = 0;
            score_screen[11][10] = W;
            score_screen[11][11] = 0;
            score_screen[11][12] = W;

            score_screen[12][4] = W;
            score_screen[12][5] = 0;
            score_screen[12][6] = W;
            score_screen[12][7] = 0;
            score_screen[12][8] = W;
            score_screen[12][9] = 0;
            score_screen[12][10] = W;
            score_screen[12][11] = 0;
            score_screen[12][12] = W;

            score_screen[13][4] = W;
            score_screen[13][5] = 0;
            score_screen[13][6] = W;
            score_screen[13][7] = W;
            score_screen[13][8] = W;
            score_screen[13][9] = 0;
            score_screen[13][10] = W;
            score_screen[13][11] = W;
            score_screen[13][12] = W;
        }

        else{
```

```
            switch(ones_digit){
                case 0:
                    score_screen[9][9] = W;
                    score_screen[9][10] = W;
                    score_screen[9][11] = W;

                    score_screen[10][9] = W;
                    score_screen[10][10] = 0;
                    score_screen[10][11] = W;

                    score_screen[11][9] = W;
                    score_screen[11][10] = 0;
                    score_screen[11][11] = W;

                    score_screen[12][9] = W;
                    score_screen[12][10] = 0;
                    score_screen[12][11] = W;

                    score_screen[13][9] = W;
                    score_screen[13][10] = W;
                    score_screen[13][11] = W;
                    break;
                case 1:
                    score_screen[9][9] = 0;
                    score_screen[9][10] = W;
                    score_screen[9][11] = 0;

                    score_screen[10][9] = W;
                    score_screen[10][10] = W;
                    score_screen[10][11] = 0;

                    score_screen[11][9] = 0;
                    score_screen[11][10] = W;
                    score_screen[11][11] = 0;

                    score_screen[12][9] = 0;
                    score_screen[12][10] = W;
                    score_screen[12][11] = 0;

                    score_screen[13][9] = W;
```

```
                score_screen[13][10] = W;
                score_screen[13][11] = W;
                break;
            case 2:
                score_screen[9][9] = W;
                score_screen[9][10] = W;
                score_screen[9][11] = W;

                score_screen[10][9] = 0;
                score_screen[10][10] = 0;
                score_screen[10][11] = W;

                score_screen[11][9] = W;
                score_screen[11][10] = W;
                score_screen[11][11] = W;

                score_screen[12][9] = W;
                score_screen[12][10] = 0;
                score_screen[12][11] = 0;

                score_screen[13][9] = W;
                score_screen[13][10] = W;
                score_screen[13][11] = W;
                break;
            case 3:
                score_screen[9][9] = W;
                score_screen[9][10] = W;
                score_screen[9][11] = W;

                score_screen[10][9] = 0;
                score_screen[10][10] = 0;
                score_screen[10][11] = W;

                score_screen[11][9] = W;
                score_screen[11][10] = W;
                score_screen[11][11] = W;

                score_screen[12][9] = 0;
                score_screen[12][10] = 0;
                score_screen[12][11] = W;
```

```
            score_screen[13][9] = W;
            score_screen[13][10] = W;
            score_screen[13][11] = W;
            break;
        case 4:
            score_screen[9][9] = W;
            score_screen[9][10] = 0;
            score_screen[9][11] = W;

            score_screen[10][9] = W;
            score_screen[10][10] = 0;
            score_screen[10][11] = W;

            score_screen[11][9] = W;
            score_screen[11][10] = W;
            score_screen[11][11] = W;

            score_screen[12][9] = 0;
            score_screen[12][10] = 0;
            score_screen[12][11] = W;

            score_screen[13][9] = 0;
            score_screen[13][10] = 0;
            score_screen[13][11] = W;
            break;
        case 5:
            score_screen[9][9] = W;
            score_screen[9][10] = W;
            score_screen[9][11] = W;

            score_screen[10][9] = W;
            score_screen[10][10] = 0;
            score_screen[10][11] = 0;

            score_screen[11][9] = W;
            score_screen[11][10] = W;
            score_screen[11][11] = W;

            score_screen[12][9] = 0;
```

```
                score_screen[12][10] = 0;
                score_screen[12][11] = W;

                score_screen[13][9] = W;
                score_screen[13][10] = W;
                score_screen[13][11] = W;
                break;
            case 6:
                score_screen[9][9] = W;
                score_screen[9][10] = W;
                score_screen[9][11] = W;

                score_screen[10][9] = W;
                score_screen[10][10] = 0;
                score_screen[10][11] = 0;

                score_screen[11][9] = W;
                score_screen[11][10] = W;
                score_screen[11][11] = W;

                score_screen[12][9] = W;
                score_screen[12][10] = 0;
                score_screen[12][11] = W;

                score_screen[13][9] = W;
                score_screen[13][10] = W;
                score_screen[13][11] = W;
                break;
            case 7:
                score_screen[9][9] = W;
                score_screen[9][10] = W;
                score_screen[9][11] = W;

                score_screen[10][9] = 0;
                score_screen[10][10] = 0;
                score_screen[10][11] = W;

                score_screen[11][9] = 0;
                score_screen[11][10] = 0;
                score_screen[11][11] = W;
```

```
            score_screen[12][9] = 0;
            score_screen[12][10] = 0;
            score_screen[12][11] = W;

            score_screen[13][9] = 0;
            score_screen[13][10] = 0;
            score_screen[13][11] = W;
            break;
        case 8:
            score_screen[9][9] = W;
            score_screen[9][10] = W;
            score_screen[9][11] = W;

            score_screen[10][9] = W;
            score_screen[10][10] = 0;
            score_screen[10][11] = W;

            score_screen[11][9] = W;
            score_screen[11][10] = W;
            score_screen[11][11] = W;

            score_screen[12][9] = W;
            score_screen[12][10] = 0;
            score_screen[12][11] = W;

            score_screen[13][9] = W;
            score_screen[13][10] = W;
            score_screen[13][11] = W;
            break;
        case 9:
            score_screen[9][9] = W;
            score_screen[9][10] = W;
            score_screen[9][11] = W;

            score_screen[10][9] = W;
            score_screen[10][10] = 0;
            score_screen[10][11] = W;

            score_screen[11][9] = W;
```

```
                score_screen[11][10] = W;
                score_screen[11][11] = W;

                score_screen[12][9] = 0;
                score_screen[12][10] = 0;
                score_screen[12][11] = W;

                score_screen[13][9] = 0;
                score_screen[13][10] = 0;
                score_screen[13][11] = W;
                break;
            }

            switch(tens_digit){
                case 0:
                    score_screen[9][5] = W;
                    score_screen[9][6] = W;
                    score_screen[9][7] = W;

                    score_screen[10][5] = W;
                    score_screen[10][6] = 0;
                    score_screen[10][7] = W;

                    score_screen[11][5] = W;
                    score_screen[11][6] = 0;
                    score_screen[11][7] = W;

                    score_screen[12][5] = W;
                    score_screen[12][6] = 0;
                    score_screen[12][7] = W;

                    score_screen[13][5] = W;
                    score_screen[13][6] = W;
                    score_screen[13][7] = W;
                    break;
                case 1:
                    score_screen[9][5] = 0;
                    score_screen[9][6] = W;
                    score_screen[9][7] = 0;
```

```
                score_screen[10][5] = W;
                score_screen[10][6] = W;
                score_screen[10][7] = 0;

                score_screen[11][5] = 0;
                score_screen[11][6] = W;
                score_screen[11][7] = 0;

                score_screen[12][5] = 0;
                score_screen[12][6] = W;
                score_screen[12][7] = 0;

                score_screen[13][5] = W;
                score_screen[13][6] = W;
                score_screen[13][7] = W;
                break;
            case 2:
                score_screen[9][5] = W;
                score_screen[9][6] = W;
                score_screen[9][7] = W;

                score_screen[10][5] = 0;
                score_screen[10][6] = 0;
                score_screen[10][7] = W;

                score_screen[11][5] = W;
                score_screen[11][6] = W;
                score_screen[11][7] = W;

                score_screen[12][5] = W;
                score_screen[12][6] = 0;
                score_screen[12][7] = 0;

                score_screen[13][5] = W;
                score_screen[13][6] = W;
                score_screen[13][7] = W;
                break;
            case 3:
                score_screen[9][5] = W;
                score_screen[9][6] = W;
```

```
                score_screen[9][7] = W;

                score_screen[10][5] = 0;
                score_screen[10][6] = 0;
                score_screen[10][7] = W;

                score_screen[11][5] = W;
                score_screen[11][6] = W;
                score_screen[11][7] = W;

                score_screen[12][5] = 0;
                score_screen[12][6] = 0;
                score_screen[12][7] = W;

                score_screen[13][5] = W;
                score_screen[13][6] = W;
                score_screen[13][7] = W;
                break;
            case 4:
                score_screen[9][5] = W;
                score_screen[9][6] = 0;
                score_screen[9][7] = W;

                score_screen[10][5] = W;
                score_screen[10][6] = 0;
                score_screen[10][7] = W;

                score_screen[11][5] = W;
                score_screen[11][6] = W;
                score_screen[11][7] = W;

                score_screen[12][5] = 0;
                score_screen[12][6] = 0;
                score_screen[12][7] = W;

                score_screen[13][5] = 0;
                score_screen[13][6] = 0;
                score_screen[13][7] = W;
                break;
            case 5:
```

```
                score_screen[9][5] = W;
                score_screen[9][6] = W;
                score_screen[9][7] = W;

                score_screen[10][5] = W;
                score_screen[10][6] = 0;
                score_screen[10][7] = 0;

                score_screen[11][5] = W;
                score_screen[11][6] = W;
                score_screen[11][7] = W;

                score_screen[12][5] = 0;
                score_screen[12][6] = 0;
                score_screen[12][7] = W;

                score_screen[13][5] = W;
                score_screen[13][6] = W;
                score_screen[13][7] = W;
                break;
            case 6:
                score_screen[9][5] = W;
                score_screen[9][6] = W;
                score_screen[9][7] = W;

                score_screen[10][5] = W;
                score_screen[10][6] = 0;
                score_screen[10][7] = 0;

                score_screen[11][5] = W;
                score_screen[11][6] = W;
                score_screen[11][7] = W;

                score_screen[12][5] = W;
                score_screen[12][6] = 0;
                score_screen[12][7] = W;

                score_screen[13][5] = W;
                score_screen[13][6] = W;
                score_screen[13][7] = W;
```

```
            break;
        case 7:
            score_screen[9][5] = W;
            score_screen[9][6] = W;
            score_screen[9][7] = W;

            score_screen[10][5] = 0;
            score_screen[10][6] = 0;
            score_screen[10][7] = W;

            score_screen[11][5] = 0;
            score_screen[11][6] = 0;
            score_screen[11][7] = W;

            score_screen[12][5] = 0;
            score_screen[12][6] = 0;
            score_screen[12][7] = W;

            score_screen[13][5] = 0;
            score_screen[13][6] = 0;
            score_screen[13][7] = W;
            break;
        case 8:
            score_screen[9][5] = W;
            score_screen[9][6] = W;
            score_screen[9][7] = W;

            score_screen[10][5] = W;
            score_screen[10][6] = 0;
            score_screen[10][7] = W;

            score_screen[11][5] = W;
            score_screen[11][6] = W;
            score_screen[11][7] = W;

            score_screen[12][5] = W;
            score_screen[12][6] = 0;
            score_screen[12][7] = W;

            score_screen[13][5] = W;
```

```
                score_screen[13][6] = W;
                score_screen[13][7] = W;
                break;
            case 9:
                score_screen[9][5] = W;
                score_screen[9][6] = W;
                score_screen[9][7] = W;

                score_screen[10][5] = W;
                score_screen[10][6] = 0;
                score_screen[10][7] = W;

                score_screen[11][5] = W;
                score_screen[11][6] = W;
                score_screen[11][7] = W;

                score_screen[12][5] = 0;
                score_screen[12][6] = 0;
                score_screen[12][7] = W;

                score_screen[13][5] = 0;
                score_screen[13][6] = 0;
                score_screen[13][7] = W;
                break;
            }
        }
        uint16_t TOP = 0;
        uint16_t LEFT = 0;
        if(score==100){
        // YEE
            //Y
            TOP = 2;
            LEFT = 3;
            score_screen[TOP][LEFT] = R;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = R;

            score_screen[TOP+1][LEFT] = R;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = R;
```

