Advanced Mechatronics Project 1: PONG

Greta Perez-Haiek, Aadhav Sivakumar, Nathan Smurthwaite

February 27th, 2025

1 Abstract

Pong, one of the most impactful games of the 20th century, remains a classic example of early video game innovation. This project aims to recreate the iconic game using an Arduino-based system, using joysticks for user input, an LED matrix for display, a buzzer for the sound, and some seven-segment displays for score and timekeeping. Through programming techniques and electronic components, we developed a functional, interactive version of Pong that maintains the spirit of the original while including modern digital control. This report goes through the design, implementation, and challenges faced during the development process, including all of the displays, memory optimization, and construction. The final demonstration of the project showcases the functionality of all sub-systems, resulting in an entertaining and transportable game module that two players can enjoy competitively.

2 History

Pong is a game that was created in 1972 as one of the earliest types of arcade games. The original game included no code, and was completely implemented through a combination of digital and analog circuitry. The creator originally

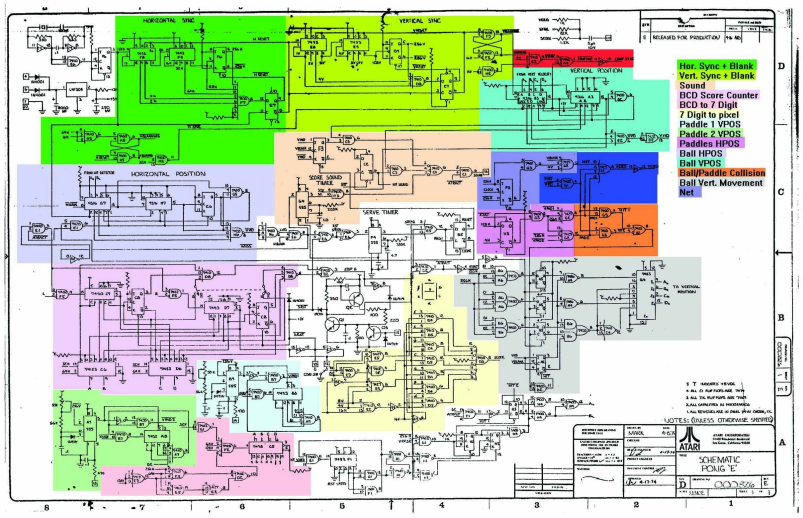


Figure 1: Schematic of the first pong game

made a small prototype, and was asked to create the entire thing with less than 20 chips for mass production. He ended up doing it with about 70, but the people still moved along with production due to its massive popularity.

3 Components

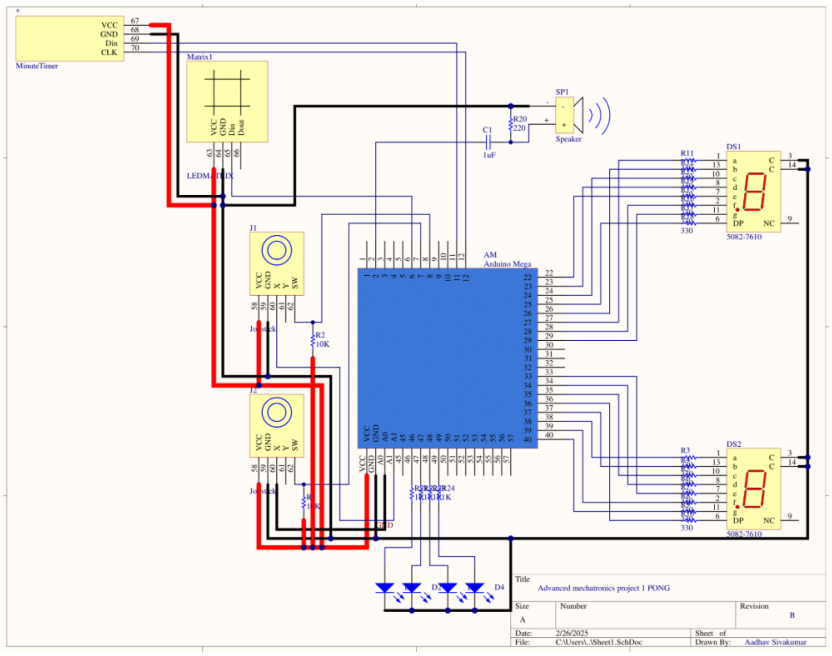


Figure 2: Schematic of all the components used

3.1 Joysticks

For this project, there are only 2 inputs being considered, two of the joystick modules that came as part of a larger kit.

Figure 3: Singular joystick module

This module requires a 5V power supply as well as a ground for inputs. There are two analog outputs for the signal, one for the movement in the x axis and one for the movement in the y axis. There is also a digital output, which is a pull-down switch, leaving the voltage floating when the joystick isn't

pressed in and pulling it to ground when it is. In the circuit there is a large pull-up resistor connected to the pin such that the voltage isn't floating when unpressed.

3.2 LED Matrix

When starting the project we used a small 8x8 LED matrix. This also had a small board attached to it, which was able to turn serial data into display data. Each row of the Matrix corresponded to a single byte, so it only takes 8 bytes to control the entire thing. A simple version of PONG was implemented on this smaller screen, only able to handle straight and 45 degree balls, and with no replayability.

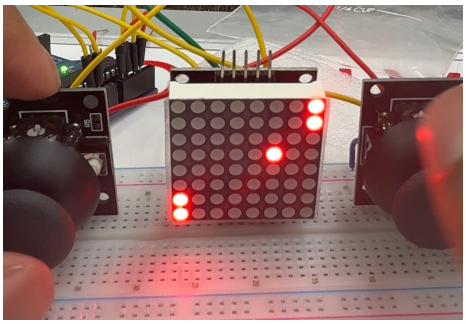


Figure 4: Initial LED matrix

Instead of being stuck with this limited design, we decided to go with a 16x16 display instead. This is harder to control and takes more current than the 8x8 display, but allows for more play area and a better user experience.

When attempting to get the matrix working for the first time, there was immediately a problem. We had to use the FASTLed.h library for individual LED control, and this assigned each LED a corresponding index in an array, with each space in the array containing an object that contains red, green, and blue values for each pixel. However, the sequence goes down the matrix in a snake pattern, instead of reaching the end of the row and restarting at the left side on a new row. To fix this, a map was implemented, numbering each pixel in the snake pattern with a new number representing where it would be in a standard matrix setup (row by row). This allowed us to use a separate matrix to represent the whole board and then put that information through this new mapping matrix to ensure the display shows up as intended.

Creating a 16x16 matrix to represent the board initially caused some problems because the original datatype used to represent each pixel was an int. This effectively instantiated 256 ints for the board and another 256 ints for the mapping matrix, which caused some memory issues. Since the game of PONG only has 3 different things on screen (paddles, ball, and background), any datatype that can represent 3 or more values can be used. Swapping out the integers

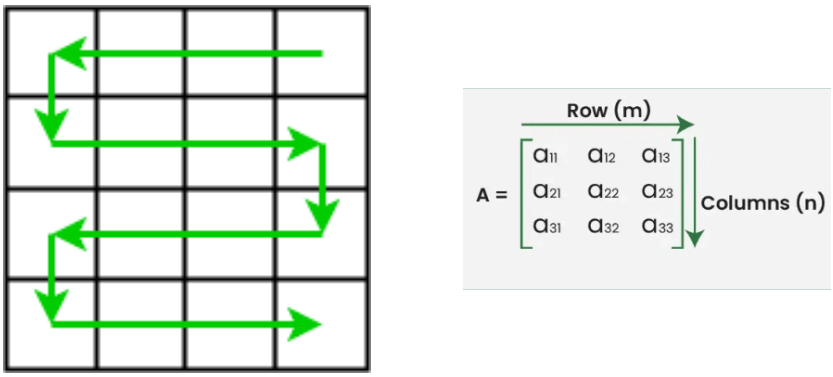


Figure 5: Original traversal sequence and ideal matrix

with unsigned chars significantly reduced the amount of memory used.

