Trabajo Practico De Evaluacion Final

Seminario de Microcontroladores

(primer cuatrimestre del 2020)

Por:

Axel Lopez Garabal

Índice

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

Objetivo

El Objetivo de este proyecto es crear un circuito sonoro capaz de reproducir diferentes melodías preestablecidas y dándole al usuario la posibilidad de crear sus melodías y almacenarlos dentro del circuito para poder reproducirlas. Alternar la forma de la reproducción para que se permita una reproducción secuencial o una reproducción random de las melodías. Darle al usuario control sobre la reproducción de la melodía permitiéndole al usuario retroceder a la melodía anterior, avanzar a la siguiente detener la reproducción que está en curso(ya sea que se reproduzca un listado de melodías o una melodía específica).

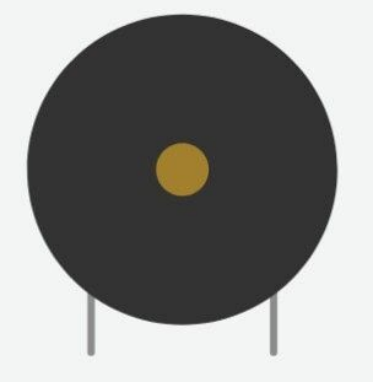
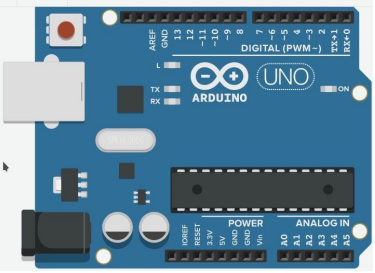
Funcionalidades

Listado de funcionalidades

- Reproducir una melodía específica.
- Reproducir todas las melodías en secuencia.
- Reproducir melodías de forma random.
- Reproducir melodías que ingresa el usuario mediante el uso del puerto serial.
- Obtener la data de la melodía(nota y duración de las mismas).
- Detener la reproducción de la/s melodia/s.
- Avanzar a la siguiente melodía.
- Retroceder a la siguiente melodía.
- Agregar(almacenar en memoria) melodías mediante el puerto serial

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

Elementos necesarios

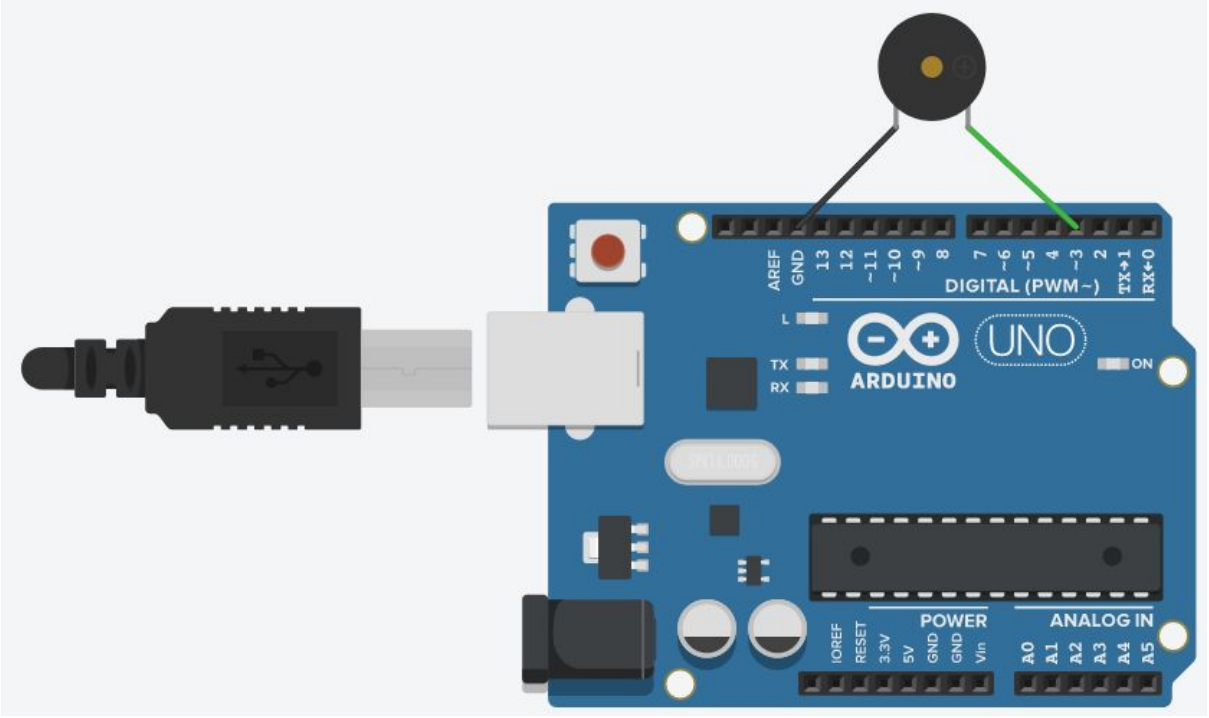
elemento	***detalle***	***imagen***
Piezo	*Un tipo de zumbador que emite sonido en distintas frecuencias*	
Arduino Uno	*Placa programable que puede ser utilizada para crear circuitos interactivos*	
Cables para conexión		

