Group Members:
Semih Kesler
Daniel Rodriguez
Michael Ruch

Project Name: Guitarduino Hero

Abstract: Our project uses three Arduino UNOs to create a competitive Guitar Hero like rhythm game for two players. Each player will get a controller consisting of four push buttons and an LED matrix. The input and output of these controllers are each handled by one Arduino board, and the third Arduino receives input from these controllers to calculate and display the player's scores. Players are scored based on how well they can synchronize button presses with the lights displayed on their LED matrix, and the player with the highest score at the end of the game wins.

Project Description:

1. Overall Description of Project Idea

The project will be a two player game inspired by the video game Guitar Hero. The players will battle syncing button presses with an LED panel. As some LEDs light up and are in the last row, the player will need to press that button to sync with it or miss. If a player has multiple hits without misses, he will earn a multiplier towards their score after each continuous hit. The player with the best score will win.

2. Final Project Design stating how Multiple Arduinos will be used

The project will use three Arduino boards. Two will be used as "controllers" for each player. At the beginning, they will handle the start signal from the "main" Arduino once both controllers have sent a ready signal to the "main". These two controller boards will handle output to the LED matrix panels. They will also read input as the players attempt to sync their button presses to the matrix output. The third Arduino will receive information from the two controllers and calculate and display each player's score on the LCD display and display the winner at the end of the game.

3. Final Plan for Use and Communication between the multiple Arduinos

In our final version, one arduino was used as the "computer" which sent and received input from both controllers. Once the "computer" arduino received both ready signals from the controllers, it sent a start signal back, which started the game for both controllers. Once the game was in progress, the controllers would each repeatedly send hit and miss bytes to the main Arduino, which it would then use to calculate the score. They all shared the same ground, but unlike in our original design, we needed to use the altSoftSerial library to use pins 8 and 9 on the main Arduino as additional RX/TX pins. The player 1 controller had its RX/TX pins wired to the main devices RX/TX pins, and the player 2 controller had its RX/TX pins wired to the main devices 8 and 9 pins.

4. Final Project Design stating Expected Inputs/Outputs

First, the controller will receive input from the ultrasonic sensor and will send ready A and B it will also output to LED respectively for player 1 and 2 which then outputs to the main arduino, then the main will output a start signal of X which will start the output of LED matrices. The expected inputs will be button presses which will send players A for hit and X for miss for player 1 and B for hit and Y for miss for player 2 it will also send an output for the buzzers for feedback then the output the hit/miss will be send to the score tracking/main Arduino. Which will output player scores and various messages on the LCD Display.

5. Final Description of the original work being attempted by your project.

The project required original work in both the coding and hardware setup. While other similar games exist, this specific implementation using multiple Arduino devices is original. With respect to the hardware, the original work came from our specific combination of components and how we used them to create this type of game. Usually this game is displayed on a big LCD screen with a fake guitar used for the controllers, but our implementation uses a LED Dot Matrix panel to display the needed button presses and a 16x2 LCD to display the score.

From a software perspective, although we took advantage of existing libraries for controlling the matrices and LCD display, our use of these libraries in the source code for both the controllers and main Arduino was original. For example, we used the text animations provided in the MD_Parola library for dot matrix displays in a novel way to create and animate note sprites. We created a custom font where each “character” in the font was actually a sprite corresponding to a note/button press combination in the game. Then, we used the existing scrolling animation from the library to simulate notes coming down the screen.

6. Discussion on how to build your project (think Lab Reports)

- Player 1 to Main - Serial: Connect Player 1 pin 0 to main pin 1 and pin1 to main pin 0

- Player 2 to Main - AltSoftSerial: Connect player 2 pin 0 to main pin 8 and player 2 pin 1 to main pin 9

- Player - Buttons: To wire the buttons you will need 10k ohm resistor connected to ground and power connect 5v on one side and another to pin 6,7,8,9 respectively

- Player - Ultrasonic Sensor: To wire the ultrasonic sensor, connect VCC to 5V, GND to ground, and trigger to a pin 2 and echo to pin 4

- Main - LCD Display: First wire the potentiometer upper left to ground and power 5V and output to G4 of LCD Display, GND to ground, VCC to 5V, RS to pin 12, RW to ground, E to pin 11, DB4 to pin 7, DB5 to pin 6, DB6 to pin 5, DB7 to pin 6, LED to 5V with 220 ohm resistor, LED to ground

- Player - Passive Buzzer: Connect to a pin 10 and connect to ground using 100 ohm resistor

- Player - LED Dot Matrix Panel: To wire the dot matrix led panel, first wire the matrix's VCC pin and ground to the Arduino's 5V power supply and the ground pins. Then wire the CS/Load to pin 3, DIN (data in) to pin 11, and CLK to pin 13.