3.3 Seven-Segment Displays

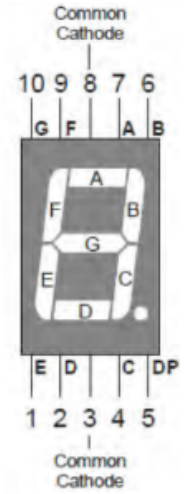


Figure 6: Schematic for 7-Segment Display

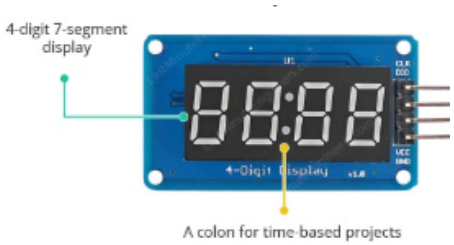


Figure 7: Schematic for 4-Digit 7 Segment Display with TM1637

Figure 8: SFM-27 Buzzer Module

4 Programming

The program was split into a few main functions:

4.1 Joystick Functions

4.1.1 readJOY

Reads the analog value for the joysticks and puts them into a global variable

4.1.2 refresh_paddles

Takes the joystick position and moves the paddle position accordingly. Also populates the board matrix with the number 2, representing where the paddle should be drawn.

4.2 Matrix Functions

4.2.1 arrayToLEDs

Takes in the board matrix and uses the LED mapping matrix to correctly display the paddle, ball, and the background. It ensures that instead of using a

snaking pattern, it uses a matrix pattern to align the positioning of the board matrix containing all the positions of things

4.2.2 board321

Displays the numbers 3 2 1 in sequence on the entire screen before a new game starts.

4.3 Game Functions

4.3.1 refresh_ball

Uses the previous direction of the ball to calculate what the next direction of the ball should be based on whether or not it hits the paddles or the side of the screen. When the ball hits the top or bottom of the screen, it bounces back at the same angle, either 45 degrees or steeply. When a ball hits a paddle, it will randomize the next direction it goes in, but it won't bounce back in the exact same direction that it came from. It calculates the winner of the game based on whether or not the ball reaches the right or the left side of the screen.

4.3.2 update_score_reset

Handles the post game functionality, like incrementing the score and waiting for both joysticks to be pressed in order to start the next round. It does this through playing states, with 1 representing normal gameplay, 2 being used to increment scores, 3 being used to play the end game sound if one player gets to 5 points, and 4 being used to wait for the reset signal (double joystick press) and reset all of the variables necessary to start the next round/game.

4.4 Seven Segment display Function

4.4.1 showDigit

This function displays the number called to it on the 8-digit Display by setting specific pins that are allocated to LEDs within the display to high or low based on bit values within the binary that represent that number for this unique display.

4.5 Timer Functions

4.5.1 startClock

First function necessary to start communication with the TM1637 Driver. The data pin is set high followed by the clock pin. This requires a micro delay as setting both pins high in the same line causes communication issues in which nothing would display. Following this, the clock pin and data pin are set low. Microdelays are also used between these statements. Given the frequency in

which this function is called, port registers are used for setting these pins high and low

4.5.2 stopClock

The second half of the communication process for the TM1637 Driver. Upon calling this function, the clock pin is set low followed by the data pin. In turn, the clock pin is then set high followed by the data pin. This function also contains Microdelays to ensure clean signal transmission.

4.5.3 writeValue

This function takes in the desired value to be displayed which is designated by a binary or hex code that is declared globally in an array. Within a for loop, every bit of the desired value is compared against a full byte to check which LED within the array of that digit should be lit. If the respective bit is 1, then the if statement runs true and the data pin is set high followed by the clock pin being set high. If the respective bit is 0, the data pin is kept low and the following high clock pin statement now does nothing. To tie up loose ends once every bit is checked, the clock pin is set low then high again.

4.5.4 write

This function fully encompasses the writing process. It takes 4 values in as parameters. 3 specific protocols are needed for the TM1637 Driver to function properly. Each of these must be preceded by startClock and stopClock. The first is 0x40, which initializes data write mode. The second is 0xc0, this specifies which of the four digits to start writing to. The last is 0x8F, this sets the display at the maximum brightness. Between 0xc0 and 0x8F, writeValue is called 4 separate times, one for each parameter described in the function header.

4.5.5 gimmeFive

This function is called within void loop. Essentially, it acts as the timing function. It uses the integrated millis function to incorporate a non-blocking delay as to not interrupt the animation that is displayed in parallel. By polling the current millis value and checking if the difference is greater than the desired period, in our case 1000 ms, we update specific values that designated the one second placement k, the 10 second placement j, and the 1 minute placement k. If any of these variables reach their maximum values, the other variables update/reset. In particular, with each reset of the one second placement (aka after 10 seconds) the speedfactor variable is decreased by 30.

4.6 Speaker Function

4.6.1 tone

The tone function takes in the pin number that the speaker is connected to, a frequency, and a duration. It then plays that frequency for the specified duration while continuing on with the code, ensuring that it's non-blocking code. There is also a high pass filter connected to get rid of all the unwanted low frequencies, which caused some problems at the start.

5 Construction

The project was completed over the course of a 3 week period, in a series of three development phases. The first phase consists of the preliminary planning and prepping stage. In this phase, the project theme was decided, and an electrical overview blueprint was sketched out. Considering the electrical and mechanical needs of the Ping-Pong Machine, a bill of materials was drafted and parts were then consequentially ordered. This phase took up all of Week 1 of the GANTT chart (see Appendix "Project Presentation") and half of Week 2.

Once the parts arrived, each sub-system (the LED matrix, the Clock, the Counter, the Speakers, and the Joystick Controls) of the project was tested and validated. The purpose of these tests was to experiment with the programming and become familiar with how the component behaves. Some examples of the tests was having the LED 8-Segment LED display count up from 0 to 9, or having the 8 x 8 matrix display a cell of color, controlled by the joystick. The buzzers were also toyed around with, creating all sorts of noises to experiment which ones would be the most entertaining for game implementation. Once all sub-systems of the game module have been verified, a miniature "proof-of-concept" was engineered (See Figure 3) was engineered using the 8 x 8 LED matrix display. Despite it being small, it proved to be entertaining, and to proved better visibility, the 8 x 8 display was soon replaced with a 16 x 16 LED matrix display. This phase took up the majority of Week 2 and some of week 3.

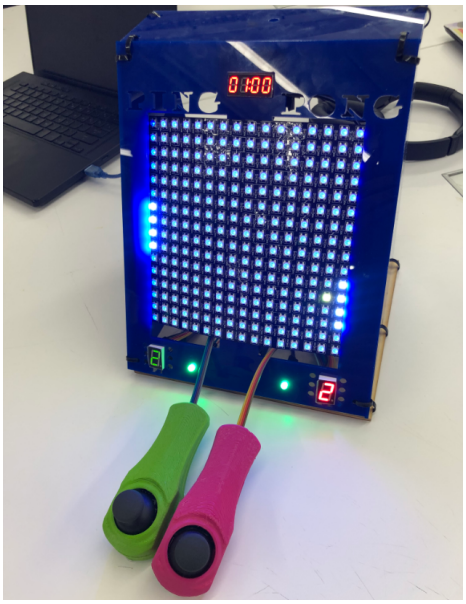


Figure 9: Final Prototype

The last phase of the project involves the iterable design and construction of the various components that will house the electronics. The "housing" structure was designed using Fusion 360 CAD, and initially 3D printed as a proof of concept. The structure, using CAD, was then blown up to it's face components, then laser cut out of shiny blue acrylic. The base of the structure was laser-cut out of wood. The structure took the tolerances of the 8-segment display and the clock in mind so that the components can easily be housed and visible. The faces of the structure was zip-tied together carefully, and the 16 x 16 LED matrix was installed meticulously so that it curves slightly to the human eye (to replicate the old-school "Atari" gaming aesthetic, and to be more easily seen from the sides). To house the joysticks, a basic "sleeve" design was first engineered using Fusion 360, but then we settled on an Open Source Joystick file, as it seemed to be more ergonomically sound than the initial design. The final component was 3D printed using green and pink PLA, to represent Player 1 and Player 2 respectfully. The "sleeves" were then screwed together to encase the joystick components to protect the electronics as the players play. The other electronics (such as the LEDs, clocks, 8-segment display, speakers, etc) was installed in the housing, and the programming (which was being developed over the course of Week 2 and Week 3), was uploaded. After assembly, the game was extensively tested for programming bugs and errors, and the code was modified accordingly. This phase took up the majority of Week 3, and resulted in the final prototype seen in Figure 9.