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

Diagrama de conexión

1) Conectar el pin negativo del piezo al pin digital GND del arduino.
2) conectar el pin positivo del piezo al pin digital 3 del arduino.

Quedando como resultado el siguiente circuito:



archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

Código relacionado a la reproducción de melodía

- *Módulo/Biblioteca pitches*

  Dado a que la aplicación tinkercad no tiene la opción de incorporar la biblioteca pitches.h se debe agregar al código del arduino.
  La biblioteca pitches.h es una biblioteca que contiene todos los valores para los tonos de las notas que se utilizan. Esta tabla fue escrita originalmente por Brett Hagman. Esta se puede encontrar en el repositorio, https://gist.github.com/mikeputnam/2820675 .

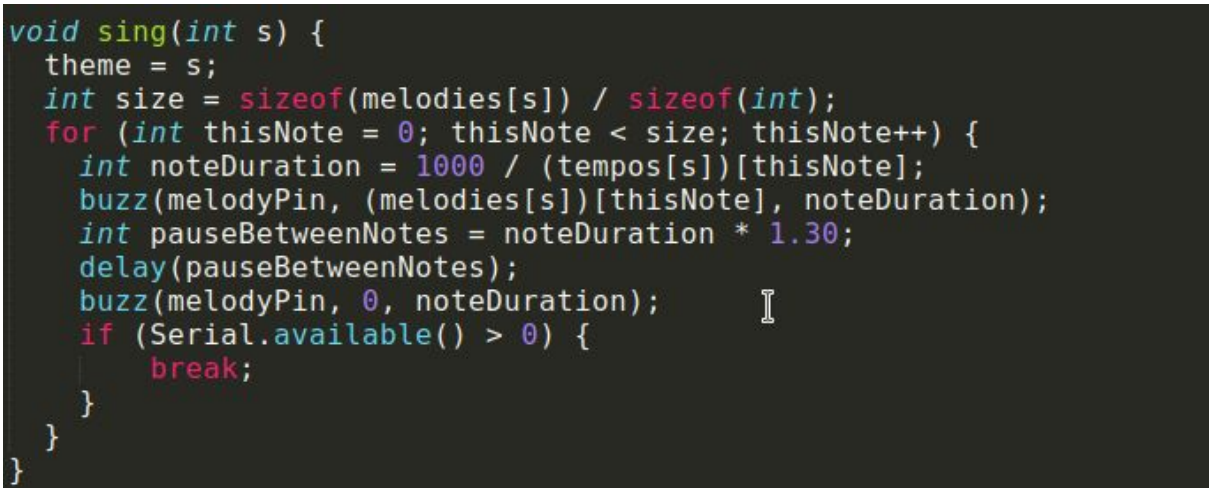
- *Estructura de representación de una melodía*