```
        score_screen[TOP+2][LEFT] = R;
        score_screen[TOP+2][LEFT+1] = R;
        score_screen[TOP+2][LEFT+2] = R;

        score_screen[TOP+3][LEFT] = 0;
        score_screen[TOP+3][LEFT+1] = R;
        score_screen[TOP+3][LEFT+2] = 0;

        score_screen[TOP+4][LEFT] = 0;
        score_screen[TOP+4][LEFT+1] = R;
        score_screen[TOP+4][LEFT+2] = 0;

        //E
        TOP = 2;
        LEFT = 7;
        score_screen[TOP][LEFT] = Y;
        score_screen[TOP][LEFT+1] = Y;
        score_screen[TOP][LEFT+2] = Y;

        score_screen[TOP+1][LEFT] = Y;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = 0;

        score_screen[TOP+2][LEFT] = Y;
        score_screen[TOP+2][LEFT+1] = Y;
        score_screen[TOP+2][LEFT+2] = Y;

        score_screen[TOP+3][LEFT] = Y;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = 0;

        score_screen[TOP+4][LEFT] = Y;
        score_screen[TOP+4][LEFT+1] = Y;
        score_screen[TOP+4][LEFT+2] = Y;

        // E
        TOP = 2;
        LEFT = 11;
        score_screen[TOP][LEFT] = G;
```

```
        score_screen[TOP][LEFT+1] = G;
        score_screen[TOP][LEFT+2] = G;

        score_screen[TOP+1][LEFT] = G;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = 0;

        score_screen[TOP+2][LEFT] = G;
        score_screen[TOP+2][LEFT+1] = G;
        score_screen[TOP+2][LEFT+2] = G;

        score_screen[TOP+3][LEFT] = G;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = 0;

        score_screen[TOP+4][LEFT] = G;
        score_screen[TOP+4][LEFT+1] = G;
        score_screen[TOP+4][LEFT+2] = G;

    //HAW
        //H
        TOP = 24;
        LEFT = 2;
        score_screen[TOP][LEFT] = C;
        score_screen[TOP][LEFT+1] = 0;
        score_screen[TOP][LEFT+2] = C;

        score_screen[TOP+1][LEFT] = C;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = C;

        score_screen[TOP+2][LEFT] = C;
        score_screen[TOP+2][LEFT+1] = C;
        score_screen[TOP+2][LEFT+2] = C;

        score_screen[TOP+3][LEFT] = C;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = C;

        score_screen[TOP+4][LEFT] = C;
```

```
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = C;

            //A
            TOP = 24;
            LEFT = 6;
            score_screen[TOP][LEFT] = B;
            score_screen[TOP][LEFT+1] = B;
            score_screen[TOP][LEFT+2] = B;

            score_screen[TOP+1][LEFT] = B;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = B;

            score_screen[TOP+2][LEFT] = B;
            score_screen[TOP+2][LEFT+1] = B;
            score_screen[TOP+2][LEFT+2] = B;

            score_screen[TOP+3][LEFT] = B;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = B;

            score_screen[TOP+4][LEFT] = B;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = B;

            //W
            TOP = 24;
            LEFT = 10;
            score_screen[TOP][LEFT] = P;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = 0;
            score_screen[TOP][LEFT+3] = 0;
            score_screen[TOP][LEFT+4] = P;

            score_screen[TOP+1][LEFT] = P;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = 0;
            score_screen[TOP+1][LEFT+3] = 0;
            score_screen[TOP+1][LEFT+4] = P;
```

```
        score_screen[TOP+2][LEFT] = P;
        score_screen[TOP+2][LEFT+1] = 0;
        score_screen[TOP+2][LEFT+2] = P;
        score_screen[TOP+2][LEFT+3] = 0;
        score_screen[TOP+2][LEFT+4] = P;

        score_screen[TOP+3][LEFT] = P;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = P;
        score_screen[TOP+3][LEFT+3] = 0;
        score_screen[TOP+3][LEFT+4] = P;

        score_screen[TOP+4][LEFT] = 0;
        score_screen[TOP+4][LEFT+1] = P;
        score_screen[TOP+4][LEFT+2] = 0;
        score_screen[TOP+4][LEFT+3] = P;
        score_screen[TOP+4][LEFT+4] = 0;

    }
    else if(score < 100 && score > 69){
    // NICE
        // N
        TOP = 2;
        LEFT = 2;
        score_screen[TOP][LEFT] = G;
        score_screen[TOP][LEFT+1] = G;
        score_screen[TOP][LEFT+2] = G;

        score_screen[TOP+1][LEFT] = G;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = G;

        score_screen[TOP+2][LEFT] = G;
        score_screen[TOP+2][LEFT+1] = 0;
        score_screen[TOP+2][LEFT+2] = G;

        score_screen[TOP+3][LEFT] = G;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = G;
```

```
        score_screen[TOP+4][LEFT] = G;
        score_screen[TOP+4][LEFT+1] = 0;
        score_screen[TOP+4][LEFT+2] = G;

        //I
        TOP = 2;
        LEFT = 6;
        score_screen[TOP][LEFT] = G;
        score_screen[TOP+1][LEFT] = G;
        score_screen[TOP+2][LEFT] = G;
        score_screen[TOP+3][LEFT] = G;
        score_screen[TOP+4][LEFT] = G;

        //C
        TOP = 2;
        LEFT = 8;
        score_screen[TOP][LEFT] = G;
        score_screen[TOP][LEFT+1] = G;
        score_screen[TOP][LEFT+2] = G;

        score_screen[TOP+1][LEFT] = G;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = G;

        score_screen[TOP+2][LEFT] = G;
        score_screen[TOP+2][LEFT+1] = 0;
        score_screen[TOP+2][LEFT+2] = 0;

        score_screen[TOP+3][LEFT] = G;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = G;

        score_screen[TOP+4][LEFT] = G;
        score_screen[TOP+4][LEFT+1] = G;
        score_screen[TOP+4][LEFT+2] = G;

        //E
        TOP = 2;
        LEFT = 12;
```

```
        score_screen[TOP][LEFT] = G;
        score_screen[TOP][LEFT+1] = G;
        score_screen[TOP][LEFT+2] = G;

        score_screen[TOP+1][LEFT] = G;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = 0;

        score_screen[TOP+2][LEFT] = G;
        score_screen[TOP+2][LEFT+1] = G;
        score_screen[TOP+2][LEFT+2] = G;

        score_screen[TOP+3][LEFT] = G;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = 0;

        score_screen[TOP+4][LEFT] = G;
        score_screen[TOP+4][LEFT+1] = G;
        score_screen[TOP+4][LEFT+2] = G;

    // JOB
        //J
        TOP = 24;
        LEFT = 3;
        score_screen[TOP][LEFT] = G;
        score_screen[TOP][LEFT+1] = G;
        score_screen[TOP][LEFT+2] = G;

        score_screen[TOP+1][LEFT] = 0;
        score_screen[TOP+1][LEFT+1] = G;
        score_screen[TOP+1][LEFT+2] = 0;

        score_screen[TOP+2][LEFT] = 0;
        score_screen[TOP+2][LEFT+1] = G;
        score_screen[TOP+2][LEFT+2] = 0;

        score_screen[TOP+3][LEFT] = 0;
        score_screen[TOP+3][LEFT+1] = G;
        score_screen[TOP+3][LEFT+2] = 0;
```

```
            score_screen[TOP+4][LEFT] = G;
            score_screen[TOP+4][LEFT+1] = G;
            score_screen[TOP+4][LEFT+2] = 0;

            //O
            TOP = 24;
            LEFT = 7;
            score_screen[TOP][LEFT] = G;
            score_screen[TOP][LEFT+1] = G;
            score_screen[TOP][LEFT+2] = G;

            score_screen[TOP+1][LEFT] = G;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = G;

            score_screen[TOP+2][LEFT] = G;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = G;

            score_screen[TOP+3][LEFT] = G;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = G;

            score_screen[TOP+4][LEFT] = G;
            score_screen[TOP+4][LEFT+1] = G;
            score_screen[TOP+4][LEFT+2] = G;

            // B
            TOP = 24;
            LEFT = 11;
            score_screen[TOP][LEFT] = G;
            score_screen[TOP][LEFT+1] = G;
            score_screen[TOP][LEFT+2] = G;

            score_screen[TOP+1][LEFT] = G;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = G;

            score_screen[TOP+2][LEFT] = G;
            score_screen[TOP+2][LEFT+1] = G;
```

```
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = G;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = G;

            score_screen[TOP+4][LEFT] = G;
            score_screen[TOP+4][LEFT+1] = G;
            score_screen[TOP+4][LEFT+2] = G;
        }
        else if(score == 69){
        // NICE
            // N
            TOP = 24;
            LEFT = 2;
            score_screen[TOP][LEFT] = P;
            score_screen[TOP][LEFT+1] = P;
            score_screen[TOP][LEFT+2] = P;

            score_screen[TOP+1][LEFT] = P;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = P;

            score_screen[TOP+2][LEFT] = P;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = P;

            score_screen[TOP+3][LEFT] = P;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = P;

            score_screen[TOP+4][LEFT] = P;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = P;

            //I
            TOP = 24;
            LEFT = 6;
            score_screen[TOP][LEFT] = P;
            score_screen[TOP+1][LEFT] = P;
```

```
            score_screen[TOP+2][LEFT] = P;
            score_screen[TOP+3][LEFT] = P;
            score_screen[TOP+4][LEFT] = P;

            //C
            TOP = 24;
            LEFT = 8;
            score_screen[TOP][LEFT] = P;
            score_screen[TOP][LEFT+1] = P;
            score_screen[TOP][LEFT+2] = P;

            score_screen[TOP+1][LEFT] = P;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = P;

            score_screen[TOP+2][LEFT] = P;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = P;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = P;

            score_screen[TOP+4][LEFT] = P;
            score_screen[TOP+4][LEFT+1] = P;
            score_screen[TOP+4][LEFT+2] = P;

            //E
            TOP = 24;
            LEFT = 12;
            score_screen[TOP][LEFT] = P;
            score_screen[TOP][LEFT+1] = P;
            score_screen[TOP][LEFT+2] = P;

            score_screen[TOP+1][LEFT] = P;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = 0;

            score_screen[TOP+2][LEFT] = P;
            score_screen[TOP+2][LEFT+1] = P;
```

```
            score_screen[TOP+2][LEFT+2] = P;

            score_screen[TOP+3][LEFT] = P;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = P;
            score_screen[TOP+4][LEFT+1] = P;
            score_screen[TOP+4][LEFT+2] = P;


        }
        else if(score <= 68 && score > 50){
        //PASS
            //P
            TOP = 2;
            LEFT = 2;
            score_screen[TOP][LEFT] = C;
            score_screen[TOP][LEFT+1] = C;
            score_screen[TOP][LEFT+2] = C;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = C;

            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = C;
            score_screen[TOP+2][LEFT+2] = C;

            score_screen[TOP+3][LEFT] = C;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = 0;

