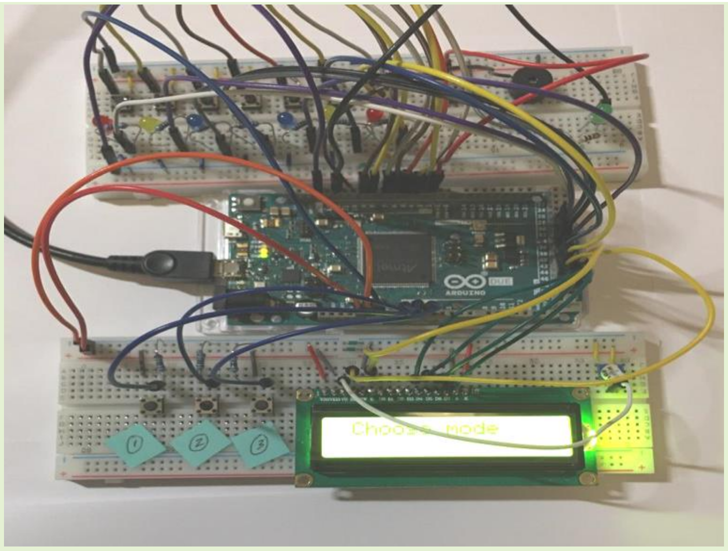
Musical Piano, Tune Recorder and Memory Game

Abhirami Venugopal and 216940256



Abstract— Using the Arduino DUE I created a mina replica of a musical piano. The main part of the piano are it's keys that are implemented using push buttons. The piano has 3 modes, and the user is able to navigate between the three different modes using a push button. The LCD display also provides the user with relevant instructions and shows the user which mode he/she is in. The first mode of the piano allows the user to play any tune he/she wants to. It can be noted that the user is alerted that he/she is in recording mode if the big green LED is lit up. This is similar to recording studios that have green signs in the rooms that light up when recording sessions are taking place. The second mode allows the user to listen to the tune he/she played before. Thus, if the user composes an interesting musical piece, he/she will be able to listen to it again. The final mode is a game mode. In this game mode the memory of the user is tested. There are seven different levels of increasing difficulty. The user can get one pattern wrong without losing the entire game.
The project can be continued and made more refined by sharpening the tones produced by the buzzer. The game mode can be developed to also keep track of the speed at which the user is able to press the buttons from memory.

Index Terms—Debounce, Tone, Interrupt Service Routine, Mode 1, Mode 2, Mode 3

Introduction

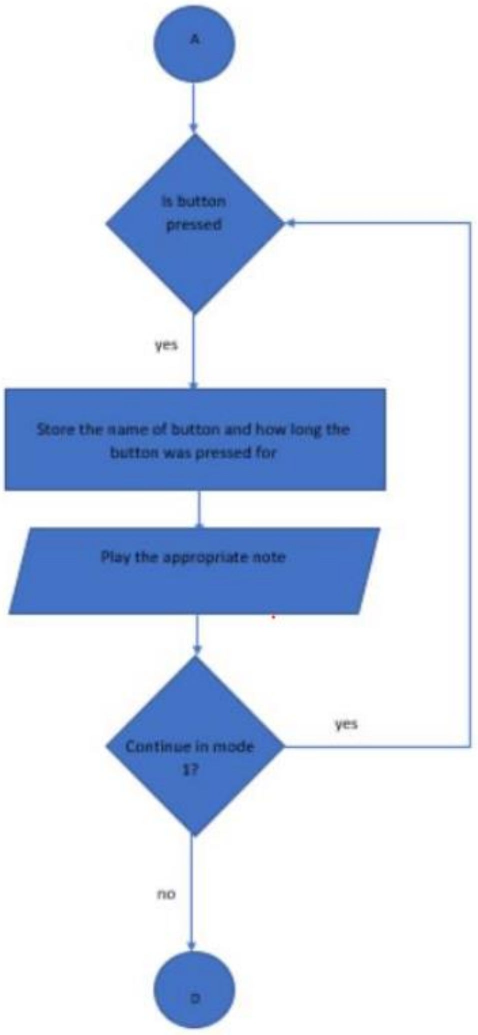
When starting the project, the main intention was to implement an electronic musical instrument. Electronic keyboards can reproduce the sounds created by several instruments like piano, Hammond organ, pipe organ, violin, etc. However, the scope of this project only covers sounds made by a piano. My younger sister owns an electronic keyboard and I found it very fascinating. It was fulfilling to have created a replica that produces notes of the correct octave and step.
The biggest advantage of my project and it's competitive edge is that extremely cheap to manufacture. This would be most suitable for beginners who would like to experiment with cheap keyboard before investing in a more expensive version. This project is a good tool to develop the memory of children as well. The project is also very light-weight.
The challenge is that several manufacturers already produce similar products (without the game mode of course). The packaging of the project need to appealing and portable.