  La estructura de los datos seleccionada para la representación de la melodía, que está compuesta por un nombre(String), un conjunto de notas y un conjunto de tempos, la más recomendable era utilizar un diccionario donde la clave es el nombre de la canción y el valor es una tupla donde el primer elemento sería el arreglo de las notas y el otro elemento el conjunto de los tempos. Pero, por simplicidad y que luego en el refactor de código se modifique, se optó por hacer tres arreglos uno que contenga los nombres, y otros dos, uno de las notas que es un arreglo de arreglos de notas, mientras que el otro con la misma estructura es de tempos. Para recorrerlo se utiliza los índices de los arreglos, ya que el invariante de representación es que las notas y los tempos de una melodía están ubicados en el mismo índice que el nombre de la canción(por ende para la canción cuyo índice es cero las notas y tempos correspondientes están en el mismo índice, el índice cero).

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

- *Codigo de reproducción de una melodía*

  Para la reproducción de una melodía se definió la función sing que recibe un int como parámetro, que es el índice que corresponde a la melodía que se desea reproducir. Se comienza calculando una variable size que calcula el tamaño del arreglo de de las notas de la melodía seleccionada(este valor se obtiene de hacer *sizeof(array)/sizeof(int)*). Luego se procede a calcular la duración de las notas que se calcula dividiendo un segundo por el tipo de nota y el resultado se lo asigna a la variable *noteDuration* este dato junto con el melodyPin(el pin al que está conectado el piezo) y la nota que se quiere reproducir son los parámetros utilizados para la función *buzz*. Luego se define la variable pauseBetweenNotes que tiene como fin generar una pausa entre las notas para que sean distinguibles y se vuelve a llamar a la función buzz para que se detenga el tono que se reproduce.



```
void sing(int s) {
  theme = s;
  int size = sizeof(melodies[s]) / sizeof(int);
  for (int thisNote = 0; thisNote < size; thisNote++) {
    int noteDuration = 1000 / (tempos[s])[thisNote];
    buzz(melodyPin, (melodies[s])[thisNote], noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    buzz(melodyPin, 0, noteDuration);
    if (Serial.available() > 0) {
        break;
    }
  }
}
```

- *Método buzz*

  El método buzz recibe tres parámetros el pin conectado al piezo, un long de la frecuencia del tono y un long que es la longitud del tono. Comienza calculando el valor del delay que hay entre frecuencias se

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

obtiene dividiendo un segundo por la frecuencia y a ese resultado se lo divide a la mitad dado a que por cada ciclo hay dos fases(una alta y otra baja).
Luego se calcula el número de ciclos para el adecuar el tiempo para una correcta reproducción(se obtiene multiplicando la frecuencia por la longitud del tono dividido por un segundo, para obtener la cantidad de ciclos que se deben reproducir).
Con un loop *for* el cual loopea tantas n veces, donde n es la cantidad calculada de tiempo, en cada loop se escribe en el pin pasado por parámetro *HIGH*, se espera en función del valor calculado con anterioridad, se escribe en el pin pasado por parámetro *LOW* para generar el efecto opuesto al anterior y así generar un sonido.

```
void buzz(int targetPin, long frequency, long length) {
  long delayValue = 1000000 / frequency / 2;
  long numCycles = frequency * length / 1000;
  for (long i = 0; i < numCycles; i++) {
    digitalWrite(targetPin, HIGH);
    delayMicroseconds(delayValue);
    digitalWrite(targetPin, LOW);
    delayMicroseconds(delayValue);
  }
}
```

Código relacionado al puerto serial

- *Código de Lectura y respuesta el puerto serial*

  Para la lectura del puerto serial se utilizan dos métodos uno encargado de verificar el input, en caso de que el input del usuario sea mayor de cero, se crea una variable input en el que se coloca lo que se leyó del puerto serial, luego se llama al método commands pasandole esta variable por parámetro, y se modifica la variable global newData para que contenga el valor true(la variable newData es un bool el cual sirve para determinar si se genero un input en el puerto serial por el usuario).