6 How to Build Your Own Ping Pong Machine

The files required to create this ping-pong machine are found on GitHub under the project "mythicane/DIY-Ping-Pong-Machine" (https://github.com/mythicane/DIY-Ping-Pong-Machine). This is a beginner-intermediate level project.

6.1 Manufacturing the Hull

Under the ".stl files" folder in the "DIY Ping Pong Machine" Github, download "Front face.stl," "bottom.stl," "side.stl," and "top.stl". It is possible to 3D print these parts, CNC them, or laser-cut them. A Ping Pong Machine requires (x1) front face part, (1x) bottom part, (2x) side parts, and (1) top part, so be sure to manufacture the necessary number of parts. For our specific prototype, we have laser-cut the parts out of Acrylic, as that is the fastest given the durability. All five of the parts could fit be nested within a 12 in. by 24 in. panel for subtractive manufacturing purposes. 3D printing may be a little bit more difficult, as some of the parts do not fit in an Ultimaker 3D Printer.

Notice the holes on the edges of each part. Once the parts have been created, feel free to zip-tie them together. It may be possible to create brackets to help secure the faces of the structure together- however, this is not necessary. The structure is stable with zip-ties alone. It is recommended to zip-tie the sides to the base, then secure the face to the sides and base, then secure the top to the face and the sides last. The back should be open to accommodate for the electronics.

6.2 Manufacturing the Joystick sleeves

Under the ".stl files" folder in the "DIY Ping Pong Machine" Github, download "joystick bottom.stl" and "joystick top.stl". These .stl files are intended to be 3D printed using PLA (with an infill 10-15 percent). You should print (x2) of the bottom, and (x2) of the top. Take your joystick module (see "Obtain Parts") connected to some wires (make sure to label the wires so that you do not lose track of which one is which), and nest them comfortably inside the top-bottom pair before screwing them together. It may be intelligent to wait until after assembling the entire circuit before assembling the joystick sleeves to make sure that the electronics works as intended.

6.3 Obtain Parts

The circuitry will consist of the following parts... Arduino Mega (x1), 5mm LED Diodes (x4), 1-Bit 7-Segment Display (x2), 4-Digit 7-Segment display (x1), Two-prong Buzzer (x1), Dual-axis XY joystick modules (x2), 6 x 16 256 pixel LED Matrix (flexible/full-color) (x1). Specifications for each part is listed under Section 3: Components. All parts can be found online or at your personal hardware/electronics store. It may also be necessary to buy either a 12 in. by

24 in. by 1/8 in. plank of wood or acrylic to manufacture the hull (See Section 6.1: "Manufacturing the Hull").

6.4 Electrical Engineering

Once the parts have been obtained, connect each module in accordance to the electrical schematic (See Figure 2 under Section 3: "Components"). If this is hard to see, feel free to go to the Github and open "schematic.png" for a better viewing experience.

6.5 Arduino Programming

Under the "code" folder in the "DIY Ping Pong Machine" Github, download "Main.ino." Then, open the file up your Arduino IDE. After assembling the circuit, connect the Arduino to your computer then launch the script. You do not need to modify the code if you followed the schematic properly. However, if the circuitry board does not agree with the schematic, it is likely that the game might not function properly. When encountering an error, please triple check your circuitry, power source, and physical components before touching the code. If you decide to modify your circuitry intentionally, make sure that the code reflects the changes- for example, changes in PINs. Each function of the code is listed in Section 4: Programming, for ease in understanding.

6.6 Assembly

First, slip the flexible LED matrix in the front of the hull from the inside. Curve it slightly so that it fits (recreated that old fashioned curved-screen 'Atari" aesthetic). Secure it to the inside using electrical or painter's tape. Slip the 1-bit 7-segment displays into the holes on the bottom left and right of the front face. It should fit snugly. The two holes to the side of these displays should hold the LEDs snugly as well (as shown through Figure 9). Slip the clock display in the hole on the top middle of the front face, and secure the back with electrical or painter's tape. Lastly, thread the wires connecting the joysticks out from the bottom of the LED matrix, and then encase the joystick module with the joystick sleeves (See Section 6.2: Manufacturing the Joystick sleeves). Screw the sleeves together. The joystick should fit in the sleeves: if they do not, try clearing the PLA supports from the inside of the sleeve, or triple check that the type of joystick module reflects the type at Section 3.1: "Joysticks".

6.7 Gaming: How to Play

Once you assembled the machine... now is time to play! This machine can be played as a 1 or two player game- each player will wield a joystick. Hold the joystick towards you, and then press down on the joystick to start the game. Both players must press down for the game to start. Move the joystick up and down to move your paddle up and down- moving the joystick left and right will

do nothing. The ball will then bounce back and forth at a randomized interval- as more time goes by, the ball will speed up, and the changes in angles will get more aggresive! The goal of the game is to keep the ball from hitting your side- if it does, a point will be rewarded to your opponent. After each "loss," both players must press down on their joysticks to start another round in the game. After a minute, the game will end, and the winner will be the one with the most points! Reset your Arduino to begin anew... Enjoy!

7 Appendix

7.1 Project Presentation

PONG

Reinventing a Classic Arcade Game

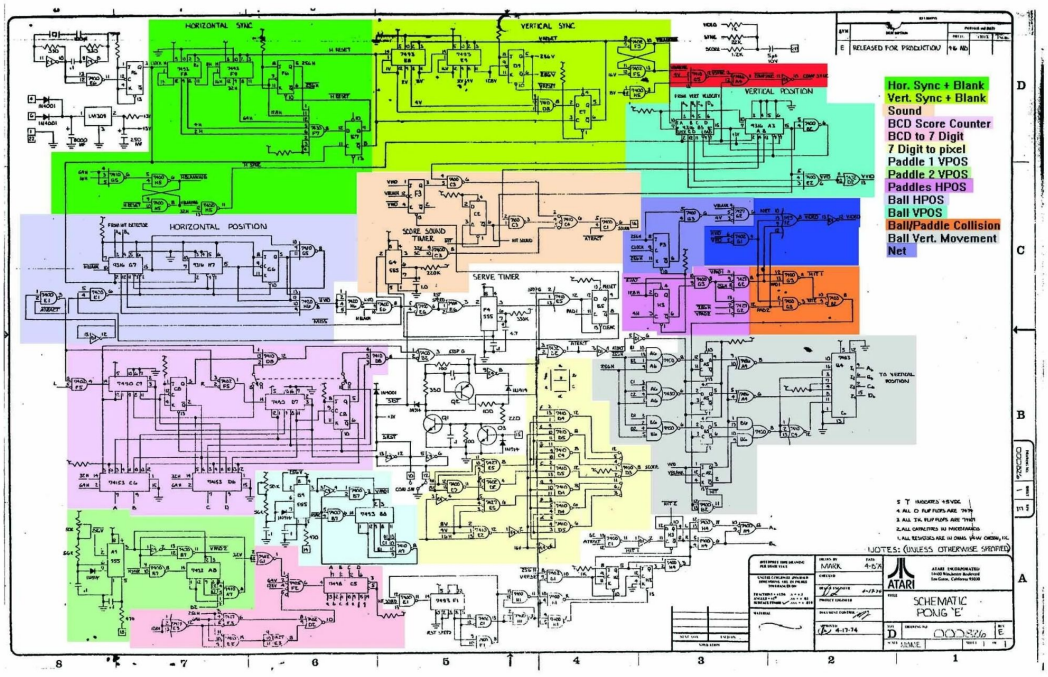
Mechatronics S25
Prof. Kapila

Greta Perez-Haiek, Aadhav Sivakumar, Nathan Smurthwaite

01

What's the Problem?