            //A
            TOP = 2;
            LEFT = 6;
```

```
            score_screen[TOP][LEFT] = C;
            score_screen[TOP][LEFT+1] = C;
            score_screen[TOP][LEFT+2] = C;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = C;

            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = C;
            score_screen[TOP+2][LEFT+2] = C;

            score_screen[TOP+3][LEFT] = C;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = C;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = C;
            // S
            TOP = 2;
            LEFT = 10;
            score_screen[TOP][LEFT] = C;
            score_screen[TOP][LEFT+1] = C;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;

            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = C;

            score_screen[TOP+3][LEFT] = 0;
            score_screen[TOP+3][LEFT+1] = C;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = C;
            // S
            TOP = 2;
            LEFT = 13;
            score_screen[TOP][LEFT] = C;
```

```
            score_screen[TOP][LEFT+1] = C;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;

            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = C;

            score_screen[TOP+3][LEFT] = 0;
            score_screen[TOP+3][LEFT+1] = C;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = C;

        //FAIL
            //F
            TOP = 24;
            LEFT = 2;
            score_screen[TOP][LEFT] = C;
            score_screen[TOP][LEFT+1] = C;
            score_screen[TOP][LEFT+2] = C;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = 0;

            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = C;
            score_screen[TOP+2][LEFT+2] = C;

            score_screen[TOP+3][LEFT] = C;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = 0;

            //A
            TOP = 24;
```

```
            LEFT = 6;
            score_screen[TOP][LEFT] = C;
            score_screen[TOP][LEFT+1] = C;
            score_screen[TOP][LEFT+2] = C;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = C;

            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = C;
            score_screen[TOP+2][LEFT+2] = C;

            score_screen[TOP+3][LEFT] = C;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = C;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = C;
            // I
            TOP = 24;
            LEFT = 10;
            score_screen[TOP][LEFT] = C;
            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+3][LEFT] = C;
            score_screen[TOP+4][LEFT] = C;

            // L
            TOP = 24;
            LEFT = 12;
            score_screen[TOP][LEFT] = C;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = 0;

            score_screen[TOP+1][LEFT] = C;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = 0;
```

```
            score_screen[TOP+2][LEFT] = C;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = C;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = C;
            score_screen[TOP+4][LEFT+1] = C;
            score_screen[TOP+4][LEFT+2] = C;
        }
        else if(score <= 50 && score > 10){
        // NICE
            // N
            TOP = 2;
            LEFT = 2;
            score_screen[TOP][LEFT] = Y;
            score_screen[TOP][LEFT+1] = Y;
            score_screen[TOP][LEFT+2] = Y;

            score_screen[TOP+1][LEFT] = Y;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = Y;

            score_screen[TOP+2][LEFT] = Y;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = Y;

            score_screen[TOP+3][LEFT] = Y;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = Y;

            score_screen[TOP+4][LEFT] = Y;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = Y;

            //I
            TOP = 2;
            LEFT = 6;
```

```
            score_screen[TOP][LEFT] = Y;
            score_screen[TOP+1][LEFT] = Y;
            score_screen[TOP+2][LEFT] = Y;
            score_screen[TOP+3][LEFT] = Y;
            score_screen[TOP+4][LEFT] = Y;

            //C
            TOP = 2;
            LEFT = 8;
            score_screen[TOP][LEFT] = Y;
            score_screen[TOP][LEFT+1] = Y;
            score_screen[TOP][LEFT+2] = Y;

            score_screen[TOP+1][LEFT] = Y;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = Y;

            score_screen[TOP+2][LEFT] = Y;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = Y;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = Y;

            score_screen[TOP+4][LEFT] = Y;
            score_screen[TOP+4][LEFT+1] = Y;
            score_screen[TOP+4][LEFT+2] = Y;

            //E
            TOP = 2;
            LEFT = 12;
            score_screen[TOP][LEFT] = Y;
            score_screen[TOP][LEFT+1] = Y;
            score_screen[TOP][LEFT+2] = Y;

            score_screen[TOP+1][LEFT] = Y;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = 0;
```

```
            score_screen[TOP+2][LEFT] = Y;
            score_screen[TOP+2][LEFT+1] = Y;
            score_screen[TOP+2][LEFT+2] = Y;

            score_screen[TOP+3][LEFT] = Y;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = Y;
            score_screen[TOP+4][LEFT+1] = Y;
            score_screen[TOP+4][LEFT+2] = Y;

        // TRY
            //T
            TOP = 24;
            LEFT = 3;
            score_screen[TOP][LEFT] = Y;
            score_screen[TOP][LEFT+1] = Y;
            score_screen[TOP][LEFT+2] = Y;

            score_screen[TOP+1][LEFT] = 0;
            score_screen[TOP+1][LEFT+1] = Y;
            score_screen[TOP+1][LEFT+2] = 0;

            score_screen[TOP+2][LEFT] = 0;
            score_screen[TOP+2][LEFT+1] = Y;
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = 0;
            score_screen[TOP+3][LEFT+1] = Y;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = 0;
            score_screen[TOP+4][LEFT+1] = Y;
            score_screen[TOP+4][LEFT+2] = 0;

            //R
            TOP = 24;
            LEFT = 7;
            score_screen[TOP][LEFT] = Y;
```

```
            score_screen[TOP][LEFT+1] = Y;
            score_screen[TOP][LEFT+2] = Y;

            score_screen[TOP+1][LEFT] = Y;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = Y;

            score_screen[TOP+2][LEFT] = Y;
            score_screen[TOP+2][LEFT+1] = Y;
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = Y;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = Y;

            score_screen[TOP+4][LEFT] = Y;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = Y;

            // Y
            TOP = 24;
            LEFT = 11;
            score_screen[TOP][LEFT] = Y;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = Y;

            score_screen[TOP+1][LEFT] = Y;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = Y;

            score_screen[TOP+2][LEFT] = Y;
            score_screen[TOP+2][LEFT+1] = Y;
            score_screen[TOP+2][LEFT+2] = Y;

            score_screen[TOP+3][LEFT] = 0;
            score_screen[TOP+3][LEFT+1] = Y;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = 0;
            score_screen[TOP+4][LEFT+1] = Y;
```

```
            score_screen[TOP+4][LEFT+2] = 0;
        }

        else { //(score <= 10)
        // YOU
            //Y
            TOP = 2;
            LEFT = 3;
            score_screen[TOP][LEFT] = R;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = R;

            score_screen[TOP+1][LEFT] = R;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = R;

            score_screen[TOP+2][LEFT] = R;
            score_screen[TOP+2][LEFT+1] = R;
            score_screen[TOP+2][LEFT+2] = R;

            score_screen[TOP+3][LEFT] = 0;
            score_screen[TOP+3][LEFT+1] = R;
            score_screen[TOP+3][LEFT+2] = 0;

            score_screen[TOP+4][LEFT] = 0;
            score_screen[TOP+4][LEFT+1] = R;
            score_screen[TOP+4][LEFT+2] = 0;

            //O
            TOP = 2;
            LEFT = 7;
            score_screen[TOP][LEFT] = R;
            score_screen[TOP][LEFT+1] = R;
            score_screen[TOP][LEFT+2] = R;

            score_screen[TOP+1][LEFT] = R;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = R;

            score_screen[TOP+2][LEFT] = R;
```

```
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = R;

            score_screen[TOP+3][LEFT] = R;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = R;

            score_screen[TOP+4][LEFT] = R;
            score_screen[TOP+4][LEFT+1] = R;
            score_screen[TOP+4][LEFT+2] = R;

            // U
            TOP = 2;
            LEFT = 11;
            score_screen[TOP][LEFT] = R;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = R;

            score_screen[TOP+1][LEFT] = R;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = R;

            score_screen[TOP+2][LEFT] = R;
            score_screen[TOP+2][LEFT+1] = 0;
            score_screen[TOP+2][LEFT+2] = R;

            score_screen[TOP+3][LEFT] = R;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = R;

            score_screen[TOP+4][LEFT] = R;
            score_screen[TOP+4][LEFT+1] = R;
            score_screen[TOP+4][LEFT+2] = R;

        // SUCK
            // S
            TOP = 24;
            LEFT = 2;
            score_screen[TOP][LEFT] = R;
            score_screen[TOP][LEFT+1] = R;
```

```
        score_screen[TOP+1][LEFT] = R;
        score_screen[TOP+1][LEFT+1] = 0;

        score_screen[TOP+2][LEFT] = R;
        score_screen[TOP+2][LEFT+1] = R;

        score_screen[TOP+3][LEFT] = 0;
        score_screen[TOP+3][LEFT+1] = R;

        score_screen[TOP+4][LEFT] = R;
        score_screen[TOP+4][LEFT+1] = R;

        // U
        TOP = 24;
        LEFT = 5;
        score_screen[TOP][LEFT] = R;
        score_screen[TOP][LEFT+1] = 0;
        score_screen[TOP][LEFT+2] = R;

        score_screen[TOP+1][LEFT] = R;
        score_screen[TOP+1][LEFT+1] = 0;
        score_screen[TOP+1][LEFT+2] = R;

        score_screen[TOP+2][LEFT] = R;
        score_screen[TOP+2][LEFT+1] = 0;
        score_screen[TOP+2][LEFT+2] = R;

        score_screen[TOP+3][LEFT] = R;
        score_screen[TOP+3][LEFT+1] = 0;
        score_screen[TOP+3][LEFT+2] = R;

        score_screen[TOP+4][LEFT] = R;
        score_screen[TOP+4][LEFT+1] = R;
        score_screen[TOP+4][LEFT+2] = R;

        //C
        TOP = 24;
        LEFT = 9;
        score_screen[TOP][LEFT] = R;
```

```
            score_screen[TOP][LEFT+1] = R;

            score_screen[TOP+1][LEFT] = R;
            score_screen[TOP+1][LEFT+1] = 0;

            score_screen[TOP+2][LEFT] = R;
            score_screen[TOP+2][LEFT+1] = 0;

            score_screen[TOP+3][LEFT] = R;
            score_screen[TOP+3][LEFT+1] = 0;

            score_screen[TOP+4][LEFT] = R;
            score_screen[TOP+4][LEFT+1] = R;

            //K
            TOP = 24;
            LEFT = 12;
            score_screen[TOP][LEFT] = R;
            score_screen[TOP][LEFT+1] = 0;
            score_screen[TOP][LEFT+2] = R;

            score_screen[TOP+1][LEFT] = R;
            score_screen[TOP+1][LEFT+1] = 0;
            score_screen[TOP+1][LEFT+2] = R;

            score_screen[TOP+2][LEFT] = R;
            score_screen[TOP+2][LEFT+1] = R;
            score_screen[TOP+2][LEFT+2] = 0;

            score_screen[TOP+3][LEFT] = R;
            score_screen[TOP+3][LEFT+1] = 0;
            score_screen[TOP+3][LEFT+2] = R;

            score_screen[TOP+4][LEFT] = R;
            score_screen[TOP+4][LEFT+1] = 0;
            score_screen[TOP+4][LEFT+2] = R;

        }
    }
```

F. Notes Library

The following header file notes.h was used to define values related to song definitions (notes, durations, button presses, etc).