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

```
void recvOneChar() {
 if (Serial.available() > 0) {
  String input = Serial.readString();
  commands(input);
  newData = true;
 }
}
```

Para la escritura sobre el puerto serial (obteniendo así un output para el usuario) se hace uso de un condicional para el determinar si se ejecutara el código, la condición para que esto ocurra es la variable global newData, del tipo bool, luego se printea el contenido de la variable receivedChar (del tipo String, esta es modificada dentro del método commands que recibe el input del usuario) y cambia el valor de la variable newData a false(esto evita que se siga ejecutando el print del receivedChar).

```
void showNewData() {
 if (newData == true) {
 Serial.println(receivedChar);
 newData = false;
 }
}
```

- *Código del comando list*

Para informar al usuario de los temas que posee el circuito se creó el comando *list* el cual no recibe parámetros y retorna un string con los títulos de los temas. Este string se genera a partir de una variable local(res) y se itera sobre el arreglo de temas(themes) donde el valor i inicia en cero y mientras sea menor a ARRAYSIZE(cantidad de elementos en el arreglo de temas) estos elementos se agregan al string res junto con un string el cual contiene un coma y se le asigna a res. Al finalizar el for se retorna res(un string compuesto con los valores del arreglo themes)

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

```
String list(){
  String res="";
  for (int i=0; i< ARRAYSIZE; i++) {
    res = res + themes[i] + ", ";
  }
  return res;
}
```

- *Código del comando play*

  Para el comando play el cual recibe un String como parámetro que es el nombre de la canción que se quiere escuchar, se inicia con un condicional preguntando si la canción no existe en caso de que no exista se retorna un mensaje de error. En caso de que exista la canción se actualiza theme(variable global que se utiliza principalmente en los comandos *next* y *back*) y se llama al método sing(pasando theme como parámetro).

```
bool exist(String theme_name){
  for (int i =0; i< ARRAYSIZE; i++) {
    if(theme_name == themes[i]) {
      return true;
    }
  }
  return false;
}

void play(String theme_name){
  if(!exist(theme_name)){
    receivedChar = "ERROR..." + theme_name + "...?";
  }
  else{
    for (int i =0; i< ARRAYSIZE; i++) {
        if(theme_name == themes[i]) {
            theme = i;
            sing(theme);
        }
    }
  }
}
```

- *Código del comando next*

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

Para que el usuario sea capaz de pasar de un tema al siguiente este tiene la opción de usar el comando *next* el cual no recibe parámetros, este le asigna a la variable global *theme* el valor que corresponde al sumarle uno a theme y de este valor se calcula el módulo de ARRAYSIZE, de esta forma no se sale de los índices del arreglo luego se llama a playByIndex que recibe como parámetro la variable theme(que fue modificado anteriormente, este mensaje se encarga de reproducir el tema que corresponde al valor de theme).

```
void next(){
  theme = theme +1 % ARRAYSIZE;
  playByIndex(theme);
}
```

- *Código del comando back*

  Como en el caso del comando *next* se creó el comando *back* el cual tiene una pequeña diferencia con respecto al comando next(es la forma en la que se modifica la variable theme, al cual en lugar de sumarle uno en este caso se le resta uno y se realiza el módulo de este valor con ARRAYSIZE).

```
void back(){
  theme = (-1*(theme -1)) % ARRAYSIZE;
  playByIndex(theme);
}
```

- *Código del comando lyrics*

  El comando lyrics es una función que retorna un string además de que puede retornar las notas y los tempos que corresponden a una melodía dada. La misma requiere un como parámetro un String que