The early days of Pong were entirely **analog**…



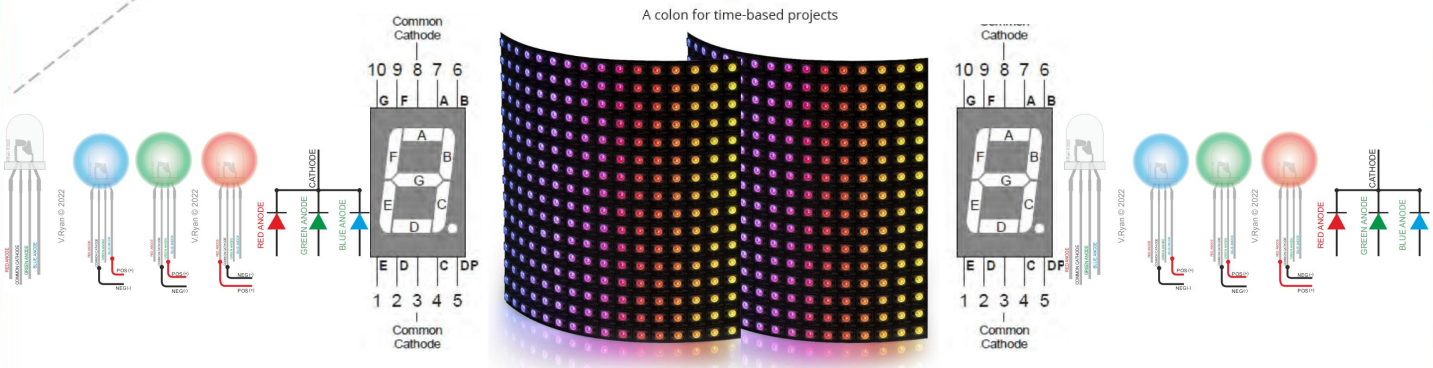
… No microcontroller or programmable device in sight.

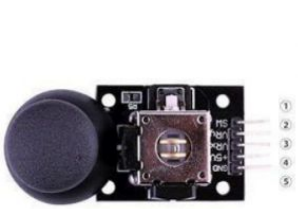
02

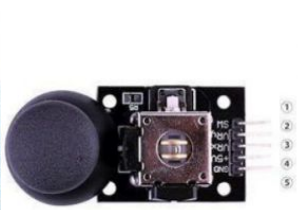
What's the Solution?

Recreate the game in a **LESS** complex manner and with **LESS** components

The Components

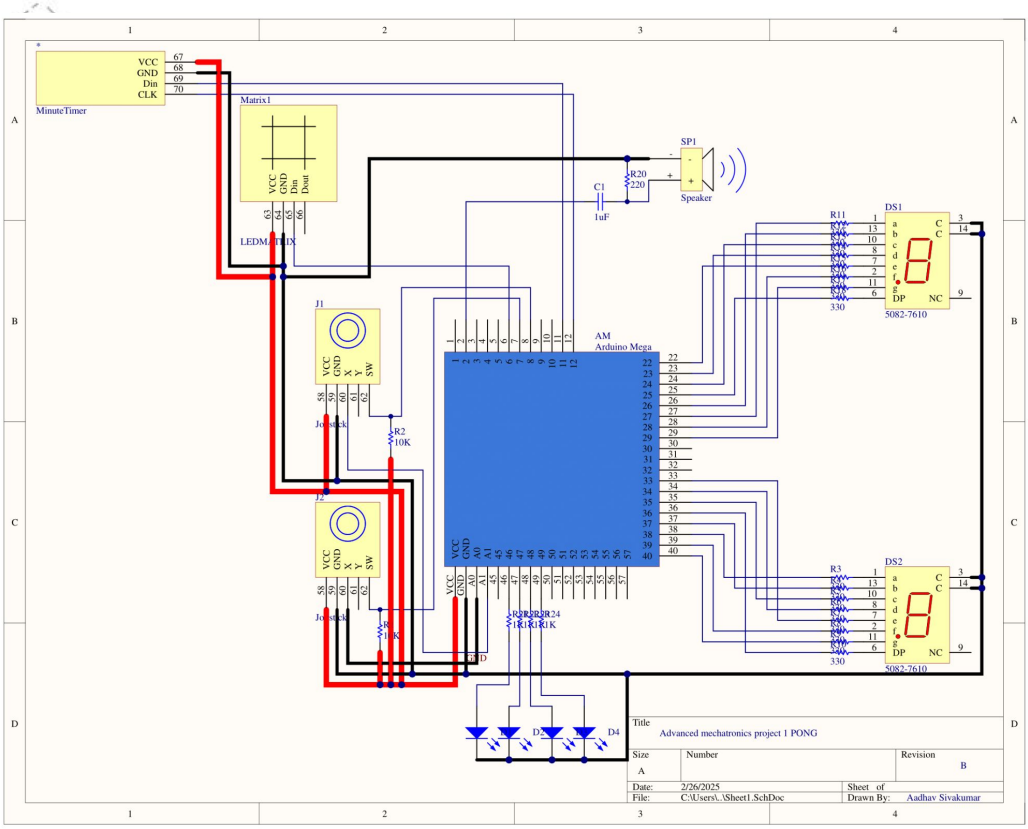




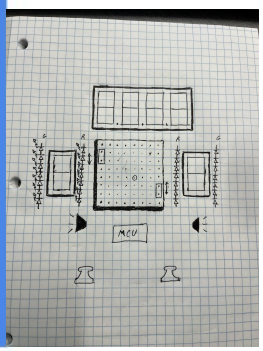


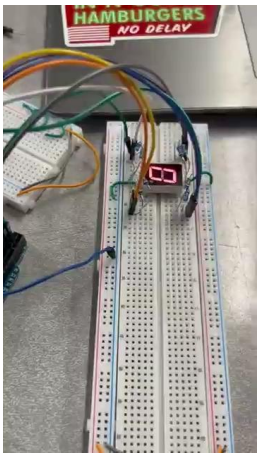
1.SW
2.VRY
3.VRX
4.VCC:+5V
5.GND:+5V

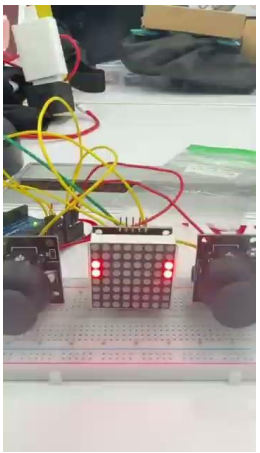
Circuit Diagram



Progress



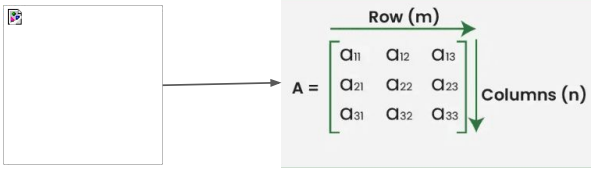




Video Demo

16x16 LED Matrix

- Max Draw of 75 Watts, needs lower brightness level to stay under current draw max of the arduino
- Color order : **G R B（NOT RGB）**
- When you light up **1 color**（red or green or blue, **0.1W/LED** .



Joystick

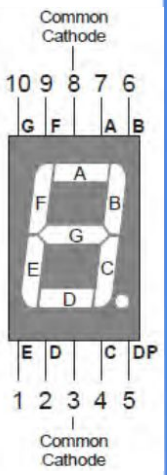
- Module requires Vcc (5V) & GND for Inputs.
- There are two analog outputs for the signal,
  - 1 for the movement in the x-axis
  - 1 for the movement in the y-axis
  - *y-axis pin not necessary given 1D movement
- 1 Digital Output, which is a pull-down switch,
  - Leaves floating voltage when joystick isn't pressed and pulling it to ground when it is

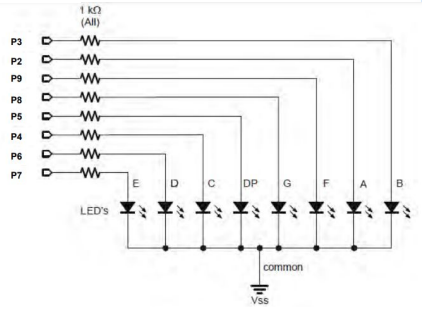
In the circuit there is a **LARGE** pull-up resistor connected to the pin such that the voltage isn't floating when switch is not pressed.