- Player - LED: connect to ground with 220 ohm and to pin 5

7. Discussion on how to your project is to be used (think User Guide)

Once the Arduinos are set up, the controllers will wait for input from the ultrasonic sensor as a ready signal. To indicate that you are ready to begin, bring your hand to the ultrasonic sensor on the controller. Once the ultrasonic sensor has registered your hand, an LED on the controller breadboard will turn on to indicate that your controller has sent the ready signal to the main device. When both players have readied up, the LCD display on main will begin a countdown, after which the game will begin.

After the game starts, notes will begin coming down the matrix displays, and players can score by timing button presses with the notes. Each note is a square on the display corresponding to a button (the leftmost square corresponds to the leftmost button, and so on). The “strum line” on the matrix display marks the point when players must press a button for a hit. It is the line between the bottom two matrices (the overall matrix display consists of four 8x8 matrices daisy-chained together). Upon a successful hit, each controller’s buzzer will play a tone and the score on the LCD display will update. Each successful hit corresponds to 10 points, and players can earn multipliers for consecutive hits, which will reset after the first miss. The multipliers are: 2x for 10 hits, 3x for 20 hits, and 4x for 30+ hits. Each game consists of 50 randomly generated notes. The game ends once all 50 notes have gone by, and the LCD will then print the player who won.

Required Supporting Materials:

- Timeline:
    - Week 7 - Work on controller code
    - Week 8 - Finish controller code
    - Week 9 - Wire a player controller and LED panel for testing
    - Week 10 - Work on main arduino parser code
    - Week 11 - Finish Arduino parser code
    - Week 12 - Wire 2 controllers, LED, LCD and parser on main arduino for testing
    - Week 13 - Wire miscellaneous devices such as Buzzer, LED, etc.
    - Week 14 - Design Presentation on Monday 4/15/24 - Rest of week: Fix bugs and test edge cases
    - Week 15 - Demonstration on Monday 4/22/24

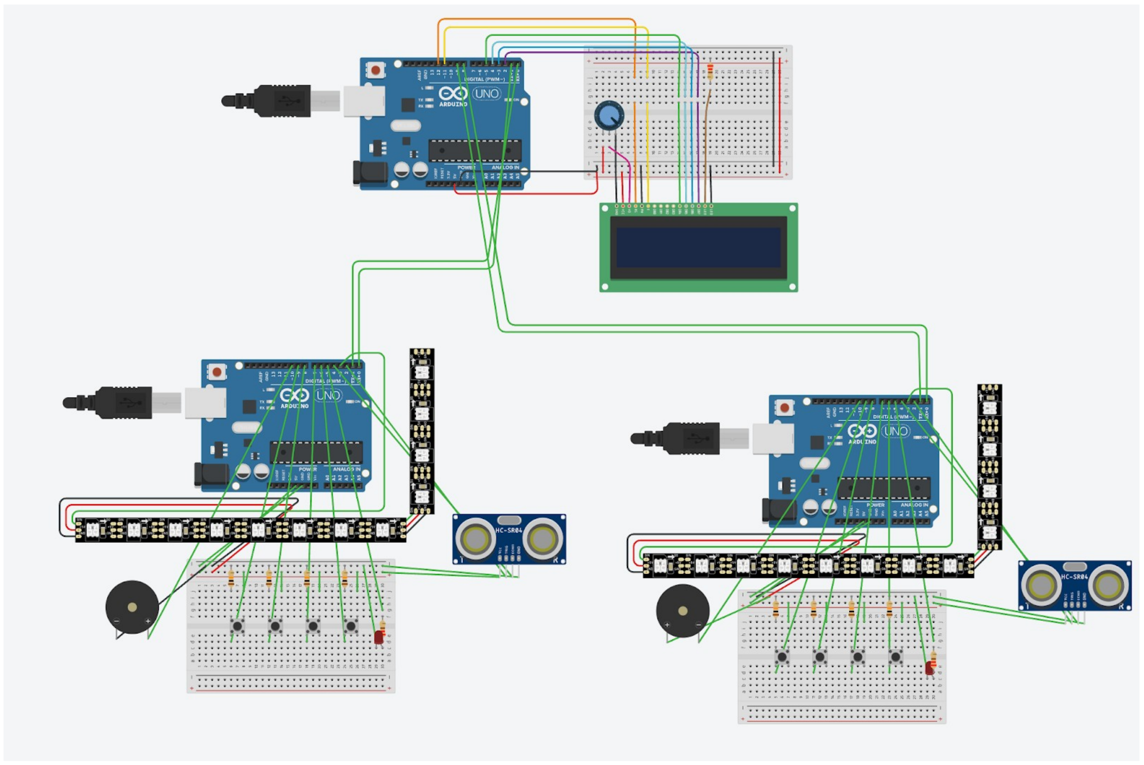
- Materials:
    - 3x Arduino
    - 3x Breadboard
    - 2x LED Dot Matrix Panel
    - LCD Display
    - 2x Passive Buzzer
    - 8x Push Buttons
    - Potentiometer
    - 2x Ultrasonic Sensor
    - 2x LED