```
// notes.h
// Header for notes for song playing

#ifndef notes_H
#define notes_H

#include <stdint.h>
#include "STM32F401RE_TIM.h"

//////////////////////////////////////////////////////////////////////////////
// Definitions
//////////////////////////////////////////////////////////////////////////////

#define NOTE_GPIO GPIOA
#define NOTE_PIN 0    //PA5 or PA0 are the only channels for this

#define NOTE_TIM TIM2

// note durations
#define SIXTEENTH 63
#define EIGHTH 125
#define DOTTED_EIGHTH EIGHTH + SIXTEENTH
#define TRIPLET 84
#define TRIPLET2 167
#define QUARTER 250
#define HALF 500
#define WHOLE 1000

// note frequencies
#define C3 131
#define C3sharp 139
#define D3flat C3sharp
#define D3 146
#define D3sharp 156
```

```
#define E3flat D3sharp
#define E3 165
#define F3 178
#define F3sharp 185
#define G3flat F3sharp
#define G3 196
#define G3sharp 210
#define A3flat G3sharp
#define A3 220
#define A3sharp 233
#define B3flat A3sharp
#define B3 247
#define C4 C3*2
#define C4sharp C3sharp*2
#define D4flat C4sharp
#define D4 D3*2
#define D4sharp D3sharp*2
#define E4flat D4sharp
#define E4 E3*2
#define F4 F3*2
#define F4sharp F3sharp*2
#define G4flat F4sharp
#define G4 G3*2
#define G4sharp G3sharp*2
#define A4flat G4sharp
#define A4 A3*2
#define A4sharp A3sharp*2
#define B4flat A4sharp
#define B4 B3*2
#define C5 C4*2
#define C5sharp C4sharp*2
#define D5flat C5sharp
#define D5 D4*2
#define D5sharp D4sharp*2
#define E5flat D5sharp
#define E5 E4*2
#define F5 F4*2
#define F5sharp F4sharp*2
#define G5flat F5sharp
#define G5 G4*2
```

```
#define G5sharp G4sharp*2
#define A5flat G5sharp
#define A5 A4*2
#define A5sharp A4sharp*2
#define B5flat A5sharp
#define B5 B4*2
#define C6 C5*2
#define G6 G5*2
#define B6flat B5flat*2
#define END 0
#define SONG 0
#define REST 0

//Mando specific notes
#define G2natural 98 //G2 overlaps w/ RGB
#define F2 87
#define A2flat 104
#define B2flat 117
#define E6flat 1245
#define D6 1175

// take on me specific note
#define A2 110

////////////////////////////////
/////// button presses //////////////
////////////////////////////////

#define NONE    0b00000
#define Rb    0b10000
#define Yb 0b01000
#define Kb  0b00100
#define Gb  0b00010
#define Bb   0b00001

// TWINKlE TWINKLE NOTE MAP
#define C3_TT       Rb
#define G3_TT       Bb
#define A3_TT       Yb
#define F3_TT       Gb
```

```
#define E3_TT       Kb
#define D3_TT       Yb

// FINAL COUNTDOWN NOTE MAP
#define C5sharp_FC  Bb
#define B4_FC       Gb
#define F4sharp_FC  Kb
#define D5_FC       Yb
#define G4sharp_FC  Gb
#define A4_FC       Rb

// Mando Notes map

#define G3_M Rb
#define D4_M Yb
#define F4_M Kb
#define B3flat_M Gb
#define C4_M Bb
#define F5_M Gb + Bb
#define D5_M Rb
#define A5flat_M Bb
#define G5_M Yb
#define E5flat_M Kb
#define C5_M Rb + Yb
#define G6_M Gb
#define B6flat_M Kb + Gb
#define C6_M Bb
#define G2_M Gb
#define E4flat_M Kb
#define C3_M Bb
#define F2_M Kb
#define A2flat_M Yb
#define G4_M Kb + Rb
#define A3flat_M Yb
#define F3_M Kb
#define A4flat_M Kb
#define D5flat_M Gb
#define B5_M Kb + Gb
#define B4_M Gb
#define B4flat_M Yb
```

```
#define G4flat_M Kb
#define A4_M Bb
#define B5flat_M Kb + Yb
#define E5_M Kb
#define B2flat_M Yb
#define E6flat_M Bb
#define D6_M Gb

// TAKE ON ME NOTE MAP
#define F5sharp_TM  Yb
#define E5_TM       Bb
#define B5_TM       Rb
#define A5_TM       Gb
#define D5_TM       Kb
#define G5sharp_TM  Kb
#define B4_TM       Gb

#define E3_TM       Kb
#define F3sharp_TM  Rb
#define G3sharp_TM  Kb
#define A3_TM       Gb
#define B3_TM       Yb
#define C4sharp_TM  Rb
#define D4_TM       Bb
#define E4_TM       Kb
#define F4sharp_TM  Yb

#define A2long_TM       Yb + Kb
#define G3sharplong_TM  Rb + Bb
#define A3long_TM       Yb + Gb
#define E4long_TM       Kb + Bb
#define F4sharplong_TM  Rb + Gb
#define C4sharplong_TM  Rb
#define G4sharplong_TM  Rb + Kb
#define A4long_TM       Rb + Kb + Bb

#define C5sharp_TM      Yb
#define A4_TM           Kb
#define E5long_TM       Rb + Yb + Kb + Gb + Bb
```

```
// STAR TREK NOTE MAP
#define G5_ST       Bb
#define G4_ST       Rb
#define F5_ST       Kb
#define E5flat_ST   Yb
#define A5flat_ST   Yb + Kb + Gb
#define C5_ST       Gb
#define B4flat_ST   Bb
#define F4_ST       Rb
#define D5_ST       Kb
#define A5_ST       Rb
#define F5sharp_ST  Gb
#define A4_ST       Yb
#define B4_ST       Kb
#define D5sharp_ST  Rb
#define G5sharp_ST  Yb
#define E5_ST       Bb
#define D4_ST       Gb

////////////////////////////////
//  song arrays
////////////////////////////////

#define SONG_TEST 0
#define SONG_FINALCOUNTDOWN 1
#define SONG_FURELISE 2
#define SONG_TWINKLE_TWINKLE 3
#define SONG_MANDO 4
#define SONG_TAKEONME 5
#define SONG_STARTREK 6


////////////////////////////////
//  song inf9
////////////////////////////////

//song arrays behave roughly like null terminated strings
#define MAX_SONG_LEN 1024
```

```
    ////////////////////////////////////////////////////////////////////////
//////////
    // Function prototypes
    ////////////////////////////////////////////////////////////////////////
//////////

    void getSongData(int songContainer[][3], int songSelection);
    /* */

    void copyToArray(int src[][3], int dest[][3]);
    /* */

    #endif
```

The following library file notes.c was used to define functions to define and copy song data into a container for use in the main function.

```
#include "notes.h"


void copyToArray(int src[][3], int dest[][3]){
    //Function: copyToArray
    //author: Jimmy Fernandez (jpfernandez@g.hmc.edu) and Kathryn Chan
(klchan@g.hmc.edu)
    //purpose: helper function to copy the predefined song array data into
the song container.
    //This function was written due to issues with defining the arrays in
a different header file
    //inputs:
        //src - an Nx3 int array that is the source of the song data
        //dest - a Kx3 int array that is the "song container" the game
controller uses to play the song

    int row = 0;
    uint8_t stop_cond = 0;
    do {

        dest[row][0] = src[row][0];
        dest[row][1] = src[row][1];
        dest[row][2] = src[row][2];
```

```
        stop_cond = src[row][1];
        row++;

    } while(stop_cond != 0);

}

void getSongData(int songContainer[][3], int songSelection){
    //Function: getSongData
    //author: Jimmy Fernandez (jpfernandez@g.hmc.edu) and Kathryn Chan
(klchan@g.hmc.edu)
    //purpose: helper function that copies the proper song data into the
song container based
    //off of the inputted song selection
    //Inputs:
        //songContainer - an Kx3 int array that stores the song data for
the game controller
        //songSelection - an int that indicates which song the user wants.
Recommended to use the
        //define statements from notes.h

    if (songSelection == SONG_TEST){

        int song_testarray[][3] = {
            {0,     QUARTER,    0b00000},
            {D4,    QUARTER,    0b00110},
            {F4,    QUARTER,    0b00101},
            {E4,     EIGHTH,    0b11000},
            {A3,       HALF,    0b10100},
            { 0,          0,          0}};

        copyToArray(song_testarray, songContainer);
        return; //unncessary if using else ifs
    }

    else if (songSelection == SONG_FINALCOUNTDOWN){
        int FinalCountdown_array[][3] = {
            {REST,      HALF,       NONE},
            {C5sharp,   EIGHTH,     C5sharp_FC},
            {B4,        EIGHTH,     B4_FC},
```

```
        {C5sharp,   HALF,       C5sharp_FC},
        {F4sharp,   3*QUARTER,  F4sharp_FC},
        {REST,      HALF,       NONE},

        {D5,        EIGHTH,     D5_FC},
        {C5sharp,   EIGHTH,     C5sharp_FC},
        {D5,        QUARTER,    D5_FC},
        {C5sharp,   QUARTER,    C5sharp_FC},
        {B4,        3*QUARTER,  B4_FC},
        {REST,      HALF,       NONE},

        {D5,        EIGHTH,     D5_FC},
        {C5sharp,   EIGHTH,     C5sharp_FC},
        {D5,        HALF,       D5_FC},
        {F4sharp,   3*QUARTER,  F4sharp_FC},
        {REST,      HALF,       NONE},

        {B4,        EIGHTH,     B4_FC},
        {A4,        EIGHTH,     A4_FC},
        {B4,        QUARTER,    B4_FC},
        {A4,        QUARTER,    A4_FC},
        {G4sharp,   QUARTER,    G4sharp_FC},
        {B4,        QUARTER,    B4_FC},
        {A4,        3*QUARTER,  A4_FC},
        {C5sharp,   EIGHTH,     C5sharp_FC},
        {B4,        EIGHTH,     B4_FC},
        {C5sharp,   HALF,       C5sharp_FC},
        {F4sharp,   3*QUARTER,  F4sharp_FC},
        {REST,      HALF,       NONE},

        {D5,        EIGHTH,     D5_FC},
        {C5sharp,   EIGHTH,     C5sharp_FC},
        {D5,        QUARTER,    D5_FC},
        {C5sharp,   QUARTER,    C5sharp_FC},
        {B4,        3*QUARTER,  B4_FC},
        {REST,      HALF,       NONE},

        {D5,        EIGHTH,     D5_FC},
        {C5sharp,   EIGHTH,     C5sharp_FC},
        {D5,        HALF,       D5_FC},
```

```
        {F4sharp,   3*QUARTER,  F4sharp_FC},
        {REST,      HALF,       NONE},

        {B4,        EIGHTH,     B4_FC},
        {A4,        EIGHTH,     A4_FC},
        {B4,        QUARTER,    B4_FC},
        {A4,        QUARTER,    A4_FC},
        {G4sharp,   QUARTER,    G4sharp_FC},
        {B4,        QUARTER,    B4_FC},
        {A4,        3*QUARTER,  A4_FC},
        {G4sharp,   EIGHTH,     G4sharp_FC},
        {A4,        EIGHTH,     A4_FC},
        {B4,        3*QUARTER,  B4_FC},
        {A4,        EIGHTH,     A4_FC},
        {B4,        EIGHTH,     B4_FC},
        {C5sharp,   QUARTER,    C5sharp_FC},
        {B4,        QUARTER,    B4_FC},
        {A4,        QUARTER,    A4_FC},
        {G4sharp,   QUARTER,    G4sharp_FC},
        {F4sharp,   HALF,       F4sharp_FC},
        {D5,        HALF,       D5_FC},
        {C5sharp,   WHOLE,      C5sharp_FC},
        {C5sharp,   3*EIGHTH,   C5sharp_FC},
        {D5,        3*EIGHTH,   D5_FC},
        {C5sharp,   EIGHTH,     C5sharp_FC},
        {B4,        EIGHTH,     B4_FC},
        {C5sharp,   2*WHOLE,    C5sharp_FC},
        {END,       SONG,       NONE}
        };

    copyToArray(FinalCountdown_array, songContainer);
    return; //unncessary if using else ifs
}

else if (songSelection == SONG_TWINKLE_TWINKLE){
    int TwinkleTwinkle[][3] = {
        {C3,    QUARTER,    C3_TT},
        {C3,    QUARTER,    C3_TT},
        {G3,    QUARTER,    G3_TT},
        {G3,    QUARTER,    G3_TT},
```