Single Digit 8-Seg LED Display

- Common Cathode
- Essentially 8 LEDs, each requiring a GPIO pin, constructed in a unique array
  - `const byte segmentPins1[8] = {25,29,28,22,23,24,26,27};`
  - `//  DP, G, F, E, D, C, B, A`
- Multiplexers (**Mux**) were considered
  - A Mux would decrease the Pin Count
  - High power draw of Matrix, mux would draw unnecessary power
- Displays the current score

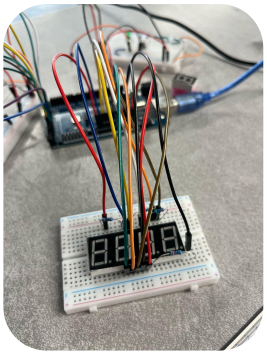
```
const byte numeral[10] = {
B11111100, // 0
B01100000, // 1
B11011010, // 2
B11110010, // 3
B01100110, // 4
B10110110, // 5
};
```

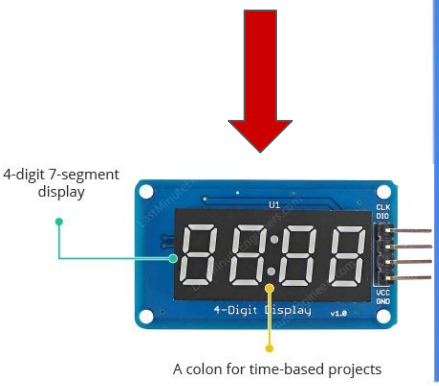




4-Digit Display w/ TM1637 Driver

- Displays clock value going from **0 s** to **1 min**
  - If game reaches **1 min**, game will end
- Typically, **12 pins** with common anode
- The **TM1637 Driver** minimizes pins through a **Mux** that reduces the Pin Count to **4**
  - **Vcc**, **DIO**, **CLK**, **GND**
- 2 wire comm. (**DIO**/**CLK**), but this is **NOT** a **TWI** device
- Protocols are similar with a **START/STOP** procedure
  - **0x08** → Max. brightness
  - **0x40** → Ready for comm.
  - **0x0c** → First to write
- Use of **`millis`**( ) for a non-blocking delay



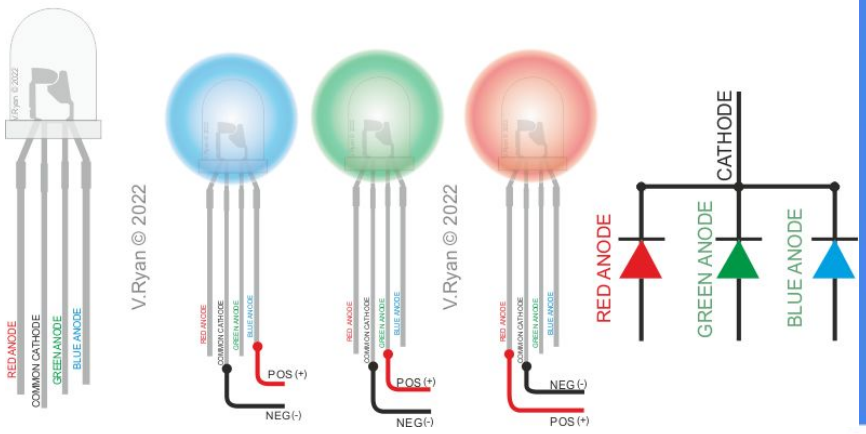


Tri-Color LED

Similar to a traditional LED

Indicates to the players who scored/lost

- Only require 2 Pins: Green, Red



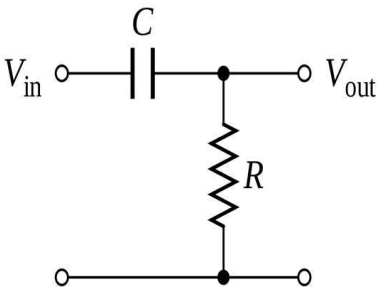
SFM-27 Buzzer Module

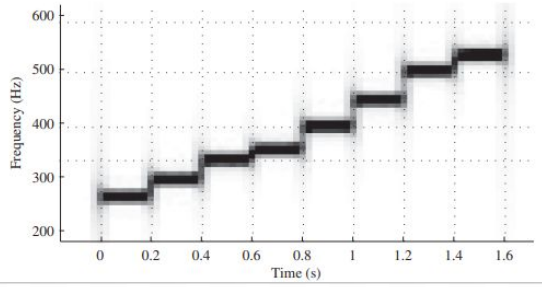
- Anode → GPIO PWM Pin
- Cathode → GND

High Pass filter to smooth out the sound

- Use of `tone()` function
  - Generates square wave of the specified freq (50% duty cycle) on pin
  - Arrays of notes for
    - P1 Hit
    - P2 Hit
    - Goal
    - Game Over

```
#define NOTE_C5 523
#define NOTE_C#5 554
#define NOTE_D5 587
#define NOTE_D#5 622
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_F#5 740
#define NOTE_G5 784
#define NOTE_G#5 831
#define NOTE_A5 880
#define NOTE_A#5 932
#define NOTE_B5 988
#define NOTE_C6 1047
#define NOTE_C#6 1109
#define NOTE_D6 1175
#define NOTE_D#6 1245
#define NOTE_E6 1319
#define NOTE_F6 1397
#define NOTE_F#6 1480
#define NOTE_G6 1568
#define NOTE_G#6 1661
#define NOTE_A6 1760
#define NOTE_A#6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093
```



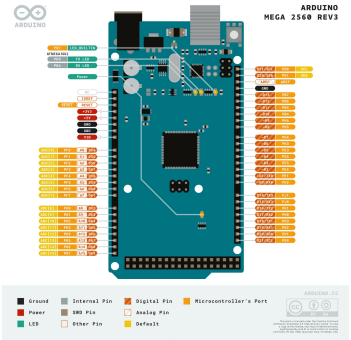


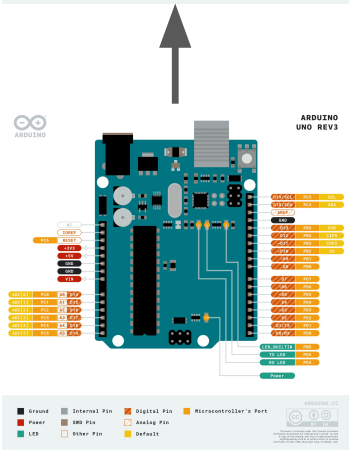
Arduino Mega 2560

- 54 Digital I/Os (14 support PWM Output)
- ATmega2560 Processor
- 16 MHz clock speed
- 256kB flash memory

Originally attempted to implement this on UNO variant (Kuman UNO)

- 16 pins for both 1-Digit Displays
- 4 pins for both joysticks
- 4 pins for both tri-color LEDs
- 2 pins for 4-Digit Display
- 1 pin for Buzzer Module
- 1 pin for LED Matrix
- 28 pins **TOTAL**

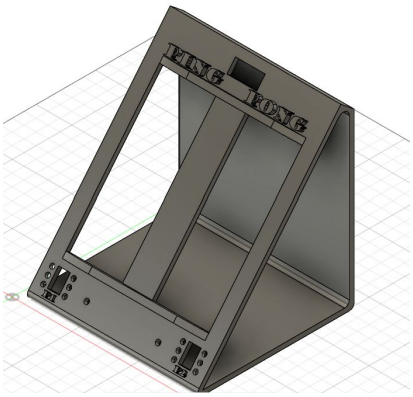




Housing

- 3D Printed Joysticks to go along with the housing
- Laser Cut Wood Base and Acrylic Casing
- Designed to house and protect the electronics
- Curved LED Matrix Display for aesthetics
- Proper tolerances for the counters, clock, and LED lights
- The joystick is stored in an ergonomically designed housing made of PLA.