  - 8x 10k Ohm resistors
  - 3x 220 Ohm resistors
  - 2x 100 Ohm Resistors
  - Wires

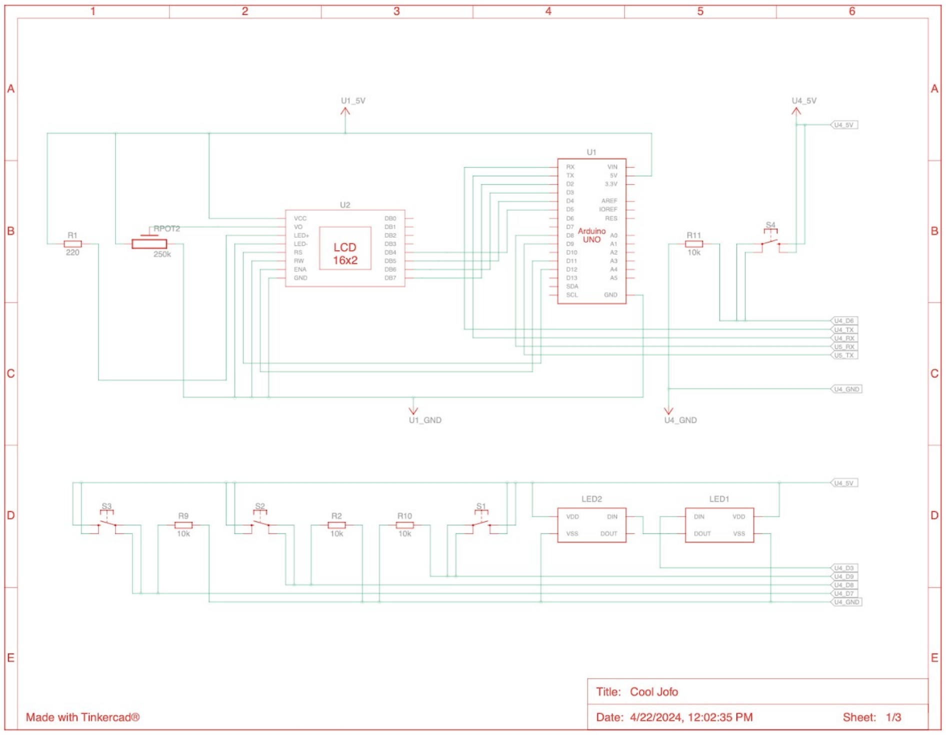
- References:

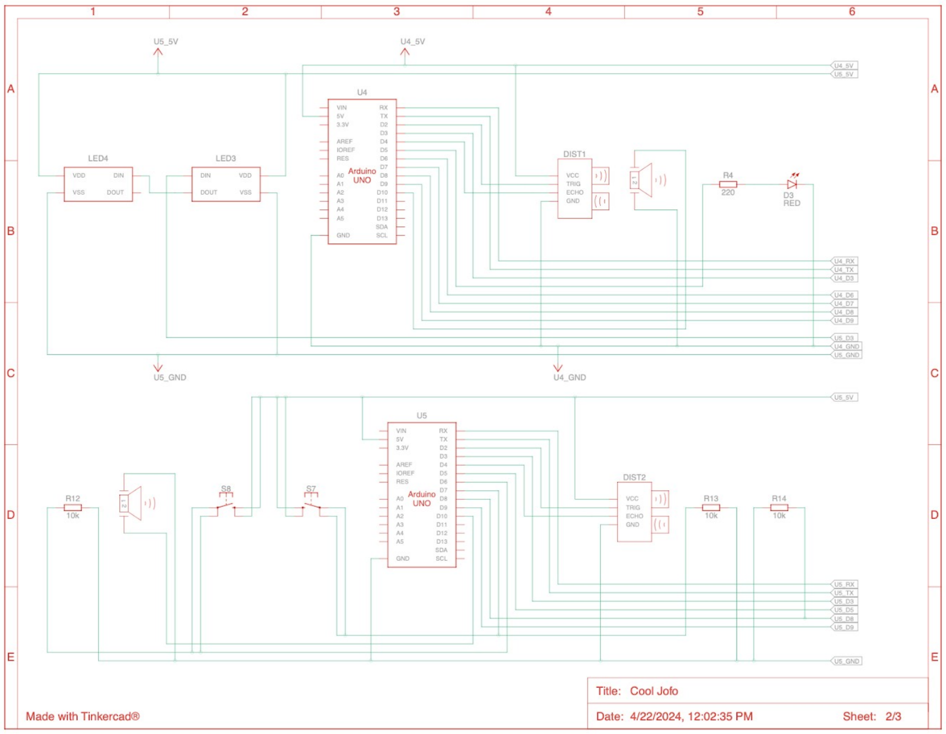
  - https://www.circuitbasics.com/how-to-setup-an-led-matrix-on-the-arduino/
  - https://docs.arduino.cc/built-in-examples/digital/Debounce/
  - https://www.instructables.com/I2C-between-Arduinos/
  - https://docs.arduino.cc/learn/electronics/lcd-displays/
  - https://create.arduino.cc/projecthub/SURYATEJA/use-a-buzzer-module-piezo-speaker-using-arduino-uno-89df45
  - https://www.gamedeveloper.com/audio/coding-to-the-beat---under-the-hood-of-a-rhythm-game-in-unity#close-modal
  - https://www.instructables.com/How-to-Use-a-PIR-Motion-Sensor-With-Arduino/
  - https://www.instructables.com/Arduino-I2C-and-Multiple-Slaves/
  - https://forum.arduino.cc/t/serial-communication-between-3-x-arduino-uno/921723
  - https://www.cuidevices.com/blog/rs-485-serial-interface-explained
  - https://lastminuteengineers.com/max7219-dot-matrix-arduino-tutorial/
  - https://arduinoplusplus.wordpress.com/2016/11/08/parola-fonts-a-to-z-defining-fonts/
  - https://pjrp.github.io/MDParolaFontEditor
  - https://arduinoplusplus.wordpress.com/2017/10/29/robot-eye-expressions-using-led-matrix-display/
  - https://arduinoplusplus.wordpress.com/2017/03/02/parola-a-to-z-managing-animation/
  - https://majicdesigns.github.io/MD_Parola/index.html

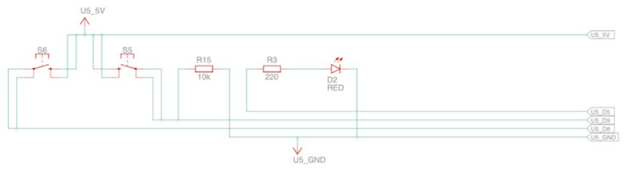
- Diagrams:



(No LED matrix, replaced by LED strip. Schematics below)


U1_5V
U4_5V
U1
Arduino UNO
U2
LCD 16x2
R1
220
RPOT2
250k
R11
10k
S4
U4_D6
U4_TX
U4_RX
U5_RX
U5_TX
U4_GND
U1_GND
S3
R9
10k
S2
R2
10k
R10
10k
S1
LED2
LED1
U4_5V
U4_D3
U4_D9
U4_D8
U4_D7
U4_GND
Title: Cool Jofo
Date: 4/22/2024, 12:02:35 PM
Sheet: 1/3
Made with Tinkercad®