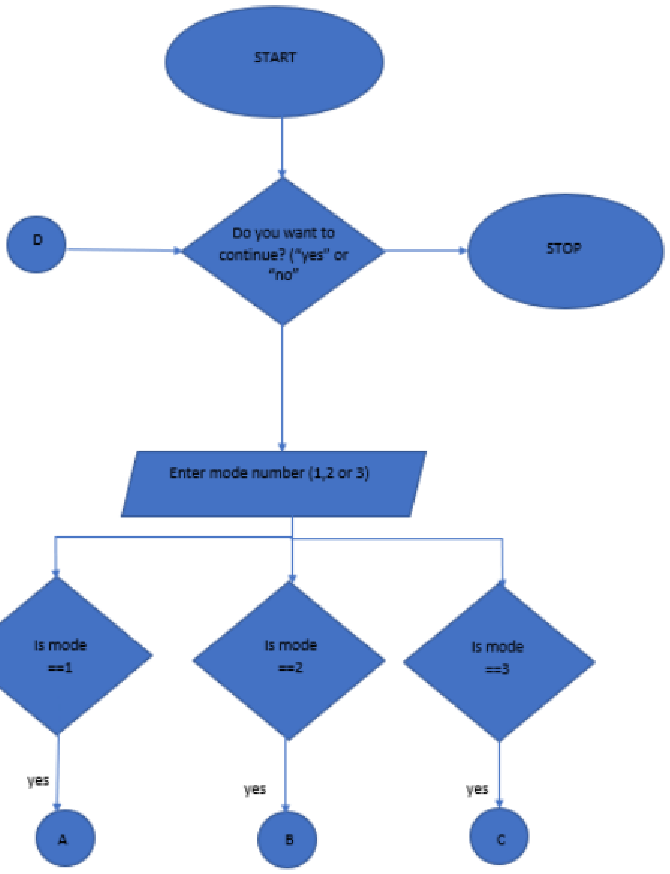
I. Proposed Project

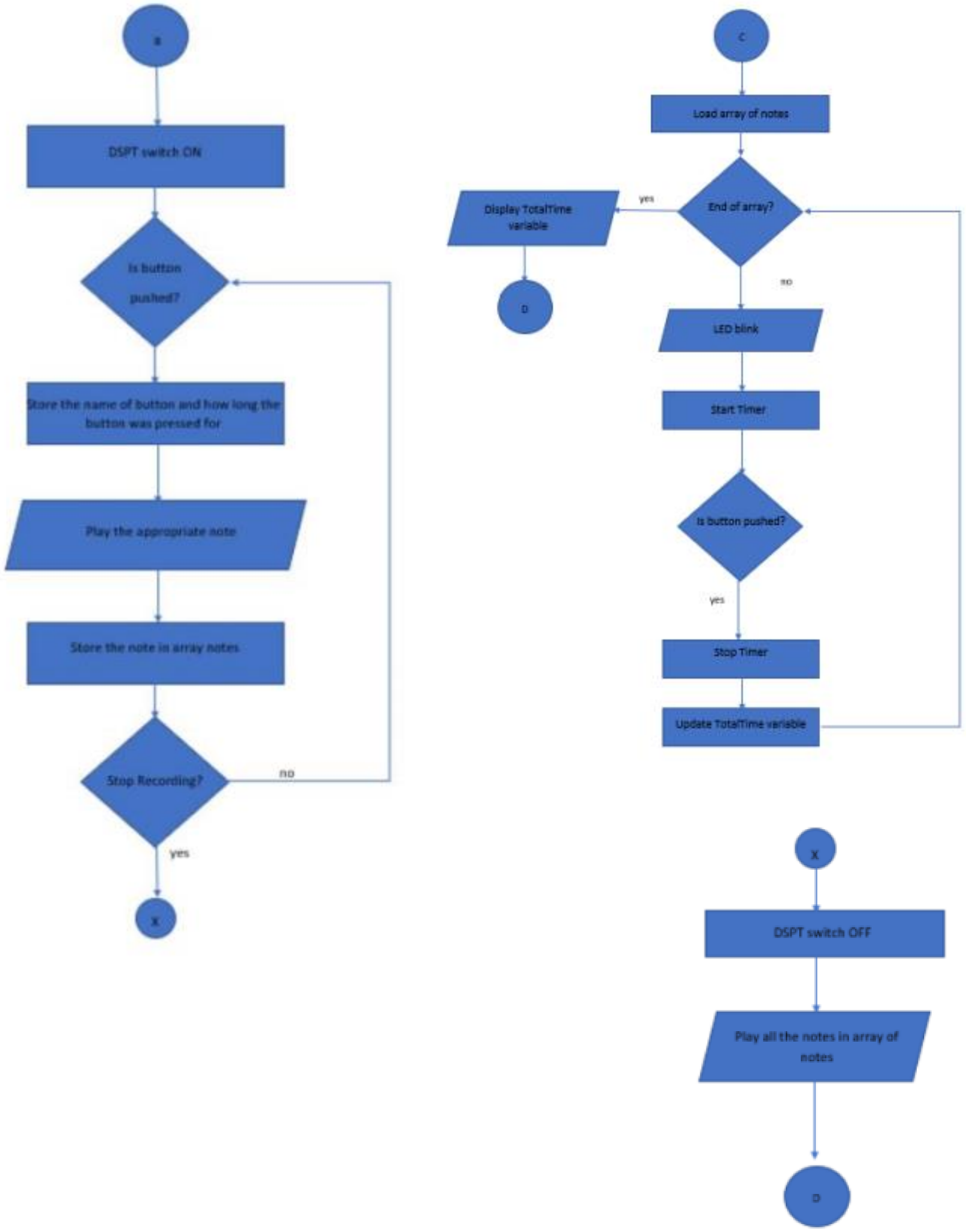
This section will give readers an in-depth understanding of the hardware and software used in the project.

A. Program

Flowchart


A
Is button pressed
yes
Store the name of button and how long the button was pressed for
Play the appropriate note
Continue in mode 1?
yes
no
D


START
D
Do you want to continue? ("yes" or "no"
STOP
Enter mode number (1,2 or 3)
Is mode ==1
Is mode ==2
Is mode ==3
yes
yes
yes
A
B
C


B
DSPT switch ON
Is button pushed?
Store the name of button and how long the button was pressed for
Play the appropriate note
Store the note in array notes
Stop Recording?
no
yes
X
C
Load array of notes
End of array?
yes
Display TotalTime variable
D
no
LED blink
Start Timer
Is button pushed?
yes
Stop Timer
Update TotalTime variable
X
DSPT switch OFF
Play all the notes in array of notes
D

Algorithm

Controller

Initially the user is welcomed by a message on the LCD Display. The user is then prompted to choose a mode. Based on the analog input received from pins A0, A1 or A2 control goes to different portions of the code. This is achieved using if statements. Throughout the program the LCD screen is used to display appropriate messages to the user.

Mode 1: Recording Mode

The user has full freedom to press any button. All buttons pressed will be stored in an array in order so that it can be accessed later to be replayed. Once a tune is played out, the array empties itself so that a new tone can be played.
The first condition that is checked is whether the playback mode is switched on, we are double checking that it is indeed mode 1 and not mode 2. Using a for loop we check which button is switched on at a particular moment. One point to note is that at one time, only one button can be pressed down.

Mode 2: Playback Mode

Once the user is ready to playback the tune that he/she constructed using the noted outputted by the seven keys, then mode 2 is called. In mode 2, the array that contains the sequence of notes that the user played in mode 1 is used. The array is iterated through, and the noted stored are played in order.

Mode 3: Game Mode

Once the user is in the game mode, he/she is given 3 lives and starts at level 1. The user can make 3 mistakes in the sequence without losing the entire game. The user starts at level 1 and can progress up to level 7. At each level, the number of notes in the sequence increase from 3 up till 10. The duration for which each note is played and the time interval between each node reduces from level 1 to level 7.

Whenever the user gets the sequence correct, the LEDs blink in a right to left pattern and a victory tune is played. Whenever the user gets the sequence wrong, the LEDs blink all at once and a failure tune is played. A different tune is played when the player wins all the rounds or loses the game by losing all the lives.

Once the user wins or loses the entire game, he/she is taken to the main menu.

Methods and Descriptions

Mode 2: Playback Mode

void playback()
This method allows us to playback the tune that the user recorded while in recording mode. We are reading through the array called button_seq[] that contains all the buttons that were pressed in the recording mode in the correct order

Mode 3: Game Mode

void generateSequence()
Generates a random sequence of notes for each level. This ensures that the user gets a different sequence of notes every time he/she plays the game. Thus, the piano is a good game and does not become predictable.

void playSequence(int notes[])
This method plays the array of notes that it is given. It is called when the piano wants to let the user listen to the tune he/she must replicate
@param notes the array of notes that contains the correct sequence

void verifyNote(int note)
This method is used to verify is the button that user presses if the correct button in the
sequence.

void playNote(int note)
This method allows the piano to play notes to the user, it sets the note duration and the interval between the notes according to the level of difficulty. Shorter durations and shorter interval between notes as the
difficulty level increases

void playButtonNote(int note)
Makes the buzzer play the tone, and the LED behind light up. This method is called when the user presses a button.
@param note the button the user presses

void win()
The tune that plays when the user wins a level and progresses onto the next

void lose()
The tune that is played when the user loses a level by pressing a wrong button. This is not
what is played if the player loses all his/her lives and loses the entire game.

void final_win()
Plays a tune when user has completed all the levels successfully

void final_lose()
Plays a tune when the player uses the entire game

Generation of Tones

void tone(uint32_t ulPin, uint32_t frequency, int32_t duration)
This method generates a tone through a buzzer if the frequency and the duration of the note is passed to it.
@param ulPin the pin number of the buzzer, @frequency the frequency of the note to be played, @duration the duration of the note

void noTone(uint32_t ulPin)
This method ensures that the buzzer is not making noise anymore.
@ulPin the pin number of the buzzer

void TC3_Handler (void)
Handler for the Interrupt Service Routine

References used and Original Code
The code for the record and playback (mode 1 and mode 2) uses the code on circuit digest as its base. [1]
In this code, I have changed the frequencies they used for each of the notes and instead I included a file called "pitches.h" that contains the mapping of different notes to their respective frequencies. [2]

I also did not use their function to record the amount of time the button is pressed for and instead hardcoded the durations in my own array called noteDurations[]. I introduced all the debounce variables to counter the bouncing of the buttons because I was not recording the durations. In addition, I was not able to use the tone() and noTone() methods which are already defined in a library for the Arduino UNO (this website shows code only for UNO). Thus I researched and found methods to suit my needs and modified them accordingly. [3]

For the game mode, I used code exerpts from the Arduino Project Hub [4]. However once again I had to modify the code to suite Arduino Due and also integrate this project into the already existing mode 1 and mode 2. I tried several other ways to implement the game mode before this one. In this website, the code was only implemented for 2 buttons, thus I had to clearly read and understand the code before duplicating it in hardware and software. This was a time-consuming process. In addition all comments were written in a foreign language, thus I had to interpret the code using my own knowledge.