LED Matrix: Length x Width x Height -**16cm x 16cm x 0.2cm**

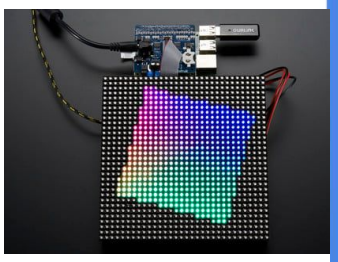


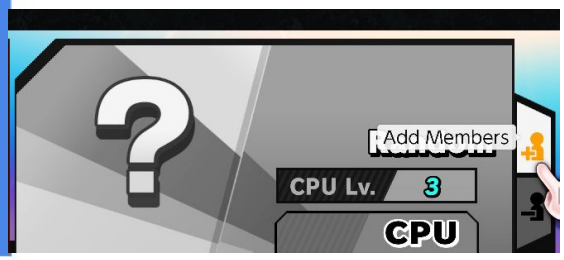
Gantt Chart

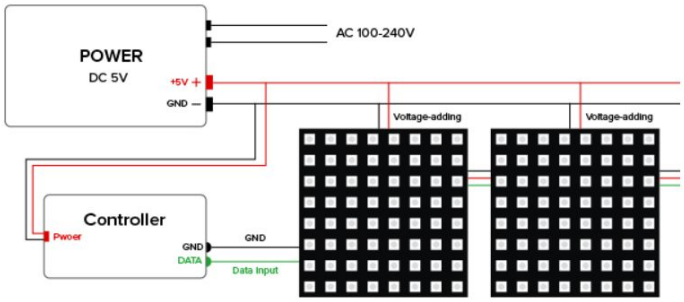
SPRING 2025	PING PONG!! GANTT CHART (Collaboration Plan)				
	Task	Team-Member	Week 1	Week 2	Week 3
	Planned and ordered parts	Greta, Aadhav, Nate			
	Brainstormed Project Themes	Greta, Aadhav, Nate			
	Created a preliminary Blueprint	Nate			
	Tested functionality of individual parts (speakers, LED matrix, counters, clocks)	Greta, Aadhav, Nate			
	Created proof of concept: 8 x 8 LED matrix	Aadhav			
	Designing Housing / Joysticks	Greta			
	Manufacturing Housing / Joysticks	Greta			
	Programming	Aadhav, Nate			
	Assembly	Greta			
	Testing (Verification and Validation)	Greta, Aadhav, Nate			

Future Work

- Creating a larger display for more challenging gameplay
- Better Resolution (More LEDs/Area)
- Computer Played Unit (CPU)
- Added Complexities (Multiple Balls)
- Map paddle movement to real-life locomotion







Would you like to play a game?