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

puede ser el nombre de una canción para esto lo primero que se realiza es la verificación de que el parámetro exista en se utiliza un for con el que se recorre el array themes(array de strings) en caso de no encontrarlo devuelve un mensaje de error. Si existe se pasa a ejecutar *printLyrics* que recibe el index correspondiente a la canción requerida el cual recorre ambos arreglos a la vez(el arreglo de los tonos y el de los tempos) y por cada uno de sus elementos se genera un print en pantalla del resultado de la función pair(que retorna una tupla donde el primer elemento es el tono y el segundo elemento es el tempo).

```
String lyrics(String theme_name){
  for (int i =0; i< ARRAYSIZE; i++) {
    if(theme_name == themes[i]) {
      return printLyrics(i);
    }
  }
  return "ERROR:: theme not found? or misspell";
}

String printLyrics(int index){
  String res = "";
  receivedChar = "lyrics above... \n";
  int size = sizeof(melodies[index]) / sizeof(int);
  for (int i = 0; i < size; i++) {
    Serial.println( pair( (melodies[index])[i], (tempos[index])[i]) );
  }
}


String pair(int x, int y){
    return "("+String(x)+" ,"+String(y)+")";
}
```

- *Código del comando play all*

  Para este comando ya que se debía reproducir los temas que están definidos definidos en el circuito se utiliza la cantidad de temas definidos para determinar el corte del loop y en cada iteración se ejecuta el método sing para reproducir la melodía y se reproduce en función del índice de este mismo.

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

```
void playAll(){
  for(int i=0; i < ARRAYSIZE; i++){
    sing(i);
  }
}
```

- *Código del comando play random*

  El comando play random reproduce todas las melodías en función de un valor random que se le asigna a la variable *theme*(este valor puede estar en el rango que va de cero hasta el valor de arraysize, que es 2, el primer valor está incluido mientras que el segundo no está incluido entre los valores que puede ser asignado). luego dentro de la iteración se ejecuta el método *next* de esta forma no puede salir de los índices válidos del arreglo.

```
void playRandom(){
  theme = random(0, ARRAYSIZE);
  for(int i=0; i < ARRAYSIZE; i++){
    next();
  }
}
```

- *Código del comando add*

  El comando add integra tres sub comandos que son agregar una nota(-n), eliminar una nota(-r) y reproducir el tema agregado(-p).
  En el caso de agregar una nota se lee que el input contenga el substring “-n” ubicado luego del add, si el input cumpe esta condición entonces este string restante se corta hasta encontrar la coma y se lo transforma en un int y dichos valores se le asigna al arreglo determinado por la variable *added*, el orden en el que se ingresan los datos es el siguiente:

    - el primer valor corresponde al tono de la nota que se almacena en el tercer arreglo de melody(que inicia vacío)

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

y se le asigna en la ubicación determinada por el valor de la variable *addI*.

- el segundo valor corresponde al tempo de la nota que se agregó, así como para el caso anterior su ubicación es determinado por el valor de *addI*

Luego se aumenta el valor de la variable *addI* para pasar al momento de agregar la nueva nota esta no se sobreescriba.

En el caso de querer remover una nota, seguido de escribir add debe aparecer “-r”, se optó por una solución simple que es si el valor de *addI* es mayor a cero el valor del mismo se reduce en uno y se le notifica al usuario del cambio(al hacer esto, reducir el valor de la variable, en el momento que se quiera agregar una nueva nota esta sobreescribirá a la vieja). En caso de que *addI* sea igual a cero se le notifica al usuario de que el remover una nota no es redundante ya que no hay nota que remover.

En el último caso para reproducir el tema agregado. Se verifica que a continuación apareca el valor “-p” de ser así se llama al método sing y se le para por parámetro el valor added(que corresponde al índice de la melodía que se agregó).