B. *Hardware*

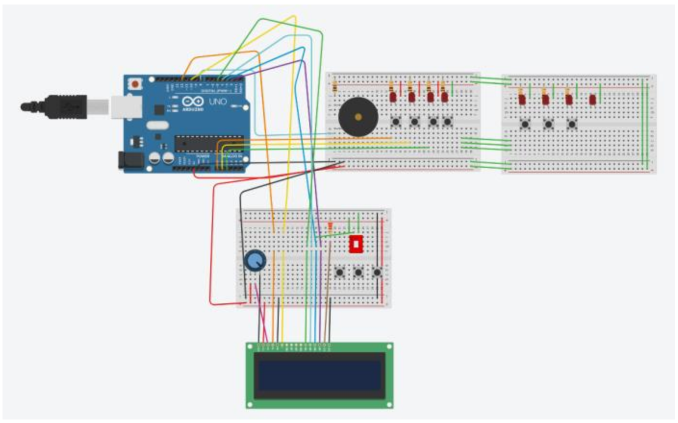


Figure 1: Circuit Schematic

1. **Arduino Due**
The microcontroller has several functions in this project. It is able to recognize when the user presses one of the push buttons and also tell the program which button is specifically is pressed. This allows the buzzer to output the right frequency tone. It also allows the piano to light up the LEDs behind the correct button in mode 3. It allows the LCD display to display messages appropriate to each level.

2. **16*2 LCD Display**
This display allows the project to present users with meaningful messages. First the users are welcomed. I think this is an important component of the user experience. Then the user gets to choose which mode he/she wants to go into. Then, the LCD displays whichever mode the user is in, for reference. The user may want to change his/her mind and this prevent them from being confused about which mode they are in.

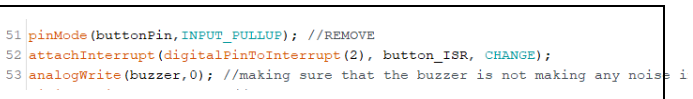
In the game mode, the LCD shows the users how many lives they have left and also which level they are on. It also shows appropriate messages when users fail or pass a particular level , when the lose the entire game and when they win the entire game.

3. **10k ohm Potentiometer**
The potentiometer is used to adjust the bias level of the LCD, aka the contrast. Without it, my LCD display remained blank.

4. **Piezo Buzzer**
The piezo buzzer is one of the most important components of the project. It allows the project to play notes and tunes. When the buzzer is given the right frequency it can play a variety of notes of different frequencies. It's volume can be adjusted using a potentiometer. In this particular project, I was comfortable with the volume of my buzzer thus I did not use the potentiometer to adjust it.

5. **Push Buttons**
In this project, the push buttons are used for 2 different reasons. The first use is to replicate the keys of a piano. When one of the 7 push buttons are pressed, a note of a particular frequency is emitted. In mode 2, we detect which button is pressed and compare it with the expected value based on the randomly generated sequence to verify if the tune played by the user is correct. The second use is for users to set the mode. The 3 buttons next to the LCD display are associated with if-else statements in the code that allow this.

An interesting note is that I had initially intended to switch between modes using an SPDT switch, however the specifications of the SPDT switch I bought were not compatible with the Arduino Due. This the way I did it, is a substitute for using an SPDT switch. I tried implementing the switching of modes using interrupt service routines, however that worked only for 2 modes as I was using a Boolean variable and setting it to true and false in the handler method. I had also considered using SSH in PuTTY to switch between the modes.



```
pinMode(buttonPin,INPUT_PULLUP); //REMOVE
attachInterrupt(digitalPinToInterrupt(2), button_ISR, CHANGE);
analogWrite(buzzer,0); //making sure that the buzzer is not making any noise i
```

Figure 2: Attempt to implement Modes using Interrupts

I realized that connecting the components in a ordered manner allowed me to view the connections better and fix any loose connections. Use wires of the appropriate size also helped with the cleanliness of the project. Using 10k ohm resistors with the LEDs ensured that I did not damage any of them.

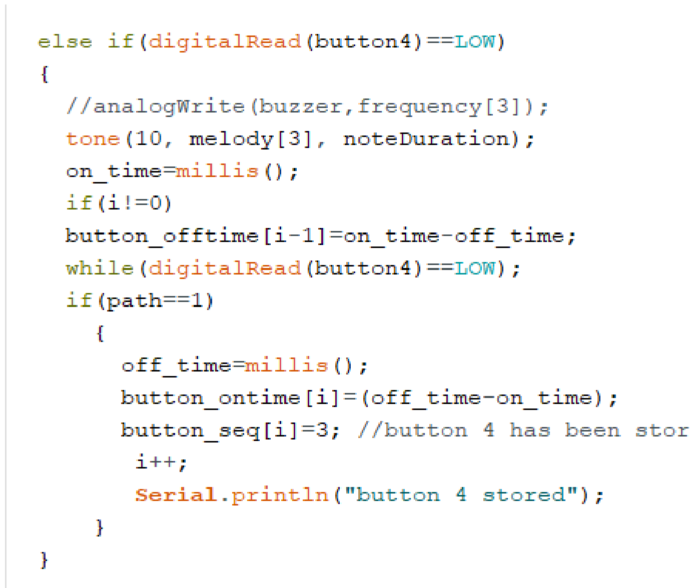
II. Results

The process of fabrication of this project started with learning about the Arduino Due, and reading it's Data Sheet. I needed a clear understanding of the pins available and also the libraries available before selecting a project. Once I learned about the micro-controller then I decided that I wanted to do a project related to music and thus searched for relevant libraries. I got inspiration from a much more complicated project called Piano Tutor (https://create.arduino.cc/projecthub/tcucinotta/arduino-leds-midi-keyboard-musescore-piano-tutor-9080fe?ref=tag&ref_id=music&offset=54). I decided to make a simpler project and researched for all the parts required. Once I ordered the parts online, I spent the time waiting for them to be delivered by learning how to test each component individually. This ensured that once I received my parts, I was able to quickly assemble them without too many roadblocks. I was successfully able to implement all the functionalities that I had aimed for.
The piano has 3 modes, and the user can navigate between the three different modes using a push button. The LCD display also provides the user with relevant instructions and shows the user which mode he/she is in. The first mode of the piano allows the user to play any tune he/she wants to. The second mode allows the user to listen to the tune he/she played before. Thus, if the user composes an interesting musical piece, he/she will be able to listen to it again. The final mode is a game mode. In this game mode the memory of the user is tested. There are seven different levels of increasing difficulty. The user can get one pattern wrong without losing the entire game. The project can be continued and made more refined by sharpening the tones produced by the buzzer. The game mode can be developed to also keep track of the speed at which the user is able to press the buttons from memory.
This product would be mainly be produced as an educational supplement for children between the ages of three and eight. It can be used for children to improve their cognitive abilities. Pressing buttons and listening to the corresponding sounds have been proven to strengthen the nerve fibres connected the left and right sides of the brain. This will lead to children developing better problem-solving and language skills. The first mode would allow the children to develop their creativity and self-expression. It would allow children to develop their own ideas and compositions. The second mode allows children to playback their composition, which would give them a sense of accomplishment. The third mode aids in improving the memory. The children would have to memorize the sequence and play it back. Since the notes become faster as the levels get harder, their reflex skills are also improved. In order to ensure that the children don't get discouraged, the project implements three lives/tries so that the child does not lose the entire game by getting one sequence wrong. Every time the user gets a sequence correct a victory tune is played, which reinforces positive action and is instant gratification. This leads the children to have a sense of accomplishment while playing the game.

III. Discussion

One of the limitations of the first mode is that only one button can be pressed down at one time. Unlike other electronic keyboard available in the market. This does limit the users in terms of the musical pieces that he/she can play. This is the case, as I used if-else statements to check whether a button is pressed or not, so only one code block is entered at a time. This structure was required as I was unable to store multiple notes with the same rank using a simple primitive array. Perhaps, the future model of this project can use an advanced data structure where simultaneous notes can be recorded and played.