U5_5V
U4_5V
U4
Arduino UNO
LED4
LED3
DIST1
R4
220
D3
RED
U5_GND
U4_GND
U5
Arduino UNO
R12
10k
S8
S7
DIST2
R13
10k
R14
10k
Title: Cool Jofo
Made with Tinkercad®
Date: 4/22/2024, 12:02:35 PM
Sheet: 2/3



Made with Tinkercad®

Title: Cool Jofo

Date: 4/22/2024, 12:02:35 PM

Sheet: 3/3

Code Sketches:

Player Arduino Code: Players are identical except A is replace with B and X with Y for serial write

```
//Matrix
#include <MD_Parola.h>
#include <MD_MAX72XX.h>
#include <SPI.h>
#include "notes.h" // Custom font to make animations easier. Each
note sprite is a character in the font
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW // Define hardware type,
number of displays (daisy-chained led matrices) and output pins
#define MAX_DEVICES 4
#define CS_PIN 3
#define ROUND_LEN 50 // change x to however many notes we want a
round to last
#define ROUNDS 3
;MD_Parola display = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);
// Create a new instance of the MD_Parola class with pin/hardware
info

//Sensor
#define TRIGGER_PIN 2
#define ECHO_PIN 4

//Game Info
bool game_state = false; //false means game is not active, true
means the game is being played (and animations are running for
matrix)
int curr_round = 0;      //Keeps track of current round
char allRounds[ROUNDS][ROUND_LEN]; //Stores string for matrix
bool startRound = false;
int roundIndex = 0;
int SEED = 1;

//Matrix Syncing
```

```
unsigned long roundStart = 0; unsigned long roundCurr = 0; int
roundIter = 500;
int SPEED = 150;

//Buttons/Debouncing
int currButtonState;
int prevButtonState = LOW;
int buttonState = LOW;
unsigned long buttonTime = 0;
unsigned long buttonWait = 25;
int b1 = 6; int b2 = 7; int b3 = 8; int b4 = 9;

//Sending
bool buttonLight = false;
int lightIndex;

//Buzzer
const int buzz_pin = 10;
const int hit_tone = 1473;
const int miss_tone = 4435;
const int buzz_period = 25;

//LED
const int ledPin = 5;


int DR(int pin){
  return digitalRead(pin);
}

bool decoder(int i){
  if (i == 1)       return (DR(b1)) == 1;
  else if (i == 2)  return (DR(b2)) == 1;
  else if (i == 3)  return (DR(b3)) == 1;
  else if (i == 4)  return (DR(b4)) == 1;
  else if (i == 5)  return (DR(b1) == 1)&&(DR(b2) == 1);
  else if (i == 6)  return (DR(b2) == 1)&&(DR(b3) == 1);
  else if (i == 7)  return (DR(b3) == 1)&&(DR(b4) == 1);
  else if (i == 8)  return (DR(b1) == 1)&&(DR(b2) == 1)&&(DR(b3) ==
1);
  else if (i == 9)  return (DR(b2) == 1)&&(DR(b3) == 1)&&(DR(b4) ==
1);
  else if (i == 10) return (DR(b1) == 1)&&(DR(b2) == 1)&&(DR(b3) ==
1)&&(DR(b4) == 1);
  else if (i == 11) return (DR(b1) == 1)&&(DR(b3) == 1);
  else if (i == 12) return (DR(b1) == 1)&&(DR(b4) == 1);
  else if (i == 13) return (DR(b2) == 1)&&(DR(b4) == 1);
  else if (i == 14) return (DR(b1) == 1)&&(DR(b2) == 1)&&(DR(b4) ==
1);
  else if (i == 15) return (DR(b1) == 1)&&(DR(b3) == 1)&&(DR(b4) ==
1);
  else              return false;
}


int debounce(int index){
  //Returns debounced button input
  if (decoder(index))
    buttonState = 1;
  else
    buttonState = 0;

  if (buttonState != prevButtonState)
        buttonTime = millis();
  if ((millis() - buttonTime) > buttonWait) {
    if (buttonState != currButtonState) {
      currButtonState = buttonState;
      if (currButtonState == HIGH)
        return 1;