```
        {A3,    QUARTER,    A3_TT},
        {A3,    QUARTER,    A3_TT},
        {G3,    HALF,       G3_TT},

        {F3,    QUARTER,    F3_TT},
        {F3,    QUARTER,    F3_TT},
        {E3,    QUARTER,    E3_TT},
        {E3,    QUARTER,    E3_TT},
        {D3,    QUARTER,    D3_TT},
        {D3,    QUARTER,    D3_TT},
        {C3,    HALF,       C3_TT},

        {G3,    QUARTER,    G3_TT},
        {G3,    QUARTER,    G3_TT},
        {F3,    QUARTER,    F3_TT},
        {F3,    QUARTER,    F3_TT},
        {E3,    QUARTER,    E3_TT},
        {E3,    QUARTER,    E3_TT},
        {D3,    HALF,       D3_TT},

        {G3,    QUARTER,    E3_TT},
        {G3,    QUARTER,    G3_TT},
        {F3,    QUARTER,    F3_TT},
        {F3,    QUARTER,    F3_TT},
        {E3,    QUARTER,    E3_TT},
        {E3,    QUARTER,    E3_TT},
        {D3,    HALF,       D3_TT},

        {C3,    QUARTER,    C3_TT},
        {C3,    QUARTER,    C3_TT},
        {G3,    QUARTER,    G3_TT},
        {G3,    QUARTER,    G3_TT},
        {A3,    QUARTER,    A3_TT},
        {A3,    QUARTER,    A3_TT},
        {G3,    HALF,       G3_TT},

        {F3,    QUARTER,    F3_TT},
        {F3,    QUARTER,    F3_TT},
        {E3,    QUARTER,    E3_TT},
        {E3,    QUARTER,    E3_TT},
```

```
        {D3,    QUARTER,    D3_TT},
        {D3,    QUARTER,    D3_TT},
        {C3,    HALF,       C3_TT},
        {END,   SONG,       NONE}
        };

    copyToArray(TwinkleTwinkle, songContainer);
    return; //unncessary if using else ifs
}

else if (songSelection == SONG_MANDO){

    int Mando[][3] = {

        {REST, HALF, NONE}, //measure 1
        {REST, QUARTER, NONE},
        {REST, EIGHTH, NONE},
        {REST, SIXTEENTH, NONE},
        {G3, SIXTEENTH, G3_M},

        {D4, SIXTEENTH, D4_M}, //measure 2
        {G3, SIXTEENTH, G3_M},
        {D4, DOTTED_EIGHTH, D4_M},
        {G3, SIXTEENTH, G3_M},
        {D4, SIXTEENTH, D4_M},
        {G3, SIXTEENTH, G3_M},
        {D4, DOTTED_EIGHTH, D4_M},
        {G3, SIXTEENTH, G3_M},
        {D4, DOTTED_EIGHTH, D4_M},
        {G3, SIXTEENTH, G3_M},

        {D4, DOTTED_EIGHTH, D4_M}, //measure 3
        {G3, SIXTEENTH, G3_M},
        {D4, DOTTED_EIGHTH, D4_M},
        {G3, SIXTEENTH, G3_M},
        {D4, DOTTED_EIGHTH, D4_M},
        {G3, SIXTEENTH, G3_M},
        {D4, DOTTED_EIGHTH, D4_M},
        {G3, SIXTEENTH, G3_M},
```

```
            {F4, SIXTEENTH, F4_M}, //measure 4
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {F4, SIXTEENTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},

            {F4, DOTTED_EIGHTH, F4_M}, //measure 5
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},

            {B3flat, SIXTEENTH, B3flat_M}, //measure 6
            {G3, SIXTEENTH, G3_M},
            {B3flat, DOTTED_EIGHTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},
            {B3flat, SIXTEENTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},
            {B3flat, DOTTED_EIGHTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},
            {B3flat, DOTTED_EIGHTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},

            {B3flat, DOTTED_EIGHTH, B3flat_M}, //measure 7
            {G3, SIXTEENTH, G3_M},
            {B3flat, DOTTED_EIGHTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},
            {B3flat, DOTTED_EIGHTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},
            {B3flat, DOTTED_EIGHTH, B3flat_M},
            {G3, SIXTEENTH, G3_M},
```

```
            {C4, SIXTEENTH, C4_M}, //measure 8
            {G3, SIXTEENTH, G3_M},
            {C4, DOTTED_EIGHTH, C4_M},
            {G3, SIXTEENTH, G3_M},
            {C4, SIXTEENTH, C4_M},
            {G3, SIXTEENTH, G3_M},
            {C4, DOTTED_EIGHTH, C4_M},
            {G3, SIXTEENTH, G3_M},
            {C4, DOTTED_EIGHTH, C4_M},
            {G3, SIXTEENTH, G3_M},

            {C4, DOTTED_EIGHTH, C4_M}, //measure 9
            {G3, SIXTEENTH, G3_M},
            {C4, DOTTED_EIGHTH, C4_M},
            {G3, SIXTEENTH, G3_M},
            {C4, DOTTED_EIGHTH, C4_M},
            {G3, SIXTEENTH, G3_M},
            {C4, DOTTED_EIGHTH, C4_M},
            {REST, SIXTEENTH, NONE},

            {F5, SIXTEENTH, F5_M}, //measure 10
            {D5, SIXTEENTH, D5_M},
            {F5, DOTTED_EIGHTH, F5_M},
            {D5, SIXTEENTH, D5_M},
            {F5, SIXTEENTH, F5_M},
            {D5, SIXTEENTH, D5_M},
            {F5, DOTTED_EIGHTH, F5_M},
            {D5, SIXTEENTH, D5_M},
            {F5, DOTTED_EIGHTH, F5_M},
            {D5, SIXTEENTH, D5_M},

            {F5, DOTTED_EIGHTH, F5_M}, //measure 11
            {D5, SIXTEENTH, D5_M},
            {F5, DOTTED_EIGHTH, F5_M},
            {D5, SIXTEENTH, D5_M},
            {F5, DOTTED_EIGHTH, F5_M},
            {D5, SIXTEENTH, D5_M},
            {F5, DOTTED_EIGHTH, F5_M},
            {D5, SIXTEENTH, D5_M},
```

```
            {A5flat, SIXTEENTH, A5flat_M}, //measure 12
            {F5, SIXTEENTH, F5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {D5, SIXTEENTH, D5_M},
            {A5flat, SIXTEENTH, A5flat_M},
            {F5, SIXTEENTH, F5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {D5, SIXTEENTH, D5_M},
            {A5flat, DOTTED_EIGHTH, A5flat_M},
            {D5, SIXTEENTH, D5_M},

            {G5, DOTTED_EIGHTH, G5_M}, //measure 13
            {D5, SIXTEENTH, D5_M},
            {A5flat, DOTTED_EIGHTH, A5flat_M},
            {D5, SIXTEENTH, D5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {D5, SIXTEENTH, D5_M},
            {A5flat, DOTTED_EIGHTH, A5flat_M},
            {G5, SIXTEENTH, G5_M},

            {G5, SIXTEENTH, G5_M}, //measure 14 may need to modify to get
distinction
            {G5, SIXTEENTH, G5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {G5, SIXTEENTH, G5_M},
            {G5, SIXTEENTH, G5_M},
            {G5, SIXTEENTH, G5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {G5, SIXTEENTH, G5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {G5, SIXTEENTH, G5_M},

            {G5, DOTTED_EIGHTH, G5_M}, //measure 15
            {G5, SIXTEENTH, G5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {G5, SIXTEENTH, G5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {G5, SIXTEENTH, G5_M},
            {G5, DOTTED_EIGHTH, G5_M},
            {G5, SIXTEENTH, G5_M},
```

```
{E5flat, SIXTEENTH, E5flat_M}, //measure 16
{C5, SIXTEENTH, C5_M},
{D5, DOTTED_EIGHTH, D5_M},
{C5, SIXTEENTH, C5_M},
{E5flat, SIXTEENTH, E5flat_M},
{C5, SIXTEENTH, C5_M},
{D5, DOTTED_EIGHTH, D5_M},
{C5, SIXTEENTH, C5_M},
{E5flat, DOTTED_EIGHTH, E5flat_M},
{C5, SIXTEENTH, C5_M},

{D5, DOTTED_EIGHTH, D5_M}, //measure 17
{C5, SIXTEENTH, C5_M},
{E5flat, DOTTED_EIGHTH, E5flat_M},
{C5, SIXTEENTH, C5_M},
{D5, DOTTED_EIGHTH, D5_M},
{C5, SIXTEENTH, C5_M},
{E5flat, DOTTED_EIGHTH, E5flat_M},
{G5, SIXTEENTH, G5_M},

{C6, WHOLE + HALF + QUARTER, C6_M}, //measure 18-19
{REST, DOTTED_EIGHTH, NONE},
{G5, SIXTEENTH, G5_M},

{C6, WHOLE + HALF + QUARTER, C6_M}, //measure 20-21
{REST, DOTTED_EIGHTH, NONE},
{G2natural, SIXTEENTH, G2_M},
//{REST, QUARTER, NONE},

//{REST, QUARTER, NONE},
{C3, QUARTER, C3_M}, //measure 22
{E4flat, QUARTER, E4flat_M},
{D4, HALF, D4_M},
{REST, SIXTEENTH, NONE},
{F2, SIXTEENTH, F2_M},
{G2natural, SIXTEENTH, G2_M},

{A2flat, SIXTEENTH, A2flat_M}, //measure 23
{C4, SIXTEENTH, C4_M},
```

```
            {G4, DOTTED_EIGHTH, G4_M},
            {F4, SIXTEENTH, F4_M},
            {D4, SIXTEENTH + QUARTER + EIGHTH, D4_M},
            {REST, SIXTEENTH, NONE},
            {G2natural, SIXTEENTH, G2_M},

            {C3, QUARTER, C3_M}, //measure 24
            {E4flat, QUARTER, E4flat_M},
            {D4, HALF, D4_M},
            {REST, SIXTEENTH, NONE},
            {F2, SIXTEENTH, F2_M},
            {G2natural, SIXTEENTH, G2_M},

            {A2flat, SIXTEENTH, A2flat_M}, //measure 25
            {C4, SIXTEENTH, C4_M},
            {G4, DOTTED_EIGHTH, G4_M},
            {F4, SIXTEENTH, F4_M},
            {D4, HALF, D4_M},


            {REST, SIXTEENTH, NONE}, //measure 26-27
            {G3, SIXTEENTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, EIGHTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, EIGHTH, G3_M},
            {C4, SIXTEENTH, C4_M},
            {REST, SIXTEENTH, NONE},
            {B3flat, QUARTER + SIXTEENTH, B3flat_M},
            {REST, SIXTEENTH, NONE},
            {F3, SIXTEENTH, F3_M},
            {G3, SIXTEENTH, G3_M},
            {A3flat, DOTTED_EIGHTH, A3flat_M},
            {REST, SIXTEENTH, NONE},
            {B3flat, SIXTEENTH, B3flat_M},
            {A3flat, SIXTEENTH, A3flat_M},
            {G3, SIXTEENTH, G3_M},
            {F3, SIXTEENTH, F3_M},
            {REST, SIXTEENTH, NONE},
```