Another way to improve mode 1 would be to record the duration for which the button is pressed down. Currently, the duration of the note is hardcoded in both mode 1 and 2. I did try implementing this function; however, I was not able to make it work successfully. The following is a screenshot of the trial. I tried to start recording the time using the millis() function. Then I recorded the time that the button was not being pressed anymore using the millis() function. The difference is calculated and this is stored as the duration of the note. With slight improvements, this will work.



```
else if(digitalRead(button4)==LOW)
{
  //analogWrite(buzzer,frequency[3]);
  tone(10, melody[3], noteDuration);
  on_time=millis();
  if(i!=0)
  button_offtime[i-1]=on_time-off_time;
  while(digitalRead(button4)==LOW);
  if(path==1)
    {
      off_time=millis();
      button_ontime[i]=(off_time-on_time);
      button_seq[i]=3; //button 4 has been stor
       i++;
       Serial.println("button 4 stored");
    }
}
```

Figure 3: Attempt to record the duration of button press

A technical issue that I faced is not having a soldered LCD display. As I did not have a soldering iron initially, I tried to simply connect the pins of the LCD display to the wires directly. However, this cause lose connection and resulted in the LCD display to only display some characters and not all and behave unexpectedly. Finally, I got the LCD display soldered to header pins and attached it to the breadboard. This allowed me to make the LCD work properly.

To switch between the different modes in the piano was one of the most challenging aspects. Initially I had decided to use an SPDT(Single Pole- Double Throw) switch to move between mode 1 and mode 3. I intended to combine mode 2 and 1.

After a note, or a sequence of notes are played, it is good practice to use noTone() to ensure that the buzzer does not create random noises. I noticed that when I did not use this method, in addition to the note(s) I was playing, random noises were heard. Initially I has used analogWrite() to make the buzzer create a sound, I did not move forward with it as it did not allow me to set the frequency of the note being played, thus all notes sounded the same.

IV. Conclusion

The main objective of this project was to implement a musical keyboard that produces the notes produced by seven keys of a piano. When creating this project, I mainly intended it to be used by children between the ages of three and eight as an educational tool. Several research papers have shown that music helps strengthen the nerve connections between the left and right side in children. Music has been proven to increase the cognitive abilities of children, especially young children. The first mode would allow the children to develop their creativity and self-expression. It would allow children to develop their own ideas and compositions. The second mode allows children to playback their composition, which would give them a sense of accomplishment. The third mode aids in improving the memory. The children would have to memorize the sequence and play it back. Since the notes become faster as the levels get harder, their reflex skills are also improved. Building the project during the COVID-19 pandemic has been challenging due to a few factors. The first being unable to select the parts in-person. I had to order all my parts online and made a mistake while buying the SPDT switch as the data sheet for it was not provided on Amazon. I do not have access to hardware stores as I am currently not in Canada, and my country does not have many hardware stores that sell products compatible with the Arduino. In addition, once the parts were ordered, I encountered several delays in receiving the parts. Another challenge that I faced is the lack of space to work on my project. Since I live with three other family members in a small apartment, I was not able to build a project that required a lot of space as it would have been inconvenient for me to keep carrying it around the apartment. Another challenge that I faced was not being able to solder parts within my apartment and not being able to go out to get it soldered due to shops closing because of the pandemic, so I got my LCD display soldered very late after COVID cases decreased in my city.

Even though I faced all these challenges, I was able to complete my project on time by working on it little by little and working with the components that arrived earlier than others. While waiting for the parts to arrive, I did all the research required so that once I got the parts I could immediately start working on my project. Even though I am in a different time zone, I worked on my project whenever my family was awake so that I would not wake them up by the random, loud buzzer noises. Instead of causing a fire hazard by attempting to solder within my small, closed apartment, I chose to wait till the COVID cases in my city decreased an got my LCD display soldered to the header pins in a repair shop.

Acknowledgment

I would like to thank Professor Ebrahim Ghafar-Zadeh for introducing me to the Arduino Due board and helping me understand the datasheet. I would like to the TA assigned to my tutorial group Mr. Fayas Ahamed Mohamed Hasan for encouraging me throughout the process of building the project. He provided me with support even after the allotted tutorial time and motivated me to perform the best I can. Finally I would like to thank my peers Mohammed Fulwala , Syed Quadri and Faruq Afolabi for the interesting discussions we had.

Abhirami Venugopal is a second year Software Engineering student at York University. She is currently in the Big Data stream as she enjoys working with large data sets. From 2017 onwards she staring coding. She initially started coding in C++ and made several projects in the language, include a quiz game and a library management system. She then went on to learn mySQL understood the basics of database management. From 2019 she has been coding in Java, and has built a tab text to musicXML application that is user-friendly and well-documented. In that project as well, she needed to learn about music theory and frequencies of different notes. She does have an inclination towards using technology in the music industry.
From 2008 to 2009, he was a Research Assistant with the Institute of Physics, Academia Sinica, Tapei, Taiwan. His research interest includes the development of surface processing and biological/medical treatment techniques using nonthermal atmospheric pressure plasmas, fundamental study of plasma sources, and fabrication of micro- or nanostructured surfaces.
Mr. Author's awards and honors include the Frew Fellowship (Australian Academy of Science), the I. I. Rabi Prize (APS), the European Frequency and Time Forum Award, the Carl Zeiss Research Award, the William F. Meggers Award and the Adolph Lomb Medal (OSA).

REFERENCES

[1] A. Raj, "Circuit Digest," Circuit Digest, 12 June 2018 . [Online]. Available: https://circuitdigest.com/microcontroller-projects/arduino-piano-with-record-and-playback. [Accessed March 2021].

[2] M. Putnam, "GitHub Gist," GitHub, 9 June 2012. [Online]. Available: https://gist.github.com/mikeputnam/2820675. [Accessed 2021].

[3] Mantoui, "Arduino Forum," Arduino , 12 November 2018. [Online]. Available: https://forum.arduino.cc/t/tone-cpp-todo-dueless/127639. [Accessed February 2021].

[4] R. Groig, "Arduino Project Hub," Arduino , 27 June 2020. [Online]. Available: https://create.arduino.cc/projecthub/raulgroig/genius-game-517648. [Accessed February 2021].