7.2 Code

```
#include <FastLED.h>//FastLED makes the matrix useable by assigning each LED to an
index in an array
//Hardcoding of certain notes and thier frequencies
#define NOTE_C5 523
#define NOTE_C#5 554
#define NOTE_D5 587
#define NOTE_D#5 622
#define NOTE_E5 659
#define NOTE_F5 698
#define NOTE_F#5 740
#define NOTE_G5 784
#define NOTE_G#5 831
#define NOTE_A5 880
#define NOTE_A#5 932
#define NOTE_B5 988
#define NOTE_C6 1047
#define NOTE_C#6 1109
#define NOTE_D6 1175
#define NOTE_D#6 1245
#define NOTE_E6 1319
#define NOTE_F6 1397
#define NOTE_F#6 1480
#define NOTE_G6 1568
#define NOTE_G#6 1661
#define NOTE_A6 1760
#define NOTE_A#6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093

//Pin used to control the LED matrix
#define LED_PIN 6
//Number of LEDS in the matrix (16x16)
#define NUM_LEDS 256
//Joystick analog inputs
#define p1joypin 0
#define p2joypin 1
//tricolor LED pins
#define LED1R 46
#define LED1G 47
#define LED2R 48
#define LED2G 49
//pin to control buzzer
int buzzPin = 2;
//Hardcoded note sequence for round over
int music[]={
  NOTE_C5,
  NOTE_D5,
  NOTE_E5,
  NOTE_F5,
  NOTE_G6,
  NOTE_A6,
  NOTE_B6,
  NOTE_C7};
//Hardcoded note sequence for game over
int gameover[] =
{NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NOTE_C5,NO
TE_C5};

int duration = 100; // 500 milliseconds}
```

```
CRGB leds[NUM_LEDS];//Structure fo LED control
int oldMillis = 0, newMillis = 0;
int digits[] = {0xBF, 0x86, 0xDB, 0xCF, 0xE6, 0xED, 0xFD, 0x87, 0xFF, 0xEF};
const byte segmentPins1[8] = {25,29,28,22,23,24,26,27};
const byte segmentPins2[8] = {35,39,38,32,33,34,36,37};
                          // DP, G, F, E, D, C, B, A
const byte numeral[10] = {
B11111100, // 0
B01100000, // 1
B11011010, // 2
B11110010, // 3
B01100110, // 4
B10110110, // 5
/*B00111110, // 6
B11100000, // 7
B11111110, // 8
B11100110, // 9*/
};
int i = 0, j = 0, k = 0;
//different directions the ball can go in
enum balldir{
  STILL,
  LEFTSTRAIGHT,
  RIGHTSTRAIGHT,
  LEFTUPSTEEP,
  LEFTUP45,
  //LEFTUPGRADUAL,
//  LEFTDOWNGRADUAL,
  LEFTDOWN45,
  LEFTDOWNSTEEP,
  RIGHTUPSTEEP,
  RIGHTUP45,
//  RIGHTUPGRADUAL,
//  RIGHTDOWNGRADUAL,
  RIGHTDOWN45,
  RIGHTDOWNSTEEP
};

//COLORS
//1 is background
//2 is paddles
//3 is ball
unsigned char board[16][16];
unsigned char LEDmapmatrix[16][16];
int p1paddlepos=0;
int p2paddlepos=12;
int ballcoords[2]={8,8};
int currentballdirection=RIGHTSTRAIGHT;
long randirection=0;
int speedfactor=100;

//1 is regular play
//2 is done playing
//3 is done playing and score counted
int PLAYING=1;

//0 is no winner
//1 is player 1 wins
//2 is player 2 wins
```

```
int WINNER=0;

int p1score=0; //p1 is right side
int p2score=0; //p2 is left side


void setup() {
  //sets LEDs as outputs and initializes them as off
  pinMode(LED1R,1);
  pinMode(LED1G,1);
  pinMode(LED2R,1);
  pinMode(LED2G,1);
  digitalWrite(LED1R,0);
  digitalWrite(LED2G,0);
  digitalWrite(LED1G,0);
  digitalWrite(LED2R,0);

  DDRB |= B01100000; //Sets Pins 11 (Data) and 12 (Clock) as OUTPUTS
  for(int i=0; i < 8; i++){
    pinMode(segmentPins1[i], OUTPUT);
    pinMode(segmentPins2[i], OUTPUT);
  }
  oldMillis = millis();  // Start timing
  Serial.begin(9600);
  randomSeed(analogRead(5));//Randomly seeds randomizer by taking a floating analog
value
  FastLED.addLeds<WS2812, LED_PIN, GRB>(leds, NUM_LEDS);//initializes LEDs for
future use
  //Creates the mapping sequence for LED matrix
  for (int i = 0; i <= 255; i++) {
    leds[i] = CRGB (0, 0, 10);
  }
  FastLED.show();
  for (int i = 0; i <= 15; i++){
    for (int j = 0; j <= 15; j++){
      board[i][j]=1;
    }
  }
  for (int j = 0; j <= 15; j++){
    if(j%2==0){
      for (int i = 0; i <= 15; i++){
      LEDmapmatrix[i][j]=i+j*16;
    }
    } else if (j%2==1) {
      for (int i = 15; i >= 0; i--){
      LEDmapmatrix[i][j]=(15-i)+j*16;
    }
    }
  }
  board321();
}

//Sets the LED color based on the value present at that point in the board matrix
void arrayToLEDs(unsigned char displayboard[16][16]){
  for (int i = 0; i <= 15; i++){
    for (int j = 0; j <= 15; j++){
      if(board[i][j]==1){
        leds[LEDmapmatrix[i][j]]=CRGB(0,0,5);//background color
      } else if (board[i][j]==2){
```

```
        leds[LEDmapmatrix[i][j]]=CRGB(100,100,100);//paddle color
      } else if (board[i][j]==3){
        leds[LEDmapmatrix[i][j]]=CRGB(100,100,0);//ball color
      }
    }
  }

  FastLED.show();//Shows LEDs
}
//Countdown sequence before the game starts
void board321(){
  //constructs number 3
  for (int i = 0; i <= 15; i++){
    for (int j = 0; j <= 15; j++){
      if(i==0 || i==8 || i==15 || j==15){
        leds[LEDmapmatrix[i][j]]=CRGB(10,0,0);
      } else {
        leds[LEDmapmatrix[i][j]]=CRGB(0,0,5);
      }
    }
  }
  FastLED.show();//shows number 3
  delay(1000);
  //constructs number 2
  for (int i = 0; i <= 15; i++){
    for (int j = 0; j <= 15; j++){
      if(i==0 || i==8 || i==15 || (j==15 && i<=8) || (j==0 && i>=8)){
        leds[LEDmapmatrix[i][j]]=CRGB(20,0,0);
      } else {
        leds[LEDmapmatrix[i][j]]=CRGB(0,0,5);
      }
    }
  }
  FastLED.show();//shows number 2
  delay(1000);
  //constructs number 1
  for (int i = 0; i <= 15; i++){
    for (int j = 0; j <= 15; j++){
      if(j==8){
        leds[LEDmapmatrix[i][j]]=CRGB(30,0,0);
      } else {
        leds[LEDmapmatrix[i][j]]=CRGB(0,0,5);
      }

    }
  }
  FastLED.show();//shows number 1
  delay(1000);
}

//creates paddles on the board
void refresh_paddles(int pos1,int pos2){
  int j=0;
  //creates player 1 paddle
  for (int i = 0; i <= 15; i++){
    if((i>=pos1) && (i<=pos1+3)){
      board[i][j]=2;
    } else {
      board[i][j]=1;
```

```
    }

  }
  j=15;
  //creates player 2 paddle
  for (int i = 0; i <= 15; i++){

    if((i>=pos2) && (i<=pos2+3)){
      board[i][j]=2;
    } else {
      board[i][j]=1;
    }
  }
}
//Reads the joystick value and accordingly adjusts the paddle position
void readJOY(){
  if((analogRead(p1joypin)<200)&&(p1paddlepos>0)){
    p1paddlepos-=2;
  } else if ((analogRead(p1joypin)>800)&&(p1paddlepos<12)) {
    p1paddlepos+=2;
  }
  if((analogRead(p2joypin)<200)&&(p2paddlepos>0)){
    p2paddlepos-=2;
  } else if ((analogRead(p2joypin)>800)&&(p2paddlepos<12)) {
    p2paddlepos+=2;
  }
}
//Deals with the movement of the ball
void refresh_ball(){
  //removes the previous position of the ball
  if(!(ballcoords[1]<=0 || ballcoords[1]>=15)){
    board[ballcoords[0]][ballcoords[1]]=1;
  }
  //Determines the next direction of the ball based on the previous direction
  switch(currentballdirection){
    //ball stays still and does nothing
    case STILL:
      break;
    case LEFTSTRAIGHT://previous direction
      ballcoords[1]-=1;//adjusts location
      if(board[ballcoords[0]][ballcoords[1]]==2){//generates random dirction
        randirection=random(2);
        tone(buzzPin, NOTE_C6, duration);//plays note once the paddle hits the ball
        if(randirection==0){
          currentballdirection=RIGHTUP45;
        } else if(randirection==1){
          currentballdirection=RIGHTDOWN45;
        }

      }
      break;
    case RIGHTSTRAIGHT:
      ballcoords[1]+=1;
      if(board[ballcoords[0]][ballcoords[1]]==2){
        randirection=random(2);
        tone(buzzPin, NOTE_A6, duration);
        if(randirection==0){
          currentballdirection=LEFTUP45;
        } else if(randirection==1){
```

```
      currentballdirection=LEFTDOWN45;
    }
  }
  break;
case LEFTUP45:
  ballcoords[1]-=1;
  ballcoords[0]-=1;
  if(ballcoords[0]<=0){
    currentballdirection=LEFTDOWN45;
  }
  if(board[ballcoords[0]][ballcoords[1]]==2){
    randirection=random(3);
    tone(buzzPin, NOTE_C6, duration);
    if(randirection==0){
      currentballdirection=RIGHTSTRAIGHT;
    } else if(randirection==1){
      currentballdirection=RIGHTUP45;
    } else if(randirection==2){
      currentballdirection=RIGHTUPSTEEP;
    }
  }

  break;
case LEFTDOWN45:
  ballcoords[1]-=1;
  ballcoords[0]+=1;
  if(ballcoords[0]>=15){
    currentballdirection=LEFTUP45;
  }
  if(board[ballcoords[0]][ballcoords[1]]==2){
    randirection=random(3);
    tone(buzzPin, NOTE_C6, duration);
    if(randirection==0){
      currentballdirection=RIGHTSTRAIGHT;
    } else if(randirection==1){
      currentballdirection=RIGHTDOWN45;
    } else if(randirection==2){
      currentballdirection=RIGHTDOWNSTEEP;
    }
  }
  break;
case RIGHTUP45:
  ballcoords[1]+=1;
  ballcoords[0]-=1;
  if(ballcoords[0]<=0){