```
{REST, SIXTEENTH, NONE}, //measure 28-29
{G3, SIXTEENTH, G3_M},
{G3, SIXTEENTH, G3_M},
{G3, EIGHTH, G3_M},
{G3, SIXTEENTH, G3_M},
{G3, SIXTEENTH, G3_M},
{G3, EIGHTH, G3_M},
{C4, SIXTEENTH, C4_M},
{REST, SIXTEENTH, NONE},
{D4, QUARTER + SIXTEENTH, D4_M},
{REST, SIXTEENTH, NONE},
{B3flat, SIXTEENTH, B3flat_M},
{C4, SIXTEENTH, C4_M},
{D4, DOTTED_EIGHTH, D4_M},
{REST, SIXTEENTH, NONE},
{E4flat, SIXTEENTH, E4flat_M},
{D4, SIXTEENTH, D4_M},
{C4, SIXTEENTH, C3_M},
{B3flat, SIXTEENTH, B3flat_M},
{REST, SIXTEENTH, NONE},

//upper part
// {G4, SIXTEENTH, G4_M}, //measure 30-31
// {REST, SIXTEENTH, NONE},
// {F4, QUARTER, F4_M},
// {D4, SIXTEENTH, D4_M},
// {REST, SIXTEENTH, NONE},
// {C4, QUARTER, C4_M},
// {B3flat, SIXTEENTH, B3flat_M},
// {REST, SIXTEENTH, NONE},
// {G3, EIGHTH + QUARTER, G3_M},
// {REST, TRIPLET, NONE},
// {A4flat, TRIPLET, A4flat_M},
// {D5flat, TRIPLET, D5flat_M},
// {F5, TRIPLET, F5_M},
// {D5, TRIPLET, D5_M},
// {G5, TRIPLET, G5_M},
// {B5, QUARTER, B5_M},
```

```
            // middle part
            // {G4, SIXTEENTH, G4_M}, //measure 30-31
            // {REST, SIXTEENTH, NONE},
            // {F4, QUARTER, F4_M},
            // {D4, SIXTEENTH, D4_M},
            // {REST, SIXTEENTH, NONE},
            // {C4, QUARTER, C4_M},
            // {B3flat, SIXTEENTH, B3flat_M},
            // {REST, SIXTEENTH, NONE},
            // {G3, EIGHTH + QUARTER, G3_M},
            // {REST, TRIPLET, NONE},
            // {E4flat, TRIPLET, E4flat_M},
            // {A4flat, TRIPLET, A4flat_M},
            // {A4flat, TRIPLET, A4flat_M},
            // {B4, TRIPLET, B4_M},
            // {D5, TRIPLET, D5_M},
            // {D5, QUARTER, D5_M},

            //lower part
            {G4, SIXTEENTH, G4_M}, //measure 30-31
            {REST, SIXTEENTH, NONE},
            {F4, QUARTER, F4_M},
            {D4, SIXTEENTH, D4_M},
            {REST, SIXTEENTH, NONE},
            {C4, QUARTER, C4_M},
            {B3flat, SIXTEENTH, B3flat_M},
            {REST, SIXTEENTH, NONE},
            {G3, EIGHTH + QUARTER, G3_M},
            {REST, TRIPLET, NONE},
            {C4_M, TRIPLET, C4_M},
            {F4, TRIPLET, F4_M},
            {A4flat, TRIPLET, A4flat_M},
            {G4, TRIPLET, G4_M},
            {B4, TRIPLET, B4_M},
            {D5, QUARTER, D5_M},

            {C5, SIXTEENTH, C5_M}, //measure 32
            {G4, SIXTEENTH, G4_M},
            {D5, DOTTED_EIGHTH, D5_M},
            {G4, SIXTEENTH, G4_M},
```

```
            {D5, SIXTEENTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {C5, DOTTED_EIGHTH, C5_M},
            {G4, SIXTEENTH, G4_M},
            {D5, DOTTED_EIGHTH, D5_M},
            {G4, SIXTEENTH, G4_M},

            {C5, SIXTEENTH, C5_M}, //measure 33
            {G4, SIXTEENTH, G4_M},
            {D5, DOTTED_EIGHTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {D5, SIXTEENTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {C5, DOTTED_EIGHTH, C5_M},
            {G4, SIXTEENTH, G4_M},
            {C5, DOTTED_EIGHTH, C5_M},
            {G4, SIXTEENTH, G4_M},

            {C5, SIXTEENTH, C5_M}, //measure 34
            {G4, SIXTEENTH, G4_M},
            {D5, DOTTED_EIGHTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {D5, SIXTEENTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {C5, DOTTED_EIGHTH, C5_M},
            {G4, SIXTEENTH, G4_M},
            {D5, DOTTED_EIGHTH, D5_M},
            {G4, SIXTEENTH, G4_M},

            {C5, SIXTEENTH, C5_M}, //measure 35
            {G4, SIXTEENTH, G4_M},
            {D5, DOTTED_EIGHTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {D5, SIXTEENTH, D5_M},
            {G4, SIXTEENTH, G4_M},
            {C5, DOTTED_EIGHTH, C5_M},
            {G4, SIXTEENTH, G4_M},
            {C5, DOTTED_EIGHTH, C5_M},
            {G4, SIXTEENTH, G4_M},
```

```
            {REST, SIXTEENTH, NONE}, //measure 36-37
            {G3, SIXTEENTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, EIGHTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, EIGHTH, G3_M},
            {C4, SIXTEENTH, C4_M},
            {REST, SIXTEENTH, NONE},
            {B3flat, QUARTER + SIXTEENTH, B3flat_M},
            {REST, SIXTEENTH, NONE},
            {F3, SIXTEENTH, F3_M},
            {G3, SIXTEENTH, G3_M},
            {A3flat, DOTTED_EIGHTH, A3flat_M},
            {REST, SIXTEENTH, NONE},
            {B3flat, SIXTEENTH, B3flat_M},
            {A3flat, SIXTEENTH, A3flat_M},
            {G3, SIXTEENTH, G3_M},
            {F3, SIXTEENTH, F3_M},
            {REST, SIXTEENTH, NONE},

            {REST, SIXTEENTH, NONE}, //measure 38-39
            {G3, SIXTEENTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, EIGHTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, SIXTEENTH, G3_M},
            {G3, EIGHTH, G3_M},
            {C4, SIXTEENTH, C4_M},
            {REST, SIXTEENTH, NONE},
            {D4, QUARTER + SIXTEENTH, D4_M},
            {REST, SIXTEENTH, NONE},
            {B3flat, SIXTEENTH, B3flat_M},
            {C4, SIXTEENTH, C4_M},
            {D4, DOTTED_EIGHTH, D4_M},
            {REST, SIXTEENTH, NONE},
            {E4flat, SIXTEENTH, E4flat_M},
            {D4, SIXTEENTH, D4_M},
            {C4, SIXTEENTH, C3_M},
            {B3flat, SIXTEENTH, B3flat_M},
```

```
        {REST, SIXTEENTH, NONE},

        //lower part
        {G4, SIXTEENTH, G4_M}, //measure 40-41
        {REST, SIXTEENTH, NONE},
        {F4, QUARTER, F4_M},
        {D4, SIXTEENTH, D4_M},
        {REST, SIXTEENTH, NONE},
        {C4, QUARTER, C4_M},
        {B3flat, SIXTEENTH, B3flat_M},
        {REST, SIXTEENTH, NONE},
        {G3, EIGHTH + QUARTER, G3_M},
        {REST, TRIPLET, NONE},
        {C4_M, TRIPLET, C4_M},
        {F4, TRIPLET, F4_M},
        {A4flat, TRIPLET, A4flat_M},
        {G4, TRIPLET, G4_M},
        {B4, TRIPLET, B4_M},
        {D5, QUARTER, D5_M},

        {C5, HALF + QUARTER, C5_M}, //measure 42
        {REST, EIGHTH, NONE},
        {B4flat, SIXTEENTH, B4flat_M},
        {C5, SIXTEENTH, C5_M},

        {D5flat, DOTTED_EIGHTH, D5flat_M}, //measure 43
        {C5, SIXTEENTH, C5_M},
        {D5flat, DOTTED_EIGHTH, D5flat_M},
        {C5, SIXTEENTH, C5_M},
        {D5flat, DOTTED_EIGHTH, D5flat_M},
        {E5flat, SIXTEENTH, E5flat_M},
        {D5flat, SIXTEENTH, D5flat_M},
        {C5, SIXTEENTH, C5_M},
        {B4flat, SIXTEENTH, B4flat_M},

        {REST, EIGHTH, NONE}, //measure 44
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, EIGHTH, B4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
```

```
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, EIGHTH, B4flat_M},
        {A4flat, SIXTEENTH, A4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {B4, EIGHTH, B4_M},
        {G4flat, SIXTEENTH, G4flat_M},
        {E4flat, SIXTEENTH, E4flat_M},

        {F4, EIGHTH, F4_M}, //measure 45
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, EIGHTH, B4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {B4flat, EIGHTH, B4flat_M},
        {A4flat, SIXTEENTH, A4flat_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {C5, SIXTEENTH, C5_M},
        {REST, SIXTEENTH, NONE},
        {G4, SIXTEENTH, G4_M},
        {F4, SIXTEENTH, F4_M},

        {D5, SIXTEENTH, D5_M}, //measure 46
        {REST, SIXTEENTH, NONE},
        {A4flat, SIXTEENTH, A4flat_M},
        {G4, SIXTEENTH, G4_M},
        {E5flat, SIXTEENTH, E5flat_M},
        {REST, SIXTEENTH, NONE},
        {A5flat, SIXTEENTH, A5flat_M},
        {B5flat, SIXTEENTH, B5flat_M},

        // {A4, EIGHTH, A4_M}, //measure 47
        // {G4, SIXTEENTH, G4_M},
        // {F4, SIXTEENTH, F4_M},
        // {G4, HALF + EIGHTH, G4_M},
        // {F4, SIXTEENTH, F4_M},
        // {G4, SIXTEENTH, G4_M},

        {F5, EIGHTH, F5_M}, //measure 47
        {E5, SIXTEENTH, E5_M},
```

```
            {D5, SIXTEENTH, D5_M},
            {E5, HALF + EIGHTH, E5_M},
            {D5, SIXTEENTH, D5_M},
            {E5, SIXTEENTH, E5_M},

            {F5, EIGHTH, F5_M}, //measure 48
            {G5, SIXTEENTH, G5_M},
            {F5, SIXTEENTH, F5_M},
            {F5, EIGHTH, F5_M},
            {E5, HALF, E5_M},
            {D5, SIXTEENTH, D5_M},
            {E5, SIXTEENTH, E5_M},

            {F5, EIGHTH, F5_M}, //measure 49
            {E5, SIXTEENTH, E5_M},
            {D5, SIXTEENTH, D5_M},
            {D5, HALF + EIGHTH, D5_M},
            {C5, SIXTEENTH, C5_M},
            {D5, SIXTEENTH, D5_M},

            {C5, EIGHTH, C5_M}, //measure 50
            {B4flat, SIXTEENTH, B4flat_M},
            {A4flat, SIXTEENTH, A4flat_M},
            {G4, EIGHTH, G4_M},
            {A4flat, HALF, A4flat_M},
            {REST, EIGHTH, NONE},

            {F5, EIGHTH, F5_M}, //measure 51
            {E5, SIXTEENTH, E5_M},
            {D5, SIXTEENTH, D5_M},
            {E5, HALF + EIGHTH, E5_M},
            {D5, SIXTEENTH, D5_M},
            {E5, SIXTEENTH, E5_M},