CODE

```
#include <LiquidCrystal.h> //including this library since we are using an LCD display

#include <Bounce2.h> //this library allows us to prevent the bounce of buttons
#include "pitches.h" //this file contains global variables, these variable indicate the various notes in the musical scale of a piano

//Define the LEDs pins
#define LED1 30
#define LED2 32
#define LED3 34
#define LED4 36
#define LED5 38
#define LED6 40
#define LED7 42

// Defining the buttons pins
#define BUTTON1 9
#define BUTTON2 8
#define BUTTON3 7
#define BUTTON4 6
#define BUTTON5 5
#define BUTTON6 4
#define BUTTON7 3

//Defining the Buzzer pin
#define BUZZER 10

// Instantiate Bounce objects, these are used to debounce the push buttons
Bounce debouncer1 = Bounce();
Bounce debouncer2 = Bounce();
Bounce debouncer3 = Bounce();
Bounce debouncer4 = Bounce();
Bounce debouncer5 = Bounce();
Bounce debouncer6 = Bounce();
Bounce debouncer7 = Bounce();

//Change this variable to increase number of notes to play
//int noteSequence[4];
//int numNotes =  sizeof(noteSequence)/sizeof(int);

/*
 * The game contains 7 different difficulty levels
 */
int noteSequence_1[3]; //level 1: contains 3 notes
int noteSequence_2[4]; //level 2: contains 4 notes
int noteSequence_3[5]; //level 3: contains 5 notes
int noteSequence_4[6]; //level 4: contains 6 notes
int noteSequence_5[7]; //level 5: contains 7 notes
int noteSequence_6[8]; //level 6: contains 8 notes
int noteSequence_7[9]; //level 7: contains 9 notes

int numNotes =  3; //keeps track of which level we are on

int currentNote = 0; //keeps track what the current note is
int lives = 3; //the player starts with 3 lives; this means that
he/she has 3 chances before they lose the game entirely
boolean isPlaying = false; //keeps track

/*These buttons allow us to change the modes of the piano*/
const int button1 = A0;
const int button2 = A1;
const int button3 = A2;

/*The LCD display is initialized here*/
const int rs = 46, en = 47, d4 = 48, d5 = 49, d6 = 50, d7 = 51;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// We will be using "countx" to keep track of how many times
a button is pressed
int count1 = 0;
int count2 = 0;
int count3 = 0;

// "statex" will keep track of the state of the button. Was it just
pressed?
int state1 = 0;
int state2 = 0;
int state3 = 0;

// These keep tabs on the previous state of the button. Was it
high? Low?
int prev1 = 0;
int prev2 = 0;
int prev3 = 0;

int val=0;

unsigned long on_time=0;
unsigned long off_time=0;
unsigned long button_ontime[20];
unsigned long button_offtime[20];

int button_seq[20];
int melody[] = {NOTE_A4, NOTE_B4, NOTE_C5,
NOTE_DS5, NOTE_E5, NOTE_G5, NOTE_F6};

int previousState = HIGH;
unsigned int previousPress;
volatile int buttonFlag;
int buttonDebounce = 20; //this variable is used to help
debounce the push buttons used

int noteDurations[] = {10, 10, 10, 10, 10, 10, 10, 10};

int path=1;
int i=0;
int ledr=12; //REMOVE
void playback (void); //the method signature for the playback
feature of the project


void button_ISR()
{
  buttonFlag = 1;

}

void setup() {
  // put your setup code here, to run once:
  lcd.begin(16, 2); //setting up the lcd display

  lcd.setCursor(0, 0); //setting the cursor so that the output
looks good
  lcd.print("Welcome"); //welcoming the user
  delay(1000);
  lcd.clear();

  //setting all the mode buttons as input, when pressed they
allow user to skip between different modes
  pinMode(button1, INPUT_PULLUP);
  pinMode(button2, INPUT_PULLUP);
  pinMode(button3, INPUT);


  lcd.setCursor(0, 0);
  lcd.scrollDisplayRight();
  lcd.print("Choose mode"); //prompting the user to chose the
mode he/she wants to go into
  //delay(1000);

  Serial.begin(9600);
  pinMode(ledr,OUTPUT);

  // Setup the buttons with an internal pull-up
  pinMode(BUTTON1, INPUT_PULLUP);
  pinMode(BUTTON2, INPUT_PULLUP);
  pinMode(BUTTON3, INPUT_PULLUP);
  pinMode(BUTTON4, INPUT_PULLUP);
  pinMode(BUTTON5, INPUT_PULLUP);
  pinMode(BUTTON6, INPUT_PULLUP);
  pinMode(BUTTON7, INPUT_PULLUP);

  //setup the leds
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  pinMode(LED5, OUTPUT);
  pinMode(LED6, OUTPUT);
  pinMode(LED7, OUTPUT);

  // Setup the Bounce instances
  debouncer1.attach(BUTTON1);
  debouncer1.interval(5); // interval in ms
  debouncer2.attach(BUTTON2);
  debouncer2.interval(5); // interval in ms
  debouncer3.attach(BUTTON3);
  debouncer3.interval(5); // interval in ms
  debouncer4.attach(BUTTON4);
  debouncer4.interval(5); // interval in ms
  debouncer5.attach(BUTTON5);
  debouncer5.interval(5); // interval in ms
  debouncer6.attach(BUTTON6);
  debouncer6.interval(5); // interval in ms
  debouncer7.attach(BUTTON7);
  debouncer7.interval(5); // interval in ms

   // wait a little before start
  delay(2000);

  // generating a random number
  randomSeed(analogRead(0));
  // get first sequence of notes
  generateSequence();

  attachInterrupt(digitalPinToInterrupt(button2), button_ISR,
CHANGE);

}
/*
 * Generates a random seqeunce of notes for each level. This
ensures that the user
 * gets a different sequence of notes everytime he/she plays
the game. Thus the piano
 * is a good game and does not become predictable.
 */
void generateSequence() {
  Serial.print("Sequence:");
  for (int i = 0; i < numNotes; i ++ ) { //numNotes ensures that
for each level only an appropriate number of notes are being
genreated
    int num = random(1, 8); //a random number between 1 and
8 is generated, this ensures that each of the buttons get played
at some point
    Serial.print(num);
    Serial.print(", ");
    switch(numNotes){ //a switch statement so that the correct
sequence is recorded based on the level that the player is at
      case 3: //level 1
        noteSequence_1[i] = num;
        break;
      case 4: //level 2
        noteSequence_2[i] = num;
        break;
      case 5: //level 3
        noteSequence_3[i] = num;
        break;
      case 6: //level 4
        noteSequence_4[i] = num;
        break;
      case 7: //level 5
        noteSequence_5[i] = num;
        break;
      case 8: //level 6
        noteSequence_6[i] = num;
        break;
      case 9: //level 7
        noteSequence_7[i] = num;
        break;
    }
  }
  Serial.println();
}

void loop() {

  //reading the 3 buttons to know which mode we should go
into
  state1 = digitalRead(button1);
  state2 = digitalRead(button2);
  Serial.print("button 2 is");
  Serial.println(state2);
  state3 = digitalRead(button3);

  // If the state of the button does not equal it's previous state (1
= HIGH, 0 = LOW), which basically
  // means, was it just pressed or released? If the state of the
button is HIGH (1)
  // we increment the count of that button.
  // Note the delay(10). This is important. It facilitates a
debounce. Buttons have an intert
  // bounciness. You can't feel it, but when you press a button,
it sometimes makes
  // contact more than once, and the Arduino will see that as
multiple presses. The delay
  // pauses the program for 10 milliseconds, allowing the
button to "settle", and then
  // resumes. 10 milliseconds should be long enough, but adjust
longer if needed.

  /*MODE 1*/
  if (state1 != prev1){

    delay(10);
    if (state1 == HIGH){
      //printing to the lcd display that user is in mode 1
      lcd.clear();
      Serial.println("mode 1");
      lcd.setCursor(0, 0);
      lcd.print("Mode 1");
     // lcd.clear();
     //count1++;


     analogWrite(BUZZER,0);
     digitalWrite(ledr,HIGH);
      //Serial.print("hello");
     while(1){
      if(path==0)
        {
          Serial.println("playback");
          playback(); //calling the playback mode
        }


      //Serial.print("inside the while loop");