Si lo introducido no cae dentro de los anteriores se llama a *unkown*(que informa al usuario de que el input no es válido)

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

```
void addMethods(String input){
  if(input.substring(0,2) == "-n"){
    int x = input.substring(3, input.indexOf(',')).toInt();
    int y = input.substring(input.indexOf(',')+1).toInt();

    (melodies[added])[addI] = x;
    (tempos  [added])[addI] = y;
    addI = addI + 1;
    Serial.println("added");
  }
  else{
    if(input.substring(0,2) == "-r"){
        if(addI > 0){
          addI = addI -1;
          receivedChar="removed note";
        }
      else{receivedChar="no note to remove";}
    }
    else{
      if(input.substring(0,2) == "-p"){
            Serial.println("sufre");
            sing(added);
      }
        else{
          unkwon(input);
        }
    }
  }
}
```

- *integración de comandos*

  Este comando es un árbol de condicionales y que en función del input se determina la rama del árbol al que se ingresa la rama sigue el siguiente orden iniciando con play, add, list, next, back, lyrics y unknow. De entre estas ramas solamente play y add no son hojas, son hojas aquellas las que no tienen un conjunto de sub comandos por ejemplo play tiene como hijos

  - -a, que es la reproducción de todos los temas de forma secuencial) .
  - -r, que es la reproducción de todos los temas iniciando por un tema elegido de forma random.

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

- el último no espera ningún parámetro simplemente ejecuta play(en este mismo, se verifica la existencia del tema que se quiere reproducir).

  Se decidió que para mejorar el entendimiento del árbol y sus tareas se dividieran los else en otros métodos, de esta forma no se anida de forma compulsiva los if(dificultando su legibilidad y haciendo que sea más difícil tanto la corrección de código como el agregar código)

```
void commands(String input){
  if(input.substring(0, 4) == "play"){
    playOrAll(input.substring(5));
  }
    else{
      if(input.substring(0, 3) == "add"){
        addMethods(input.substring(4));
      }
      else{
        listAndOthers(input);
      }
    }
}
```

```
void listAndOthers(String input){
  if(input.substring(0, 4) == "list" & input.length() == 4){
    receivedChar = "list of the songs... \n" + list();
    }
  else{nextBackAndOthers(input);}
}
```

```
void nextBackAndOthers(String input){
  if(input.substring(0, 4) == "next" & input.length() == 4){
    receivedChar = "starting next song...";
    next();
  }
  else{backAndUnkown(input);}
}
```


archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

```
void backAndUnkown(String input){
  if(input.substring(0, 4) == "back" & input.length() == 4){
    receivedChar = "starting previous song...";
    back();
  }
  else{
    lyricsAndUnkwon(input);
  }
}
```

```
void lyricsAndUnkwon(String input){
  if(input.substring(0, 6) == "lyrics"){
      lyrics(input.substring(7));
    }
  else{ unkwon(input); }
}
```

```
void unkwon(String input){
  receivedChar = "unknown command..." + input;
}
```

```
void playOrAll(String input){
  if(input == "-a"){
    playAll();
  }
  else{
    if(input == "-r"){
      playRandom();
    }
    else{play(input);}
  }
}
```

- *Variables globales*

  Las variables globales utilizadas son las que aparecen en la imagen de abajo

  - melodyPin : Variable constante cuyo valor asignado es 3 corresponde al pin conectado al piezo.

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

- added : Variable constante con el valor 2, índice que corresponde al arreglo en el que se escribirán las notas y temor de la melodía agrega por el usuario.

- melodies: Es un arreglo de tres arreglos, cuya capacidad máxima es de 54(del cero al cincuenta y tres). En estos se encuentran las notas de las melodías(que son ints, pero estos mismos son representados como variables constantes).

- tempos: Como la variable melodies pero el contenido de los subarreglos son int cuyo fin es determinar la cantidad de tiempo que suena una nota.

- receivedChar: Del tipo String, es el medio para notificar al usuario de acerca de cómo resultó la tarea que le fue requerida al circuito.

- newData: Del tipo boolean, es un “semáforo” para evitar que al momento de ejecutar showNewData este no acceda a la escritura de la terminal.

- ARRAYSIZE: Int que determina la longitud del arreglo themes.

- theme: Int, indice del tema que se reproducirá.