            {F5, EIGHTH, F5_M}, //measure 52
            {G5, SIXTEENTH, G5_M},
            {F5, SIXTEENTH, F5_M},
            {F5, EIGHTH, F5_M},
            {E5, HALF, E5_M},
            {D5, SIXTEENTH, D5_M},
```

```
        {E5, SIXTEENTH, E5_M},

        {F5, EIGHTH, F5_M}, //measure 53
        {E5, SIXTEENTH, E5_M},
        {D5, SIXTEENTH, D5_M},
        {D5, HALF + EIGHTH, D5_M},
        {C5, SIXTEENTH, C5_M},
        {D5, SIXTEENTH, D5_M},

        {C5, EIGHTH, C5_M}, //measure 54
        {A4flat, SIXTEENTH, A4flat_M},
        {F4, SIXTEENTH, F4_M},
        {D5, EIGHTH, D5_M},
        {A4flat, SIXTEENTH, A4flat_M},
        {G4, SIXTEENTH, G4_M},
        {E5, EIGHTH, E5_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {C5, SIXTEENTH, C5_M},
        {F5, EIGHTH, F5_M},
        {B4flat, SIXTEENTH, B4flat_M},
        {C5, SIXTEENTH, C5_M},

        {G5, EIGHTH, G5_M}, //measure 55
        {C5, SIXTEENTH, C5_M},
        {D5, SIXTEENTH, D5_M},
        {A5flat, EIGHTH, A5flat_M},
        {D5, SIXTEENTH, D5_M},
        {A5flat, SIXTEENTH, A5flat_M},

        {C3, HALF + QUARTER, C3_M}, //measure 56
        {REST, EIGHTH, NONE},
        {F2, SIXTEENTH, F2_M},
        {G2natural, SIXTEENTH, G2_M},

        {A2flat, QUARTER + EIGHTH, A2flat_M}, //measure 57
        {REST, SIXTEENTH, NONE},
        {B2flat, SIXTEENTH, B2flat_M},
        {F2, QUARTER + EIGHTH, F2_M},
        {REST, SIXTEENTH, NONE},
        {G2natural, SIXTEENTH, G2_M},
```

```
{C3, HALF + QUARTER, C3_M}, //measure 58
{REST, EIGHTH, NONE},
{F2, SIXTEENTH, F2_M},
{G2natural, SIXTEENTH, G2_M},

{A2flat, QUARTER + EIGHTH, A2flat_M}, //measure 59
{REST, SIXTEENTH, NONE},
{B2flat, SIXTEENTH, B2flat_M},
{F2, QUARTER + EIGHTH, F2_M},
{REST, SIXTEENTH, NONE},
{C4, SIXTEENTH, C4_M},

{F4, SIXTEENTH, F4_M}, //measure 60
{G3, SIXTEENTH, G3_M},
{E4flat, DOTTED_EIGHTH, E4flat_M},
{G3, SIXTEENTH, G3_M},
{E4flat, SIXTEENTH, E4flat_M},
{G3, SIXTEENTH, G3_M},
{F4, DOTTED_EIGHTH, F4_M},
{G3, SIXTEENTH, G3_M},
{E4flat, DOTTED_EIGHTH, E4flat_M},
{G3, SIXTEENTH, G3_M},

{F4, SIXTEENTH, F4_M}, //measure 61
{G3, SIXTEENTH, G3_M},
{E4flat, DOTTED_EIGHTH, E4flat_M},
{G3, SIXTEENTH, G3_M},
{F4, SIXTEENTH, F4_M},
{G3, SIXTEENTH, G3_M},
{E4flat, DOTTED_EIGHTH, E4flat_M},
{G3, SIXTEENTH, G3_M},
{E4flat, DOTTED_EIGHTH, E4flat_M},
{G3, SIXTEENTH, G3_M},

{F4, SIXTEENTH, F4_M}, //measure 62
{G3, SIXTEENTH, G3_M},
{E4flat, DOTTED_EIGHTH, E4flat_M},
{G3, SIXTEENTH, G3_M},
{E4flat, SIXTEENTH, E4flat_M},
```

```
            {G3, SIXTEENTH, G3_M},
            {F4, DOTTED_EIGHTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {E4flat, DOTTED_EIGHTH, E4flat_M},
            {G3, SIXTEENTH, G3_M},

            {F4, SIXTEENTH, F4_M}, //measure 63
            {G3, SIXTEENTH, G3_M},
            {E4flat, DOTTED_EIGHTH, E4flat_M},
            {G3, SIXTEENTH, G3_M},
            {F4, SIXTEENTH, F4_M},
            {G3, SIXTEENTH, G3_M},
            {E4flat, DOTTED_EIGHTH, E4flat_M},
            {G3, SIXTEENTH, G3_M},
            {E4flat, DOTTED_EIGHTH, E4flat_M},
            {REST, SIXTEENTH, NONE},

            {G5*2, EIGHTH, G5_M}, //measure 65
            {C6, SIXTEENTH, C6_M},
            {E6flat, SIXTEENTH, E6flat_M},
            {F5*2, EIGHTH, F5_M},
            {D5*2, SIXTEENTH, D5_M},
            {E6flat, SIXTEENTH, E6flat_M},
            {G5*2, EIGHTH, G5_M},
            {C6, SIXTEENTH, C6_M},
            {E6flat, SIXTEENTH, E6flat_M},
            {F5*2, EIGHTH, F5_M},
            {D5*2, SIXTEENTH, D5_M},
            {E6flat, SIXTEENTH, E6flat_M},

            {G5*2, EIGHTH, G5_M}, //measure 66
            {C6, SIXTEENTH, C6_M},
            {E6flat, SIXTEENTH, E6flat_M},
            {F5*2, EIGHTH, F5_M},
            {D5*2, SIXTEENTH, D5_M},
            {E6flat, SIXTEENTH, E6flat_M},
            {G5*2, EIGHTH, G5_M},
            {C6, SIXTEENTH, C6_M},
            {E6flat, SIXTEENTH, E6flat_M},
            {F5*2, EIGHTH, F5_M},
```

```
        {D5*2, SIXTEENTH, D5_M},
        {E6flat, SIXTEENTH, E6flat_M},

        {G5*2, EIGHTH, G5_M}, //measure 67
        {C6, SIXTEENTH, C6_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {F5*2, EIGHTH, F5_M},
        {D5*2, SIXTEENTH, D5_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {G5*2, EIGHTH, G5_M},
        {C6, SIXTEENTH, C6_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {F5*2, EIGHTH, F5_M},
        {D5*2, SIXTEENTH, D5_M},
        {E6flat, SIXTEENTH, E6flat_M},

        {G5*2, EIGHTH, G5_M}, //measure 68
        {C6, SIXTEENTH, C6_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {F5*2, EIGHTH, F5_M},
        {D5*2, SIXTEENTH, D5_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {G5*2, EIGHTH, G5_M},
        {C6, SIXTEENTH, C6_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {F5*2, EIGHTH, F5_M},
        {D5*2, SIXTEENTH, D5_M},
        {E6flat, SIXTEENTH, E6flat_M},

        {G5*2, EIGHTH, G5_M}, //measure 69
        {C6, SIXTEENTH, C6_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {F5*2, EIGHTH, F5_M},
        {D5*2, SIXTEENTH, D5_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {G5*2, EIGHTH, G5_M},
        {C6, SIXTEENTH, C6_M},
        {E6flat, SIXTEENTH, E6flat_M},
        {F5*2, EIGHTH, F5_M},
        {D5*2, SIXTEENTH, D5_M},
```

```
            {E6flat, SIXTEENTH, E6flat_M},

            {G5*2, WHOLE, G5_M}, //measure 70
            {END, SONG, NONE}};

        copyToArray(Mando, songContainer);
        return; //unncessary if using else ifs

    }

    else if (songSelection == SONG_TAKEONME){

        int TakeOnMe_array[][3] = {
            {REST,      HALF,       NONE},          // measure 1
            {F5sharp,   EIGHTH,     F5sharp_TM},
            {F5sharp,   EIGHTH,     F5sharp_TM},
            {D5,        EIGHTH,     F5sharp_TM},
            {B4,        EIGHTH,     B4_TM},
            {REST,      EIGHTH,     NONE},
            {B4,        EIGHTH,     B4_TM},
            {REST,      EIGHTH,     NONE},
            {E5,        EIGHTH,     E5_TM},
            {REST,      EIGHTH,     NONE},
            {E5,        EIGHTH,     E5_TM},
            {REST,      EIGHTH,     NONE},
            {E5,        EIGHTH,     E5_TM},
            {G5sharp,   EIGHTH,     G5sharp_TM},
            {G5sharp,   EIGHTH,     G5sharp_TM},
            {A5,        EIGHTH,     B5_TM},
            {B5,        EIGHTH,     B5_TM},

            {A5,        EIGHTH,     A5_TM},         // measure 3
            {A5,        EIGHTH,     A5_TM},
            {A5,        EIGHTH,     A5_TM},
            {F5sharp,   EIGHTH,     F5sharp_TM},
            {REST,      EIGHTH,     NONE},
            {D5,        EIGHTH,     D5_TM},
            {REST,      EIGHTH,     NONE},
            {F5sharp,   EIGHTH,     F5sharp_TM},
            {REST,      EIGHTH,     NONE},
```

```
        {F5sharp,   EIGHTH,     F5sharp_TM},
        {REST,      EIGHTH,     NONE},
        {F5sharp,   EIGHTH,     F5sharp_TM},
        {E5,        EIGHTH,     E5_TM},
        {E5,        EIGHTH,     E5_TM},
        {E5,        EIGHTH,     E5_TM},
        {REST,      EIGHTH,     NONE},

        {D4,        QUARTER,    D4_TM},         // measure 5
        {REST,      EIGHTH,     NONE},
        {D4,        QUARTER,    D4_TM},
        {C4sharp,   EIGHTH,     C4sharp_TM},
        {B3,        QUARTER,    B3_TM},
        {REST,      WHOLE,      NONE},

        {C4sharp,   EIGHTH,     C4sharp_TM},    // measure 7
        {C4sharp,   EIGHTH,     C4sharp_TM},
        {REST,      EIGHTH,     NONE},
        {C4sharp,   QUARTER,    C4sharp_TM},
        {A3,        EIGHTH,     A3_TM},
        {REST,      QUARTER,    NONE},
        {REST,      EIGHTH,     NONE},
        {F4sharp,   EIGHTH,     F4sharp_TM},
        {REST,      EIGHTH,     NONE},
        {F4sharp,   EIGHTH,     F4sharp_TM},
        {F4sharp,   QUARTER,    F4sharp_TM},
        {E4,        QUARTER,    E4_TM},

        {D4,        QUARTER,    D4_TM},         // measure 9
        {REST,      EIGHTH,     NONE},
        {D4,        EIGHTH,     D4_TM},
        {D4,        EIGHTH,     D4_TM},
        {C4sharp,   QUARTER,    C4sharp_TM},
        {B3,        QUARTER,    B3_TM},
        {REST,      3*QUARTER,  NONE},

        {E3,        EIGHTH,     E3_TM},
        {C4sharp,   QUARTER,    C4sharp_TM},    // measure 11
        {D4,        EIGHTH,     D4_TM},
        {C4sharp,   QUARTER,    C4sharp_TM},
```

```
            {B3,        QUARTER,    B3_TM},
            {A3,        QUARTER,    A3_TM},
            {B3,        QUARTER,    B3_TM},
            {C4sharp,   EIGHTH,     C4sharp_TM},
            {B3,        QUARTER,    B3_TM},
            {A3,        QUARTER,    A3_TM},