        if((millis() - previousPress) > buttonDebounce &&
buttonFlag)
   {
    previousPress = millis();
    if(digitalRead(button1) == LOW && previousState ==
HIGH)
    {
      path =! path;
      previousState = LOW;
    }

    else if(digitalRead(button1) == HIGH && previousState ==
LOW)
    {
      previousState = HIGH;
    }
    buttonFlag = 0;
  }
 for (int thisNote = 0; thisNote < 8; thisNote++) { //we have 7
keys on the piano, so we are looping through them all
    //Serial.println("inside for loop");
    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000/noteDurations[thisNote];

    /*Storing all the keys that are pressed while in recording
mode*/
    if(digitalRead(BUTTON1)==LOW)
    {

      //analogWrite(buzzer,frequency[0]);
      tone(10, melody[0], noteDuration);
      on_time=millis();
      if(i>0)
      {
        button_offtime[i-1]=on_time-off_time;
      }
      while(digitalRead(BUTTON1)==LOW);
      if(path==1)
      {
        off_time=millis();
        button_ontime[i]=(off_time-on_time);
        button_seq[i]=0; //button 1 has been stored
        i++;
        Serial.println("button 1 stored");
      }
    }

  else if(digitalRead(BUTTON2)==LOW)
  {
    tone(10, melody[1], noteDuration);
    //analogWrite(buzzer,frequency[1]);
    on_time=millis();
    if(i!=0)
    button_offtime[i-1]=on_time-off_time;
    while(digitalRead(BUTTON2)==LOW);
    if(path==1)
    {
      off_time=millis();
      button_ontime[i]=(off_time-on_time);
      button_seq[i]=1; //button 2 has been stored
       i++;
       Serial.println("button 2 stored");
    }
  }
```

```
else if(digitalRead(BUTTON3)==LOW)
{

  tone(10, melody[2], noteDuration);
  //analogWrite(buzzer,frequency[2]);
  on_time=millis();
  if(i!=0)
  button_offtime[i-1]=on_time-off_time;
  while(digitalRead(BUTTON3)==LOW);
  if(path==1)
    {
     off_time=millis();
     button_ontime[i]=(off_time-on_time);
     button_seq[i]=2; //button 3 has been stored
      i++;
      Serial.println("button 3 stored");
    }
}

else if(digitalRead(BUTTON4)==LOW)
{
  //analogWrite(buzzer,frequency[3]);
  tone(10, melody[3], noteDuration);
  on_time=millis();
  if(i!=0)
  button_offtime[i-1]=on_time-off_time;
  while(digitalRead(BUTTON4)==LOW);
  if(path==1)
    {
     off_time=millis();
     button_ontime[i]=(off_time-on_time);
     button_seq[i]=3; //button 4 has been stored
      i++;
      Serial.println("button 4 stored");
    }
}

else if(digitalRead(BUTTON5)==LOW)
{
  //analogWrite(buzzer,frequency[4]);
  tone(10, melody[4], noteDuration);
  on_time=millis();
  if(i!=0)
  button_offtime[i-1]=on_time-off_time;
  while(digitalRead(BUTTON5)==LOW);
  if(path==1)
    {
     off_time=millis();

     button_ontime[i]=(off_time-on_time);
     button_seq[i]=4; //button 5 has veen stored
      i++;
      Serial.println("button 5 stored");
    }
}

else if(digitalRead(BUTTON6)==LOW)
{
//analogWrite(buzzer,frequency[5]);
tone(10, melody[5], noteDuration);
```

```
  on_time=millis();
  if(i!=0)
  button_offtime[i-1]=on_time-off_time;
  while(digitalRead(BUTTON6)==LOW);
  if(path==1)
    {
     off_time=millis();
     button_ontime[i]=(off_time-on_time);
     button_seq[i]=5; //button 6 has been stored
      i++;
      Serial.println("button 6 stored");
    }
  }

  else if(digitalRead(BUTTON7)==LOW)
  {
    //analogWrite(buzzer,frequency[6]);
    tone(10, melody[6], noteDuration);
    on_time=millis();
    if(i!=0)
    button_offtime[i-1]=on_time-off_time;
    while(digitalRead(BUTTON7)==LOW);
    if(path==1)
      {
       off_time=millis();
       button_ontime[i]=(off_time-on_time);
       button_seq[i]=6; //button 7 has been stored
        i++;
        Serial.println("button 7 stored");
      }
    }
 }
//analogWrite(buzzer,0);
noTone(10); //stopping the buzzer from making any more
noise
}

     }
    }



    else{}


//
//  /*MODE 2*/
//  if (state2 != prev2){
//    delay(10);
//    if (state2 == HIGH){
//      //printing to the lcd display that user is in mode 2
//      lcd.clear();
//      Serial.println("you are now in mode 2");
//      lcd.setCursor(0, 0);
//      lcd.print("Mode 2");
//      //lcd.clear();
//      //count2++;
//      Serial.println("playback");
//      playback(); //calling the playback mode
//    }
//    else{}
// }

  /*MODE 3*/
  if (state3 != prev3){
    delay(10);

    //printing to the lcd display that user is in mode 1

    if (state3 == HIGH){
      lcd.clear();
      Serial.println("you are now in mode 3");
      lcd.setCursor(0, 0);
      lcd.print("Mode 3");
      //lcd.clear();

    //printing to lcd display that the game is starting and
informing the user of the number of lives he/she has
    //and also about the level he/she is on
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Starting Game");
    delay(500);
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("Level:");
    lcd.print(numNotes-2);
    lcd.print(" Lives:");
    lcd.print(lives);
    while(1){

    /*Arduino plays the tune that user must repeat*/
    if (!isPlaying) {
      isPlaying = true;
      currentNote = 0;
    switch(numNotes){
      case 3: //level 1
        playSequence(noteSequence_1);
        break;
      case 4: //level 2
        playSequence(noteSequence_2);
        break;
      case 5: //level 3
        playSequence(noteSequence_3);
        break;
      case 6: //level 4
        playSequence(noteSequence_4);
        break;
      case 7: //level 5
        playSequence(noteSequence_5);
        break;
      case 8: //level 6
        playSequence(noteSequence_6);
        break;
      case 9: //level 7
        playSequence(noteSequence_7);
        break;
    }

  }
  // Update the Bounce instances
  debouncer1.update();
  debouncer2.update();
  debouncer3.update();
  debouncer4.update();
  debouncer5.update();
  debouncer6.update();
  debouncer7.update();


  // Check the pressed buttons
  if (debouncer1.fell()){
    playButtonNote(1); //regardless of whether the button is the
correct one in sequence or not, the tone is emitted by the
buzzer
    verifyNote(1);
  }

  if (debouncer2.fell()){
    playButtonNote(2);
    verifyNote(2);
  }

  if (debouncer3.fell()){
    playButtonNote(3);
    verifyNote(3);
  }
  if (debouncer4.fell()){
    playButtonNote(4);
    verifyNote(4);
  }

  if (debouncer5.fell()){
    playButtonNote(5);
    verifyNote(5);
  }

  if (debouncer6.fell()){
    playButtonNote(6);
    verifyNote(6);
  }

  if (debouncer7.fell()){
    playButtonNote(7);
    verifyNote(7);
  }
  }
  }

// This will help keep track of what kind of actions are going
on with
// the buttons. When the button is pushed, its state does not
match
// it's previous state, indicating a change. Here we set the
previous
// state to the current state, so that the moment you push or
// release, the Arduino knows it, and acts accordingly.

  prev1 = state1;
  prev2 = state2;
  prev3 = state3;



}
}

/*
 * This method plays the array of notes that it is given. It is
called when the piano
 * wants to let the user listen to the tune he/she must replicate
 * @param notes the array of notes that contains the correct
sequence
 */
void playSequence(int notes[]) {
  for (int i=0; i< numNotes; i++) {
    playNote(notes[i]);
  }
}

/*
 * This method is used to verify is the button that user presses
if the correct button in the
 * sequence.
 * @param note the note that the user presses
 */
void verifyNote(int note) {
  //getting the actual note that should be played in the
sequence, so that later we can compare
  //with what the user played and decide if user is right or
wrong
  int noteInSequence;
  switch(numNotes){ //figuring out which level the user is in
    case 3: //level 1