    }
  }
  prevButtonState = buttonState;
  return 0;
}


void sendButton(int index){
  //If user presses correct buttons send hit else sends miss and
play corresponding sound
  if ((roundCurr-roundStart) >= roundIter){
    if (buttonLight){
      Serial.write('A');
      tone(buzz_pin, hit_tone, buzz_period);
    }else{
      Serial.write('X');
    }
    buttonLight = false;
    roundStart = roundCurr;
    lightIndex = allRounds[curr_round][++roundIndex];
  }
}


void setup() {
  //Initializations the buttons
  pinMode(b1,INPUT);
  pinMode(b2,INPUT);
  pinMode(b3,INPUT);
  pinMode(b4,INPUT);
  //Sensor
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  //Buzzer
  pinMode(buzz_pin,OUTPUT);
  //LED
  pinMode(ledPin,OUTPUT);

  // Initialize the display object and serial monitor
  display.begin();
  Serial.begin(9600);
  // Set the intensity (brightness) of the display (0-15)
  display.setIntensity(0); // Start with low intensity, since
higher ones could require an external power source
  display.setInvert(false); // Makes sure matix doesn't invert on
bootup (all lights on at once can draw too much power)
  display.displayClear();
  display.displayReset();
  display.setFont(notes);

  //Intializes the matrix char arrays
  srand(SEED);
  for(int i = 0; i < ROUNDS; i++){
    for (int j = 0; j < ROUND_LEN - 1; j++)
      allRounds[i][j] = random(1,16);
    allRounds[i][ROUND_LEN - 1] = 0;
  }
  delay(1000);
}


void loop() {
  if (!game_state){
    while(true){
      digitalWrite(TRIGGER_PIN, LOW);
      delayMicroseconds(2);
      digitalWrite(TRIGGER_PIN, HIGH);
      delayMicroseconds(10);
      digitalWrite(TRIGGER_PIN, LOW);

      // measure the time it takes for the echo signal to return
      long duration = pulseIn(ECHO_PIN, HIGH);

      // calculate distance
      float distance_cm = duration * 0.034 / 2;
      // check if there is an obstacle within range
      if (distance_cm < 5) {
        Serial.write("A");
        digitalWrite(ledPin,HIGH);
        game_state = true;
        break;
      }
    }
  }

  //Checks when main signifies to start a round
  if (Serial.available() > 0 && game_state){
    char input = Serial.read();
    if (input == 'X')
      startRound = true;
  }

  if (startRound && (curr_round < ROUNDS)){
    startRound = false;
    roundIndex = 0;
    bool setRound = false;
    digitalWrite(ledPin,HIGH);
    display.displayText(allRounds[curr_round], PA_LEFT, SPEED, 0,
PA_SCROLL_LEFT, PA_SCROLL_LEFT); //Displays round sequence
    while (!display.displayAnimate()) {
      roundCurr = millis();
      if (roundIndex < ROUND_LEN){
        if (roundIndex == 0 && !setRound){ //Sets delay until first
light reaches bottom
          roundIter = 4150;
          roundStart = roundCurr;
          setRound = true;
        }else if(roundIndex == 1 && setRound){
          roundIter = 750;
          setRound = false;
          lightIndex = allRounds[curr_round][roundIndex];
        }

        if (debounce(lightIndex) == 1) //Checks if hit or miss
          buttonLight = true;
        sendButton(lightIndex);
      }
    };
    digitalWrite(ledPin,LOW);
    Serial.write('$');
    display.displayReset();//Resets and updates current round
    display.displayClear();
    curr_round++;
    startRound = false;
  }
}
```