            {REST,      QUARTER,    NONE},          // measure 13
            {D4,        QUARTER,    D4_TM},
            {D4,        QUARTER,    D4_TM},
            {D4,        EIGHTH,     D4_TM},
            {D4,        EIGHTH,     D4_TM},
            {REST,      WHOLE,      NONE},

            {REST,      QUARTER,    NONE},          // measure 15
            {F3sharp,   EIGHTH,     F3sharp_TM},
            {A3,        EIGHTH,     A3_TM},
            {A3,        EIGHTH,     A3_TM},
            {A3,        EIGHTH,     A3_TM},
            {A3,        EIGHTH,     A3_TM},
            {A3,        EIGHTH,     A3_TM},
            {A3,        EIGHTH,     A3_TM},
            {G3sharp,   QUARTER,    G3sharp_TM},
            {REST,      EIGHTH,     NONE},
            {G3sharp,   EIGHTH,     G3sharp_TM},
            {F3sharp,   QUARTER,    F3sharp_TM},
            {REST,      EIGHTH,     NONE},

            {A2,        WHOLE,      A2long_TM},
            {G3sharp,   WHOLE,      G3sharplong_TM},
            {A3,        WHOLE,      A3long_TM},

            {E4,        QUARTER,    E4_TM},
            {REST,      EIGHTH,     NONE},
            {F4sharp,   3*EIGHTH,   F4sharp_TM},
            {E4,        QUARTER,    E4_TM},

            {A3,        WHOLE,      A3long_TM},
            {E4,        WHOLE,      E4long_TM}, // maybe change if want
dual
```

```
        {F4sharp,   WHOLE,      F4sharplong_TM},

        {E4,        QUARTER,    E4_TM},
        {REST,      EIGHTH,     NONE},
        {F4sharp,   3*EIGHTH,   F4sharp_TM},
        {E4,        QUARTER,    E4_TM},

        {C4sharp,   WHOLE,      C4sharplong_TM},
        {G4sharp,   WHOLE,      G4sharplong_TM},
        {A4,        WHOLE,      A4long_TM},

        {REST,      QUARTER,    NONE},
        {B4,        EIGHTH,     B4_TM},
        {C5sharp,   EIGHTH,     C5sharp_TM},
        {REST,      EIGHTH,     NONE},
        {B4,        EIGHTH,     B4_TM},
        {A4,        HALF,       A4_TM},
        {E5,        13*QUARTER, E5long_TM},

        {REST,      HALF,       NONE},
        {END, SONG, NONE}};


    copyToArray(TakeOnMe_array, songContainer);
    return; //unncessary if using else ifs
}

else if (songSelection == SONG_STARTREK) {

    int StarTrek_array[][3] = {
        {REST,      HALF,       NONE},          // intro
        {F4,        3*EIGHTH,   F4_ST},
        {B4flat,    EIGHTH,     B4flat_ST},
        {E5flat,    3*QUARTER,  E5flat_ST},
        {D5,        QUARTER,    D5_ST},
        {B4flat,    TRIPLET2,   B4flat_ST},
        {G4,        TRIPLET2,   G4_ST},
        {C5,        TRIPLET2,   C5_ST},
        {F5,        3*QUARTER,  F5_ST},
        {REST,      EIGHTH,     NONE},
```

```
            {F5,        EIGHTH,     F5_ST},
            {A5flat,    WHOLE,      A5flat_ST},

            {F4,        QUARTER+TRIPLET,    F4_ST},     // measure 1 (A
section)
            {B4flat,    TRIPLET,            B4flat_ST},
            {D5,        TRIPLET,            B4flat_ST},
            {C5,        QUARTER+TRIPLET,    C5_ST},
            {A4flat,    TRIPLET,            G5_ST},
            {G5,        TRIPLET,            G5_ST},
            {F5,        HALF,               F5_ST},

            {REST,      TRIPLET2,           NONE},      // measure 4
            {E5flat,    TRIPLET,            F5_ST},
            {F5,        TRIPLET,            G5_ST},
            {G5,        TRIPLET,            G5_ST},
            {E5flat,    TRIPLET,            G5_ST},
            {F5,        QUARTER,            F5_ST},     // measure 5
            {E5flat,    QUARTER,            E5flat_ST},
            {D5,        QUARTER+TRIPLET,    D5_ST},
            {B4flat,    TRIPLET,            B4flat_ST},
            {D5,        TRIPLET,            B4flat_ST},
            {C5,        3*QUARTER,          C5_ST},
            {REST,      QUARTER,            NONE},

            {F4,        QUARTER+TRIPLET,    F4_ST},     // measure 9
(Repeat A section)
            {B4flat,    TRIPLET,            B4flat_ST},
            {D5,        TRIPLET,            B4flat_ST},
            {C5,        QUARTER+TRIPLET,    C5_ST},
            {A4flat,    TRIPLET,            G5_ST},
            {G5,        TRIPLET,            G5_ST},
            {F5,        HALF,               F5_ST},

            {REST,      TRIPLET2,           NONE},      // measure 12
            {E5flat,    TRIPLET,            F5_ST},
            {F5,        TRIPLET,            G5_ST},
            {G5,        TRIPLET,            G5_ST},
            {E5flat,    TRIPLET,            G5_ST},
            {F5,        QUARTER,            F5_ST},     // measure 13
```

```
        {E5flat,    QUARTER,            E5flat_ST},
        {D5,        QUARTER+TRIPLET,    D5_ST},
        {B4flat,    TRIPLET,            B4flat_ST},
        {D5,        TRIPLET,            B4flat_ST},
        {C5,        3*QUARTER,          C5_ST},
        {REST,      QUARTER,            NONE},

        {A5,        HALF,               A5_ST},     // (B section)
        {F5sharp,   QUARTER+TRIPLET,    F5sharp_ST},
        {D5,        TRIPLET,            D5_ST},
        {B5,        TRIPLET,            D5_ST},
        {A5,        HALF,               A5_ST},
        {F5sharp,   QUARTER,            F5sharp_ST},
        {REST,      QUARTER,            NONE},

        {A4,        QUARTER,            A4_ST},
        {B4,        QUARTER,            B4_ST},
        {C5,        QUARTER,            C5_ST},
        {B4,        QUARTER,            B4_ST},
        {A4,        TRIPLET,            A4_ST},
        {D4,        TRIPLET,            A4_ST},
        {A4,        TRIPLET,            A4_ST},
        {G4,        HALF,               G4_ST},
        {REST,      QUARTER,            NONE},

        {F4,        QUARTER+TRIPLET,    F4_ST},     // (A section)
        {B4flat,    TRIPLET,            B4flat_ST},
        {D5,        TRIPLET,            B4flat_ST},
        {C5,        QUARTER+TRIPLET,    C5_ST},
        {A4flat,    TRIPLET,            G5_ST},
        {G5,        TRIPLET,            G5_ST},
        {F5,        HALF,               F5_ST},

        {REST,      TRIPLET2,           NONE},
        {E5flat,    TRIPLET,            F5_ST},
        {F5,        TRIPLET,            G5_ST},
        {G5,        TRIPLET,            G5_ST},
        {E5flat,    TRIPLET,            G5_ST},
        {F5,        QUARTER,            F5_ST},
        {E5flat,    QUARTER,            E5flat_ST},
```

```
            {D5,        QUARTER+TRIPLET,    D5_ST},
            {B4flat,    TRIPLET,            B4flat_ST},
            {D5,        TRIPLET,            B4flat_ST},
            {C5,        3*QUARTER,          C5_ST},
            {REST,      QUARTER,            NONE},      // glissando
here??

            {F5sharp,   HALF,               F5sharp_ST},     // (B
section)
            {D5sharp,   QUARTER+TRIPLET,    D5sharp_ST},
            {B4,        TRIPLET,            B4_ST},
            {G5sharp,   TRIPLET,            B4_ST},
            {F5sharp,   HALF,               F5sharp_ST},
            {D5sharp,   QUARTER,            D5sharp_ST},
            {REST,      QUARTER,            NONE},

            {F5sharp,   QUARTER,            F5sharp_ST},
            {G5sharp,   QUARTER,            G5sharp_ST},
            {A5,        QUARTER,            A5_ST},
            {G5sharp,   QUARTER,            G5sharp_ST},
            {F5sharp,   TRIPLET,            F5sharp_ST},
            {B4,        TRIPLET,            F5sharp_ST},
            {F5sharp,   TRIPLET,            F5sharp_ST},
            {E5,        HALF,               E5_ST},
            {REST,      QUARTER,            NONE},

            {E5,        QUARTER+TRIPLET,    E5_ST},     // (B' section)
            {D5,        TRIPLET,            C5_ST},
            {C5,        TRIPLET,            C5_ST},
            {D5,        QUARTER+TRIPLET,    D5_ST},
            {A4,        TRIPLET,            A4_ST},
            {F5sharp,   TRIPLET,            A4_ST},
            {E5,        TRIPLET,            E5_ST},
            {REST,      TRIPLET,            NONE},
            {C5,        TRIPLET,            C5_ST},
            {D5,        HALF,               D5_ST},
            {REST,      QUARTER,            NONE},

            {E5,        QUARTER,            Rb},
            {F5sharp,   QUARTER,            Rb + Yb},
```

```
        {G5,        QUARTER,            Rb + Yb + Kb},
        {F5,        QUARTER,            Rb},
        {F5,        TRIPLET,            C5_ST},
        {E5,        TRIPLET,            C5_ST},
        {C5,        TRIPLET,            C5_ST},
        {D5,        QUARTER,            D5_ST},
        {REST,      EIGHTH,             NONE},
        {A4,        EIGHTH,             A4_ST},
        {D5,        EIGHTH,             D5_ST},
        {REST,      EIGHTH,             NONE},

        {D4,        QUARTER+TRIPLET,    D4_ST},     // (A' section)
        {G4,        TRIPLET,            G4_ST},
        {B4,        TRIPLET,            G4_ST},
        {A4,        QUARTER+TRIPLET,    A4_ST},
        {F4,        TRIPLET,            F4_ST},
        {E5,        TRIPLET,            F4_ST},
        {D5,        HALF,               D5_ST},

        {REST,      TRIPLET2,           NONE},
        {C5,        TRIPLET,            C5_ST},
        {D5,        TRIPLET,            E5_ST},
        {E5,        TRIPLET,            E5_ST},
        {C5,        TRIPLET,            E5_ST},
        {D5,        QUARTER,            D5_ST},
        {E5,        QUARTER,            E5_ST},
        {F5,        QUARTER,            F5_ST},
        {E5flat,    QUARTER,            E5flat_ST},

        {G5,        QUARTER,            G5_ST},
        {REST,      TRIPLET,            NONE},
        {G5,        TRIPLET/2,          G5_ST},
        {REST,      TRIPLET/2,          NONE},
        {G5,        TRIPLET/2,          G5_ST},
        {REST,      TRIPLET/2,          NONE},
        {G5,        TRIPLET,            G5_ST},
        {REST,      TRIPLET,            NONE},
        {F5,        TRIPLET2,           F5_ST},
        {E5flat,    TRIPLET2,           E5flat_ST},
        {G5,        TRIPLET,            G5_ST},
```

```
            {REST,      TRIPLET2,           NONE},
            {G4,        TRIPLET/2,          G4_ST},
            {REST,      TRIPLET/2,          NONE},
            {G4,        TRIPLET/2,          G4_ST},
            {REST,      TRIPLET/2,          NONE},
            {G4,        TRIPLET/2,          G4_ST},
            {REST,      TRIPLET/2,          NONE},
            {G4,        QUARTER,            G4_ST},

            {REST,      HALF,       NONE},
            {END, SONG, NONE}
        };

        copyToArray(StarTrek_array, songContainer);
        return; //unncessary if using else ifs
    }
}
```