      noteInSequence = noteSequence_1[currentNote];
      break;
    case 4: //level 2
      noteInSequence = noteSequence_2[currentNote];
      break;
    case 5: //level 3
      noteInSequence = noteSequence_3[currentNote];
      break;
    case 6: //level 4
      noteInSequence = noteSequence_4[currentNote];
      break;
    case 7: //level 5
      noteInSequence = noteSequence_5[currentNote];
      break;
    case 8: //level 6
      noteInSequence = noteSequence_6[currentNote];
      break;
    case 9: //level 7
      noteInSequence = noteSequence_7[currentNote];
      break;
  }
  if (noteInSequence == note) { //Note is correct
    currentNote++; // Go to the next note
    if (currentNote == numNotes && numNotes == 9) { //
Finish all levels
```

```
      //printing to lcd display
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("You won, congratulations!");
      delay(500);

      //Serial.println("You won, congratulations!");
      final_win(); //a special tune is played when user
successfully completes all levels
    } else if (currentNote == numNotes){ // Finish current level
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Correct tune, move on to next level");
      delay(500);

      //Serial.println("Very well, you got it right!");
      win();

      //displaying the level that the user is at, and the number of
lives he/she has left
      lcd.clear();
      lcd.setCursor(0, 1);
      lcd.print("Level:");
      lcd.print(numNotes-2);
      lcd.print(" Lives:");
      lcd.print(lives);
    }
  } else if (lives == 1){ //Note is wrong, the user has only 3
lives, after getting sequences wrong 3 times the game is lost
      lcd.clear();
      lcd.setCursor(0, 1);
      lcd.print("You lost the game!");
      final_lose();
  } else { //Note is wrong, however the user still has more
lives, so he/she can continue playing that game
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("You lost one life");

      //displaying the level the user is at and the no. of lives
he/she has left
      lose();
      delay(500);
      lcd.clear();
      lcd.setCursor(0, 1);
      lcd.print("Level:");
      lcd.print(numNotes-2);
      lcd.print(" Lives:");
      lcd.print(lives);
  }
}

/*
 * This method allows the piano to play notes to the user, it
sets the note duration and the interval between
 * the notes according to the level of difficulty. Shorter
durations and shorter interval between notes as the
 * difficulty level increases
 * @param note this is the note that has to played
 */
void playNote(int note) {
  int noteDuration; //how long the note is played for
  int intervalDuration; //the time in between in each note
  /*
   * Setting the difficulty of the levels by decreasing the
interval
   */
  switch (numNotes) {
    case 3:
      noteDuration = 1500; //notice that the note duration
decreases as the levels increase
      intervalDuration = 700; //notice that the note interval
decreases as the levels increase
      break;
    case 4:
      noteDuration = 1300;
      intervalDuration = 625;
      break;
    case 5:
      noteDuration = 1100;
      intervalDuration = 550;
      break;
    case 6:
      noteDuration = 900;
      intervalDuration = 475;
      break;
    case 7:
      noteDuration = 700;
      intervalDuration = 400;
      break;
    case 8:
      noteDuration = 550;
      intervalDuration = 325;
      break;
    case 9:
      noteDuration = 400;
      intervalDuration = 250;
      break;
  }
  /*This is where the note is actually played by the buzzer,
cases for each of the push buttons*/
  switch (note) {
    case 1:
      tone(BUZZER, NOTE_C4, 100);
      digitalWrite(LED1, HIGH); //we turn the LED on as well
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED1, LOW);
      delay (intervalDuration);
      break;
    case 2:
      tone(BUZZER, NOTE_D4, 100 );
      digitalWrite(LED2, HIGH);
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED2, LOW);
      delay (intervalDuration);
      break;
    case 3:
      tone(BUZZER, NOTE_E4, 100);
      digitalWrite(LED3, HIGH);
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED3, LOW);
      delay (intervalDuration);
      break;
    case 4:
      tone(BUZZER, NOTE_F4, 100);
      digitalWrite(LED4, HIGH);
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED4, LOW);
      delay (intervalDuration);
      break;
    case 5:
      tone(BUZZER, NOTE_G4, 100);
      digitalWrite(LED5, HIGH);
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED5, LOW);
      delay (intervalDuration);
      break;
    case 6:
      tone(BUZZER, NOTE_B4, 100);
      digitalWrite(LED6, HIGH);
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED6, LOW);
      delay (intervalDuration);
      break;
    case 7:
      tone(BUZZER, NOTE_C5, 100);
      digitalWrite(LED7, HIGH);
      delay(noteDuration);
      noTone(BUZZER);
      digitalWrite(LED7, LOW);
      delay (intervalDuration);
      break;


  }
}
/*
 * Makes the buzzer play the tone, and the LED behind light up.
 * This method is called when the user presses a button.
 * @param note the buttont the user presses
 */
void playButtonNote(int note) {
  //one case for each of the 7 buttons
  switch (note) {
    case 1:
      tone(BUZZER, NOTE_C4, 100);
      digitalWrite(LED1, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED1, LOW);
      break;
    case 2:
      tone(BUZZER, NOTE_D4, 100);
      digitalWrite(LED2, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED2, LOW);
      break;
    case 3:
      tone(BUZZER, NOTE_E4, 100);
      digitalWrite(LED3, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED3, LOW);
      break;
    case 4:
      tone(BUZZER, NOTE_F4, 100);
      digitalWrite(LED4, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED4, LOW);
      break;
    case 5:
      tone(BUZZER, NOTE_G4, 100);
      digitalWrite(LED5, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED5, LOW);
      break;
    case 6:
      tone(BUZZER, NOTE_B4, 100);
      digitalWrite(LED6, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED6, LOW);
      break;
    case 7:
      tone(BUZZER, NOTE_C5, 100);
      digitalWrite(LED7, HIGH);
      delay(500);
      noTone(BUZZER);
      digitalWrite(LED7, LOW);
      break;    }
}

/*
 *The tune that plays when the user wins a level and progresses onto the next
 */
void win() {
  currentNote = 0;
  isPlaying = false;
  numNotes++;
  generateSequence();
  delay(500);

  // Classic melody used in the Arduino tone example
  // notes in the melody:
  int melody[] = {
    NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3,
0, NOTE_B3, NOTE_C4, NOTE_C4, NOTE_G3, NOTE_G3,
NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
  };
  // note durations: 4 = quarter note, 8 = eighth note, etc.:
  int noteDurations[] = {
    4, 8, 8, 4, 4, 4, 4, 4, 4, 4, 8, 8, 4, 4, 4, 4, 4
  };

  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 16; thisNote++) {

    // to calculate the note duration, take one second divided by
the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(BUZZER, melody[thisNote], noteDuration);
    switch(thisNote){
      case 0:
        digitalWrite(LED1, HIGH);
        break;
      case 1:
        digitalWrite(LED2, HIGH);
        break;
      case 2:
        digitalWrite(LED3, HIGH);
        break;
      case 3:
        digitalWrite(LED4, HIGH);
        break;
      case 4:
        digitalWrite(LED1, LOW);
        break;
      case 5:
        digitalWrite(LED2, LOW);
        break;
      case 6:
        digitalWrite(LED3, LOW);
        break;
      case 7:
        digitalWrite(LED4, LOW);
        break;

      case 8:
        digitalWrite(LED4, HIGH);
        break;