- addI: Int que determina el índice donde se escribirá la siguiente nota.

- themes: Arreglo de Strings que son los títulos de las canciones que están pre almacenadas en el circuito.

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

```
#define melodyPin 3
#define added 2

//melodias
int melodies[3][53] = {
  { NOTE_C4, NOTE_C5, NOTE_A3, NOTE_A4, NOTE_AS3, NOTE_AS4, 0, 0, NOTE_C4, NOTE_C5, NOTE_A3, NOTE_A4,
    NOTE_AS3, NOTE_AS4, 0, 0, NOTE_F3, NOTE_F4, NOTE_D3, NOTE_D4, NOTE_DS3, NOTE_DS4, 0, 0,
    NOTE_F3, NOTE_F4, NOTE_D3, NOTE_D4, NOTE_DS3, NOTE_DS4, 0, 0, NOTE_DS4, NOTE_CS4, NOTE_D4,
    NOTE_CS4, NOTE_DS4, NOTE_DS4, NOTE_GS3, NOTE_G3, NOTE_CS4, NOTE_C4, NOTE_FS4, NOTE_F4, NOTE_E3, NOTE_AS4,
    NOTE_A4, NOTE_GS4, NOTE_DS4, NOTE_B3, NOTE_AS3, NOTE_A3, NOTE_GS3,},
  { NOTE_FS5,NOTE_FS5,NOTE_D5,NOTE_B4,NOTE_B4,NOTE_E5,NOTE_E5,NOTE_E5,NOTE_GS5,NOTE_GS5,NOTE_A5,NOTE_B5,
    NOTE_A5,NOTE_A5,NOTE_A5,NOTE_E5,NOTE_D5,NOTE_FS5,NOTE_FS5,NOTE_FS5,NOTE_E5,NOTE_E5,NOTE_FS5,NOTE_E5,
    NOTE_FS5,NOTE_FS5,NOTE_D5,NOTE_B4,NOTE_B4,NOTE_E5,NOTE_E5,NOTE_E5,NOTE_GS5,NOTE_GS5,NOTE_A5,NOTE_B5,
    NOTE_A5,NOTE_A5,NOTE_A5,NOTE_E5,NOTE_D5,NOTE_FS5,NOTE_FS5,NOTE_FS5,NOTE_E5,NOTE_E5,NOTE_FS5,NOTE_E5  },
  { }
};

int tempos[3][53] = {
};

String receivedChar = "";
boolean newData = false;
const int ARRAYSIZE = 2;
int theme = 0;
int addI = 0;
String themes[ARRAYSIZE] = {"mario", "take_on_me"};
```

- *Problemas durante el desarrollo*

  El siguiente es un listado de los problemas que surgieron durante el desarrollo:

  - Estructuras de datos para la representación: para la representación de los datos se había planeado la utilizar un diccionario donde la clave seria el titulo de la canción(un String) y el valor un conjunto de tuplas *X Y*, donde las *X* serian las notas y las *Y*. Para reemplazar el diccionario se optó por un arreglo que contiene los títulos de las canciones y otros dos arreglos de arreglos uno dedicado a almacenar las notas y otro dedicado a almacenar los tempos.

  - IF Vs switch case: para el código del método commands, hubiera sido preferible utilizar un switch case pero el mismo solo puede recibir como parámetro tipos específicos(Int, Char). Dado a esto se utilizaron las sentencias if y para mejorar su legibilidad se la dividió en múltiples métodos.

  - Tiempo estimado que se le asignó al desarrollo: para el desarrollo y realización del informe se determinó que se requerirían un máximo de 5 horas por semana. Este fue un error

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

de cálculo con respecto a la cantidad de tiempo que llevaría el resolver testear la terminal para asegurar de que no presenta errores con respecto a los inputs del usuario.