Main Arduino Code

```
// This file contains the code for the main Arduino. It
communicates with the two controller Arduinos and controls
// a 16x2 LCD display that outputs game information to the players.
On startup, it waits for a ready signal from both
// controllers, and after receiving the ready signals, signals the
controllers to start the game. While the game is in
// progress, it receives hit/miss data from each controller, then
calculates and displays each player's score on the LCD.
// Once the total number of hits/misses received is equal to the
predetermined round length, the game ends and the winning
// player is printed to the display.


#include <LiquidCrystal.h>
#include <AltSoftSerial.h>
// Define game state values and the number of notes/button presses
in a game
#define PREGAME 0
#define COUNTDOWN 1
```

```
#define IN_PROGRESS 2
#define GAME_OVER 6
#define ROUND_LEN 50

// Set up variables for the LCD display
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Variables for keeping track of overall game data.
int game_state;
char in_byte; // for reading in Serial input
char game_string[ROUND_LEN];
bool p1_ready = false;
bool p2_ready = false;

// Variables for in-progress data (scores, consecutive hits,
current multiplier)
int p1_score = 0;
int p1_hits = 0;
int p1_multi = 1;
int p2_score = 0;
int p2_hits = 0;
int p2_multi = 1;
int p1_wins = 0;
int p2_wins = 0;
// Keep track of total hit/misses received from both players
int p1_counter = 0;
int p2_counter = 0;

// Variables for timings using millis()
unsigned long countdown_start;
unsigned long countdown_period = 3000;
unsigned long last_recieved;  // for tracking

AltSoftSerial altSerial; // RX, TX
```

```
// Generates a string of note sprites for 1 round.
void generate_game_string() {
 for (int i = 0; i < ROUND_LEN - 1; i++) {
   game_string[i] = random(1, 16);
 }
 game_string[ROUND_LEN - 1] = 0;  // null terminate
}


// Sends a string of note sprites to the controllers
void send_string() {
 for (int i = 0; i < ROUND_LEN; i++) {
   Serial.write(game_string[i]);
   altSerial.write(game_string[i]);
 }
 Serial.write('$');  // Use '$' to indicate the end of sent data
 altSerial.write('$');
}


// Attempts to read from Serial and altSerial sources to see if
each player is ready.
void check_ready() {
 // Read in the next available byte and compare it to each player's
ready character
 // Check player 1 Ready
 if (Serial.available() > 0) {
   in_byte = Serial.read();
   if (in_byte == 'A') {
     p1_ready = true;
   }
 }
  //Check player 2 Ready
 if (altSerial.available() > 0) {
   in_byte = altSerial.read();
  if (in_byte == 'B') {
      p2_ready = true;
    }
  }
}


// Returns a score multiplier for the given player based on their
number of
// consecutive hits. Multipliers are:
// 10 note streak  = 2x, 20 note streak = 3x, 30 note streak = 4x
int get_multiplier(int player) {
 int num_hits;
 // Check which player we're getting the multiplier for player 1 or
2
 if (player == 1)
   num_hits = p1_hits;
 else
   num_hits = p2_hits;

 // Return the appropriate multiplier value based on player streak
 if (num_hits > 20)
   return 4;
 else if (num_hits > 10)
   return 3;
 else if (num_hits > 5)
   return 2;
 else
   return 1;
}

// Updates the given players score
// Each hit is worth 10 points times the current multiplier
void update_score(int player) {
 if (player == 1) {
   p1_hits++;
   p1_score += 10 * get_multiplier(1);
 } else {
   p2_hits++;
   p2_score += 10 * get_multiplier(2);
 }
}

// Same idea as check_ready(). Have each controller send it's own
distinct character for a hit/miss
// player 1: Hit = 'A' Miss = 'X'
// player 2: Hit = 'B', Miss = 'Y'
void check_hit() {

  // Check hit/miss from player 1
 if (Serial.available() > 0) {
   char input[1];
   Serial.readBytes(input, 1);
  // On hit, call updatescore
   if (input[0] == 'A') {
     update_score(1);
     print_scores();
     p1_counter++;
  // Reset consecutive hits for misses
   } else if (input[0] == 'X') {
     p1_hits = 0;
     p1_counter++;
    }
 }

  // Check for hit/miss from player 2
 if (altSerial.available() > 0) {
   char input[1];
   Serial.readBytes(input, 1);
  // On hit, call updatescore
   if (input[0] == 'B') {
     update_score(2);
     print_scores();
     p2_counter++;
  // Reset consecutive hits for misses
  } else if (input[0] == 'Y') {
     p2_hits = 0;
     p2_counter++;
    }
  }
}

// Helper function that prints player scores to the lcd
// Player 1 is printed on the top half of the screen, and player 2
is on the bottom
void print_scores() {
 lcd.setCursor(0, 0);
 lcd.print("Player 1: " + String(p1_score));
 lcd.setCursor(0, 1);
 lcd.print("Player 2: " + String(p2_score));
 lcd.setCursor(0, 0);  // put cursor back to start (in case other
functions print after this one)
}

void setup() {
 // Initialize Serial and lcd display
 Serial.begin(9600);
 altSerial.begin(9600);
 lcd.begin(16, 2);
 lcd.setCursor(0, 0);

 // Initialize globals
 game_state = PREGAME;
 lcd.print("Guitarduino Hero");
 lcd.setCursor(0, 1);
 lcd.print("Ready?");
 lcd.setCursor(0, 0);
}
```

```
void loop() {
 // Before game starts, wait for ready signal
 if (game_state == PREGAME) {
   // Keep checking for input until both controllers are ready
   check_ready();
   if (p1_ready && p2_ready) {
     for (int i = 0; i < 3; i++) {
       generate_game_string();
       send_string();
     }
     Serial.write('@');
     altSerial.write('@');
     lcd.clear();
     game_state = COUNTDOWN;
     countdown_start = millis();  // initialize countdown start
time
   }
 }

 // Countdown for 3 seconds after both players are ready
 else if (game_state == COUNTDOWN) {
   unsigned long counter = millis() - countdown_start;
   lcd.setCursor(7, 0);
   if (counter < 3000) {
     lcd.print(3 - (counter / 1000));  // countdown from 3
   } else {
     // Print start on lcd and give a short delay (maybe not
necessary)
     lcd.setCursor(5, 0);
     lcd.print("START!");
     delay(1500);
     lcd.clear();
     print_scores();
     // Tell controllers to start game and transition game state to
IN_PROGRESS
     Serial.write('X');  // Can use any character for start symbol
     altSerial.write('X');  // Can use any character for start
symbol
     game_state = IN_PROGRESS;
     // initialize player scores and hits
     p1_score = 0;
     p1_hits = 0;
     p1_multi = 1;
     p2_score = 0;
     p2_hits = 0;
     p2_multi = 1;
   }
 }

 // Main gameplay loop, continuously update scores until game ends
 else if (game_state == IN_PROGRESS) {
   check_hit();
  // End the game once both note counters are >= than ROUND_LEN
   if(p1_counter >= ROUND_LEN && p2_counter >= ROUND_LEN){
      game_state == GAME_OVER;
   }
 }

 // End game state. Display a game over message and the winning
player.
 else if (game_state == GAME_OVER) {
   lcd.clear();
   lcd.setCursor(3, 0);
   lcd.print("Game Over!");
   lcd.setCursor(0, 1);
   if (p1_wins > p2_wins) {
     lcd.print("Winner: Player 1!");
   } else {
     lcd.print("Winner: Player 2!");
   }
   game_state = 10;
 }
```

```
}
```