    currentballdirection=RIGHTDOWN45;
  }
  if(board[ballcoords[0]][ballcoords[1]]==2){
    randirection=random(3);
    tone(buzzPin, NOTE_A6, duration);
    if(randirection==0){
      currentballdirection=LEFTSTRAIGHT;
    } else if(randirection==1){
      currentballdirection=LEFTUP45;
    } else if(randirection==2){
      currentballdirection=LEFTUPSTEEP;
    }
  }
  break;
```

```
case RIGHTDOWN45:
  ballcoords[1]+=1;
  ballcoords[0]+=1;
  if(ballcoords[0]>=15){
    currentballdirection=RIGHTUP45;
  }
  if(board[ballcoords[0]][ballcoords[1]]==2){
    randirection=random(3);
    tone(buzzPin, NOTE_A6, duration);
    if(randirection==0){
      currentballdirection=LEFTSTRAIGHT;
    } else if(randirection==1){
      currentballdirection=LEFTDOWN45;
    } else if(randirection==2){
      currentballdirection=LEFTDOWNSTEEP;
    }
  }
  break;
case LEFTUPSTEEP:
  ballcoords[1]-=1;
  ballcoords[0]-=2;
  if(ballcoords[0]<0){
    ballcoords[0]=0;
  }
  if(ballcoords[0]==0){
    currentballdirection=LEFTDOWNSTEEP;
  }
  if(board[ballcoords[0]][ballcoords[1]]==2){
    randirection=random(3);
    tone(buzzPin, NOTE_C6, duration);
    if(randirection==0){
      currentballdirection=RIGHTSTRAIGHT;
    } else if(randirection==1){
      currentballdirection=RIGHTUP45;
    } else if(randirection==2){
      currentballdirection=RIGHTUPSTEEP;
    }
  }
  break;
case LEFTDOWNSTEEP:
  ballcoords[1]-=1;
  ballcoords[0]+=2;
  if(ballcoords[0]>15){
    ballcoords[0]=15;
  }
  if(ballcoords[0]==15){
    currentballdirection=LEFTUPSTEEP;
  }
  if(board[ballcoords[0]][ballcoords[1]]==2){
    randirection=random(3);
    tone(buzzPin, NOTE_C6, duration);
    if(randirection==0){
      currentballdirection=RIGHTSTRAIGHT;
    } else if(randirection==1){
      currentballdirection=RIGHTDOWN45;
    } else if(randirection==2){
      currentballdirection=RIGHTDOWNSTEEP;
    }
  }
```

```
    break;
  case RIGHTUPSTEEP:
    ballcoords[1]+=1;
    ballcoords[0]-=2;
    if(ballcoords[0]<0){
      ballcoords[0]=0;
    }
    if(ballcoords[0]==0){
      currentballdirection=RIGHTDOWNSTEEP;
    }
    if(board[ballcoords[0]][ballcoords[1]]==2){
      randirection=random(3);
      tone(buzzPin, NOTE_A6, duration);
      if(randirection==0){
        currentballdirection=LEFTSTRAIGHT;
      } else if(randirection==1){
        currentballdirection=LEFTUP45;
      } else if(randirection==2){
        currentballdirection=LEFTUPSTEEP;
      }
    }
    break;
  case RIGHTDOWNSTEEP:
    ballcoords[1]+=1;
    ballcoords[0]+=2;
    if(ballcoords[0]>15){
      ballcoords[0]=15;
    }
    if(ballcoords[0]==15){
      currentballdirection=RIGHTUPSTEEP;
    }
    if(board[ballcoords[0]][ballcoords[1]]==2){
      randirection=random(3);
      tone(buzzPin, NOTE_A6, duration);
      if(randirection==0){
        currentballdirection=LEFTSTRAIGHT;
      } else if(randirection==1){
        currentballdirection=LEFTDOWN45;
      } else if(randirection==2){
        currentballdirection=LEFTDOWNSTEEP;
      }
    }
    break;
}


if((ballcoords[1]<=0) && board[ballcoords[0]][ballcoords[1]]==1){
  //checks if player 2 wins
  WINNER=2;
  PLAYING=2;
  currentballdirection=STILL;
  board[ballcoords[0]][ballcoords[1]]=3;
} else if ((ballcoords[1]>=15) && board[ballcoords[0]][ballcoords[1]]==1){
  //checks if player 1 wins
  WINNER=1;
  PLAYING=2;
  currentballdirection=STILL;
  board[ballcoords[0]][ballcoords[1]]=3;
} else {
```

```
    board[ballcoords[0]][ballcoords[1]]=3;
  }


}
//controls behavior once round ends
void update_score_reset(){
  if(PLAYING==1){
    if(i>=1){
      write(digits[0], digits[i], digits[j], digits[k]);
      PLAYING=3;
      if(p1score>p2score){
        digitalWrite(LED1R,1);
        digitalWrite(LED2G,1);
      } else if (p1score<p2score){
        digitalWrite(LED1G,1);
        digitalWrite(LED2R,1);
      } else if (p1score==p2score){
        digitalWrite(LED2R,1);
        digitalWrite(LED1R,1);
      }
      currentballdirection=STILL;
    }
  }
  if(PLAYING==2){
    Serial.println(digitalRead(7));
    //changes player score based on last round winner
    PLAYING=3;
    if(WINNER==1){
      p1score+=1;
      digitalWrite(LED1R,1);
      digitalWrite(LED2G,1);
    } else if(WINNER==2){
      p2score+=1;
      digitalWrite(LED1G,1);
      digitalWrite(LED2R,1);
    }
  } else if (PLAYING==3){
    if(p1score>=5 || p2score>=5){
      for (int noteIndx = 0; noteIndx < 10; noteIndx++) {
          tone(buzzPin, gameover[noteIndx], duration); // Output the music
          delay(100);
        }
    } else {
    for (int noteIndx = 0; noteIndx < 8; noteIndx++) {
      tone(buzzPin, music[noteIndx], duration); // Output the music
      delay(100);
    }
    }
    PLAYING=4;
  } else if (PLAYING==4){
    //Round reset, does basically the same things as setup
    if((digitalRead(7)==0) && (digitalRead(8)==0)){
      digitalWrite(LED1G,0);
      digitalWrite(LED2R,0);
      digitalWrite(LED1R,0);
      digitalWrite(LED2G,0);
      speedfactor=100;
      Serial.println("check1");
```

```
      PLAYING=1;
      p1paddlepos=0;
      p2paddlepos=12;
      ballcoords[0]=8;
      ballcoords[1]=8;
      currentballdirection=RIGHTSTRAIGHT;
      //Ends the full game if the score goes above 5
      for (int i = 0; i <= 15; i++){
        for (int j = 0; j <= 15; j++){
          board[i][j]=1;
        }
      }
      if(p1score >= 5){
        board321();
        p1score = 0;
        p2score = 0;
        i = 0;
        j = 0;
        k = 0;
      }
      else if(p2score >= 5){
        board321();
        p1score = 0;
        p2score = 0;
        i = 0;
        j = 0;
        k = 0;
      } else if(i>=1) {
        board321();
        p1score = 0;
        p2score = 0;
        i = 0;
        j = 0;
        k = 0;
      }

    }
  }
  }

void startClock() {
    //PORTB = PORTB | B01100000; //Set both Clock and Data Pin HiGH
    PORTB = PORTB | B00100000; //Set Data Pin HiGH
    delayMicroseconds(2);
    PORTB = PORTB | B01000000; //Set Clock Pin HiGH
    delayMicroseconds(2);
    PORTB &= ~B00100000; //LOW Clock Pin
    delayMicroseconds(2);
    PORTB &= ~B01000000; //LOW Data Pin
    delayMicroseconds(2);
}

void stopClock() {
    PORTB &= ~B01000000; //LOW Clock Pin
    delayMicroseconds(2);
    PORTB &= ~B00100000; // LOW Data Pin
    delayMicroseconds(2);
    PORTB = PORTB | B01000000; //HIGH Clock Pin
    delayMicroseconds(2);
```

```
    PORTB = PORTB | B00100000; // HIGH Data Pin
    delayMicroseconds(2);
}

void writeValue(int value) {
    for (int i = 0; i < 8; i++) {
        PORTB &= ~B01000000; //LOW Clock Pin
        if (value & (1 << i)) {
          PORTB = PORTB | B00100000; // HIGH Data Pin
        } else {
          PORTB &= ~B00100000; // LOW Data Pin
        }
        PORTB = PORTB | B01000000; //HIGH Clock Pin
    }
    PORTB &= ~B01000000; //LOW Clock Pin
    PORTB = PORTB | B01000000; //HIGH Clock Pin
}

void write(int first, int second, int third, int fourth) {
    startClock();
    writeValue(0x40);
    stopClock();

    startClock();
    writeValue(0xc0);
    writeValue(first);
    writeValue(second);
    writeValue(third);
    writeValue(fourth);
    stopClock();

    startClock();
    writeValue(0x8F);  // Display ON, max brightness
    stopClock();
}

void gimmeFive() {
    newMillis = millis();
    if (newMillis - oldMillis >= 1000) {
        oldMillis = newMillis;
        write(digits[0], digits[i], digits[j], digits[k]); // Display time
        k++;
        if (k == 10) {
            k = 0;
            speedfactor-=30;
            j++;}
        if (j == 6) {
            j = 0;

            i++;}
        if (i == 1) {
            k = 0;
            j = 0;}
    }
}
void showDigit(int number, byte segmentPins[8]){
  boolean isBitSet;
  for(int segment = 1; segment < 8; segment++){
    if(number < 0 || number > 9){
```

```
      isBitSet = 0; // turn off all segments
    }
    else{
    // isBitSet will be true if given bit is 1
      isBitSet = bitRead(numeral[number], segment);
      // bitRead() returns the value of the bit as either 0 or 1
    }
    digitalWrite(segmentPins[segment], isBitSet);
  }
}

void loop() {

  arrayToLEDs(board);
  if(PLAYING==1){
    readJOY();
    refresh_paddles(p2paddlepos,p1paddlepos);
    gimmeFive();
  }
  showDigit(p1score,segmentPins1);
  showDigit(p2score,segmentPins2);
  refresh_ball();
  update_score_reset();

  delay(100+speedfactor);

}
```