- Error durante el desarrollo del comando add -n: durante el desarrollo de comando add -n que se encontró un problema con respecto a la asignación de los valores que se introducían dentro del arreglo.
  además de las variables que ya están establecidas se habían definido las siguientes variables:

```
int addedM[] = {}; ← arreglo donde se ingresan las notas para el tema agregado por el usuario.
int addedT[] = {};  ← arreglo donde se ingresan los tempos para el tema agregado por el usuario
int addedI = 0;    ← índice del arreglo donde se escribirá la siguiente nota/tempo.
int addedS= 0;   ← cantidad de elementos de ambos arreglos.
```

código correspondiente al fallo:

```
if(input.substring(0,2) == "-n"){
        int x = input.substring(3, input.indexOf(',')).toInt();
        int y = input.substring(input.indexOf(',')+1).toInt();

        Serial.println( x );  ← print A
        Serial.println( y );  ← print B

        addedM[addedI] = x;
        addedT[addedI] = y;

        Serial.println( addedM[addedI] );  ← print C
        Serial.println( addedT[addedI] );  ← print D

        addedI = addedI + 1;
}
```

al momento de la ejecucion, si el input del usuario era “add -n 123, 7“ en los prints A y B que corresponden a las variables locales, definidas en las líneas anteriores, se printea correctamente los datos ingresados por el usuarios

print A = 123
print B = 7

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

pero en los print C y D que son el print de los datos que se obtienen de los arreglos se printea ambas veces el mismo dato para ambos prints(que siguiendo el ejemplo anterior se printeaba ambas veces el valor 7).

- Uso de Sublime Text para la escritura del código: para una mayor facilidad al momento de escribir el código se prefirió utilizar Sublime Text para redactar código y luego pasar el codigo a la plataforma online, tinkercad.

- *Documentación de los métodos*

  El siguiente es un listado de los métodos utilizados para desarrollados:

  - pinMode: configura el pin especificado en uno de sus dos posibles modo(input o output).
    - sintaxis: pinMode(pin, mode)

      https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/

  - Serial.begin: setea una cantidad de bits por segundo para la transmisión de datos a través del puerto serial.
    - sintaxis: Serial.begin(speed)
      Serial.begin(speed, config)

      https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/?queryID=undefined

  - Serial.println: printea datos como texto leible por humanos usando ASCII en el puerto serial.
    - sintaxis: Serial.println(val)

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

Serial.println(val, format)

- Serial.available: obtiene el número de bytes disponibles para lectura del puerto serial.
    - sintaxis: Serial.available()

      https://www.arduino.cc/reference/en/language/functions/communication/serial/available/?queryID=79f7c373f49d196ece3696c0d2c9a3e3

- Serial.readString: lee los caracteres del buffer serial en un String.
    - sintaxis: Serial.readString()

      https://www.arduino.cc/reference/en/language/functions/communication/serial/readstring/

- substring: retorna un segmento de un String.
    - sintaxis: myString.substring(from)
      myString.substring(from, to)

      https://www.arduino.cc/reference/en/language/variables/data-types/string/functions/substring/

- sizeof: retorna el número de bytes que ocupa un arreglo.
    - sintaxis: sizeof(variable)

      https://www.arduino.cc/reference/en/language/variables/utilities/sizeof/

- indexOf: localiza un Char o un String dentro de otro String
    - sintaxis: myString.indexOf(val)
      myString.indexOf(val, from)

      https://www.arduino.cc/reference/en/language/variables/data-types/string/functions/indexof/

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing

- digitalWrite: asigna un valor alto o bajo a un pin digital.
    - sintaxis: digitalWrite(pin, value)

      https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/?queryID=1edbbd5d630efff9effb731352de063c

- delayMicroseconds: pone en pausa el programa por una cantidad de tiempo delimitada.
    - sintaxis: delayMicroseconds(us)

      https://www.arduino.cc/reference/en/language/functions/time/delaymicroseconds/?queryID=d77baf105ae57f0a4d8479342ee977df

archivo con el código:
https://drive.google.com/file/d/1Xd5T4Eeb0h2ChXENOsE--hDOMe84tblQ/view?usp=sharing