“Notes.h” Header file: Defines a custom font for use with the MD_Parola library

```
// Header file “notes.h”
// Custom font that makes up the note sprites displayed on the
matrix display
// Must be included in the controller code (save in same folder as
the .ino file)
MD_MAX72XX::fontType_t notes[] PROGMEM =
{
    0,     // 0     // null
    4, 192, 192, 0, 0,     // 1  note-1
    4, 48, 48, 0, 0,     // 2  note-2
    4, 12, 12, 0, 0,     // 3  note-3
    4, 3, 3, 0, 0,     // 4  note-4
    4, 240, 240, 0, 0,     // 5  note-1-2
    4, 60, 60, 0, 0,     // 6  note-2-3
    4, 15, 15, 0, 0,     // 7  note-3-4
    4, 252, 252, 0, 0,     // 8  note-1-2-3
    4, 63, 63, 0, 0,     // 9  note-2-3-4
    4, 255, 255, 0, 0,     // 10  note-1-2-3-4
    4, 204, 204, 0, 0,     // 11  note-1-3
    4, 195, 195, 0, 0,     // 12  note-1-4
    4, 51, 51, 0, 0,     // 13  note-2-4
    4, 243, 243, 0, 0,     // 14  note-1-2-4
    4, 207, 207, 0, 0,     // 15  note-1-3-4
    4, 0, 0, 0, 0,     // 16  blank
    // Can have up to 256 chars/sprites, but we only need these 17
for now
} // End of “notes.h”
```