      case 9:
        digitalWrite(LED5, HIGH);
        break;
      case 10:
        digitalWrite(LED6, HIGH);
        break;
      case 11:
        digitalWrite(LED7, HIGH);
        break;
      case 12:
        digitalWrite(LED4, LOW);
        break;
      case 13:
        digitalWrite(LED5, LOW);
        break;
      case 14:
        digitalWrite(LED6, LOW);
        break;
      case 15:
        digitalWrite(LED7, LOW);
        break;

    }

    // to distinguish the notes, set a minimum time between
them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(BUZZER);
  }
  delay(1500);
}

/*
 * The tune that is played when the user loses a level by
pressing a wrong button. This is not what is played if
 * the player loses all his/her lives and loses the entire game.
 */
void lose() {
  currentNote = 0;
  lives--;
  isPlaying = false;
  generateSequence();
  delay(500);

  // notes in the melody:
  int melody[] = {
    NOTE_C3, NOTE_C2, NOTE_C1
  };

  // note durations:
  int noteDurations[] = {
    400, 500, 800
  };

  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 3; thisNote++) {

    // note duration directly
    tone(BUZZER, melody[thisNote], 100);
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, HIGH);
    digitalWrite(LED4, HIGH);
    digitalWrite(LED5, HIGH);
    digitalWrite(LED6, HIGH);
    digitalWrite(LED7, HIGH);
    delay(noteDurations[thisNote]);

    // stop the tone playing:
    noTone(BUZZER);
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED4, LOW);
    digitalWrite(LED5, LOW);
    digitalWrite(LED6, LOW);
    digitalWrite(LED7, LOW);

    // to distinguish the notes, set a minimum time between
them.
    delay(200);
  }
  delay(1500);
}

/*
 * Plays a tune when user has completed all the levels
successfully
 */
void final_win(){
  currentNote = 0;
  isPlaying = false;
  numNotes = 3;
  lives = 3;
  generateSequence();
  delay(500);

  // Melody of "We are the champions" by Queen
  // notes in the melody:

  int melody[] = {
    NOTE_F3, NOTE_DS3, NOTE_F3, NOTE_DS3,
NOTE_C3, NOTE_GS2, NOTE_D3, 0,
    NOTE_F3, NOTE_G3, NOTE_A4, NOTE_C4, NOTE_A4,
NOTE_D3, NOTE_E3, NOTE_D3, 0,
    NOTE_D3, NOTE_E3, NOTE_D3, NOTE_E3,
NOTE_AS3,
    NOTE_AS4, NOTE_A4, NOTE_AS4, NOTE_A4,
NOTE_G3,
    NOTE_A4, NOTE_F3, NOTE_AS4, NOTE_A4,
NOTE_F3, NOTE_AS4,
    NOTE_GS3, NOTE_F3, NOTE_AS4, NOTE_GS3,
NOTE_F3, 0,
    NOTE_DS3, NOTE_C3, NOTE_F3
  };

  // note durations:
  int noteDurations[] = {
    1000, 250, 250, 500, 750, 250, 1000, 1500,
    1000, 250, 250, 500, 500, 250, 250, 1500, 1500,
    750, 500, 250, 750, 750,
    750, 500, 250, 750, 750,
    750, 500, 250, 750, 500, 250,
    750, 500, 250, 750, 750, 750,
    250, 250, 3000
  };

  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 42; thisNote++) {
    if(melody[thisNote] == 0){
      noTone(BUZZER);
    } else {
      tone(BUZZER, melody[thisNote], 100);
    }
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, HIGH);
    digitalWrite(LED4, HIGH);
    digitalWrite(LED5, HIGH);
    delay(noteDurations[thisNote]);

    // stop the tone playing:
    noTone(BUZZER);
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED4, LOW);
    digitalWrite(LED5, LOW);
    // to distinguish the notes, set a minimum time between
them.
    delay(100);
  }
  delay(1500);
}

/*
 * Plays a tune when the player uses the entire game
 */
void final_lose(){
  currentNote = 0;
  numNotes = 3;
  lives = 3;
  isPlaying = false;
  generateSequence();
  delay(500);

  // Melody of "Sonata n2, op 35 (Marche Funebre)" by
Chopin
  // notes in the melody:
  int melody[] = {
    NOTE_D2, NOTE_D2, NOTE_D2, NOTE_D2, NOTE_F2,
NOTE_E2, NOTE_E2, NOTE_D2, NOTE_D2, NOTE_D2,
NOTE_D2
  };

  // note durations:
  int noteDurations[] = {
    1000, 750, 250, 1000, 750, 250, 750, 250, 750, 250, 1000
  };

  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 11; thisNote++) {
    tone(BUZZER, melody[thisNote], 100);
    digitalWrite(LED1, HIGH);
    digitalWrite(LED2, HIGH);
    digitalWrite(LED3, HIGH);
    digitalWrite(LED4, HIGH);
    digitalWrite(LED5, HIGH);
    digitalWrite(LED6, HIGH);
    digitalWrite(LED7, HIGH);
    delay(noteDurations[thisNote]);

    // stop the tone playing:
    noTone(BUZZER);
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED4, LOW);
    digitalWrite(LED5, LOW);
    digitalWrite(LED6, LOW);
    digitalWrite(LED7, LOW);
    // to distinguish the notes, set a minimum time between
them.
    delay(100);
  }
  delay(1500);
  setup();
}


/*
Tone generator
v1  use timer, and toggle any digital pin in ISR
   funky duration from arduino version
   TODO use FindMckDivisor?
   timer selected will preclude using associated pins for PWM
etc.
    could also do timer/pwm hardware toggle where caller
controls duration
*/


// timers TC0 TC1 TC2   channels 0-2 ids 0-2  3-5  6-8     AB
0 1
// use TC1 channel 0
#define TONE_TIMER TC1
#define TONE_CHNL 0
#define TONE_IRQ TC3_IRQn

// TIMER_CLOCK4   84MHz/128 with 16 bit counter give 10
Hz to 656KHz
//  piano 27Hz to 4KHz

static uint8_t pinEnabled[PINS_COUNT];
static uint8_t TCChanEnabled = 0;
static boolean pin_state = false ;
static Tc *chTC = TONE_TIMER;
static uint32_t chNo = TONE_CHNL;

volatile static int32_t toggle_count;
static uint32_t tone_pin;

// frequency (in hertz) and duration (in milliseconds).

void tone(uint32_t ulPin, uint32_t frequency, int32_t duration)
{
    const uint32_t rc = VARIANT_MCK / 256 / frequency;
    tone_pin = ulPin;
    toggle_count = 0;  // strange  wipe out previous duration
    if (duration > 0 ) toggle_count = 2 * frequency * duration /
1000;
     else toggle_count = -1;

    if (!TCChanEnabled) {
      pmc_set_writeprotect(false);
      pmc_enable_periph_clk((uint32_t)TONE_IRQ);
      TC_Configure(chTC, chNo,
        TC_CMR_TCCLKS_TIMER_CLOCK4 |
        TC_CMR_WAVE |         // Waveform mode
        TC_CMR_WAVSEL_UP_RC ); // Counter running up
and reset when equals to RC

      chTC->TC_CHANNEL[chNo].TC_IER=TC_IER_CPCS;
// RC compare interrupt
      chTC-
>TC_CHANNEL[chNo].TC_IDR=~TC_IER_CPCS;
       NVIC_EnableIRQ(TONE_IRQ);
                         TCChanEnabled = 1;
    }
    if (!pinEnabled[ulPin]) {
      pinMode(ulPin, OUTPUT);
      pinEnabled[ulPin] = 1;
    }
    TC_Stop(chTC, chNo);
                TC_SetRC(chTC, chNo, rc);    // set frequency
    TC_Start(chTC, chNo);
}

void noTone(uint32_t ulPin)
{
  TC_Stop(chTC, chNo);  // stop timer
  digitalWrite(ulPin,LOW);  // no signal on pin
}

// timer ISR  TC1 ch 0
void TC3_Handler ( void ) {
  TC_GetStatus(TC1, 0);
  if (toggle_count != 0){
    // toggle pin  TODO  better
    digitalWrite(tone_pin,pin_state= !pin_state);
    if (toggle_count > 0) toggle_count--;
  } else {
    noTone(tone_pin);
  }
}

/**
 * This method allows us to playback the tune that the user
recorded
 * while in recording mode. We are reading through the array
called button_seq[]
 * that contains all the buttons that were pressed in the
recording mode in the
 * correct order
 */
void playback (void)
{

  lcd.clear();
//      Serial.println("you are now in mode 2");
      lcd.setCursor(0, 0);
      lcd.print("Mode 2");
//      //lcd.clear();
```

```
 digitalWrite(ledr,LOW); //REMOVE
 for(int j=0; j<i; j++) //iterating through the array that stores
the notes in order
 {
  Serial.print("button in sequence:
");Serial.print(melody[button_seq[j]]);
  //analogWrite(buzzer,melody[button_seq[j]]);
  tone(10, melody[button_seq[j]], 100); //playing the note
  delay(400);
  //noTone(buzzer);
  //delay(400);
  if(j == i-1) break;
 }
 noTone(BUZZER); //ensuring that the buzzer stops making
noise

 i=0;
 off_time=0;
 on_time=0;
 path=1;
 //digitalWrite(ledr,HIGH);
 setup();
}
```