Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

Color Rush

Para poner la mente a prueba

Autora:

Ana Lucía Vico Martínez

Nuestro juego

¿QUÉ PODEMOS HACER CON COLOR RUSH?

Un juego tipo arcade hecho con Arduino en el que pondrás a prueba tu memoria.

¿El reto? Memorizar secuencias de colores y repetirlas con los botones.

Pero puede ser tan extenso como queráis, podemos hacer también un piano, un juego de kahoot, …

Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

Materiales Necesarios

Materiales

- Arduino Uno
- Pantalla OLED I2C 1.3" (128x64)
- 5 Botones de colores
- 5 LEDs de colores
- 5 Resistencias 220–330Ω
- Buzzer
- Protoboard pequeña
- Cables

¿Sabéis identificarlos en vuestro kit?

Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

PASOS A SEGUIR

PASOS A SEGUIR

- **Prueba de Componentes:**

  Probaremos cada componente por separado para entender cómo funciona.

- **Diseño del funcionamiento del juego:**

  Crearemos la lógica y el código que hará que nuestro robot se mueva y reaccione.

- **Montaje del arcade:**

  Juntaremos todas las piezas para dar forma a nuestro juego.

- **Añadir Funcionalidades Extra:**

  Pensaremos en funciones nuevas y las añadiremos para personalizar nuestro juego.

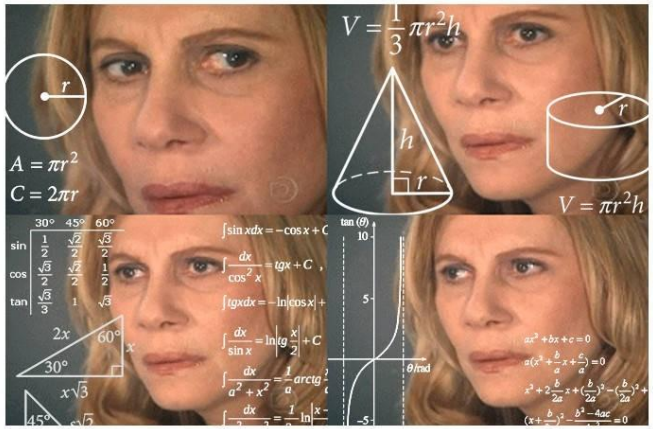
Prueba de Componentes

La protoboard

PROTOBOARD

Por ahora la protoboard en la prueba de componentes no va a ser necesaria, pero es muy importante para el montaje del proyecto, porque conectar toooodos los elementos a la vez no sería posible sin ella.

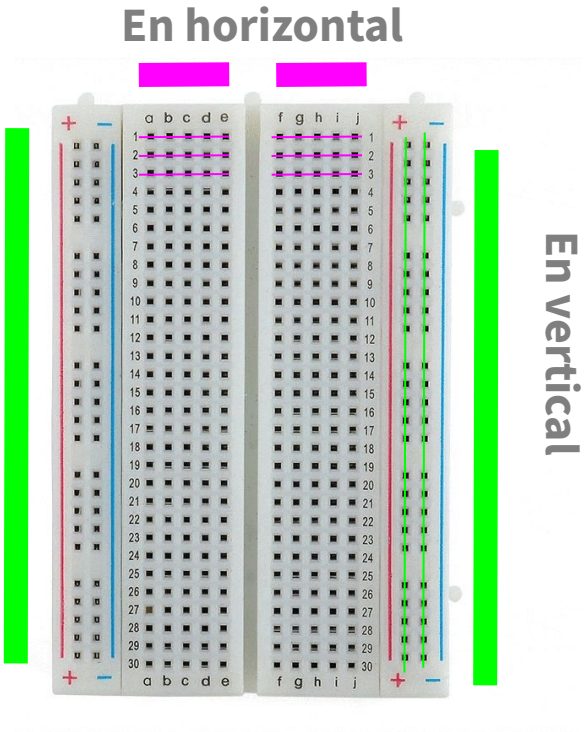
¿Cómo hacemos esto? ¿Todos los elementos tienen que conectarse al GND y 5V a la vez, pero no hay suficientes huecos para todos?



PROTOBOARD

¡¡¡No hay problema si utilizamos la **protoboard!!!**

Todas las filas del medio están conectadas horizontalmente y las dos columnas de los extremos están conectadas verticalmente. Normalmente, en los extremos pondremos la carga positiva y negativa, (5V y GND) y en medio para poder conectar otras cosas



Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

Prueba de Componentes

Conectar y programar un LED

LED

Un **LED** (Light Emitting Diode) es un componente electrónico que emite luz cuando se le aplica una corriente eléctrica.

- Los LEDs son conocidos por su eficiencia energética y larga vida útil.
- Pueden emitir luz en diferentes colores, dependiendo del material semiconductor utilizado en su construcción.

LED

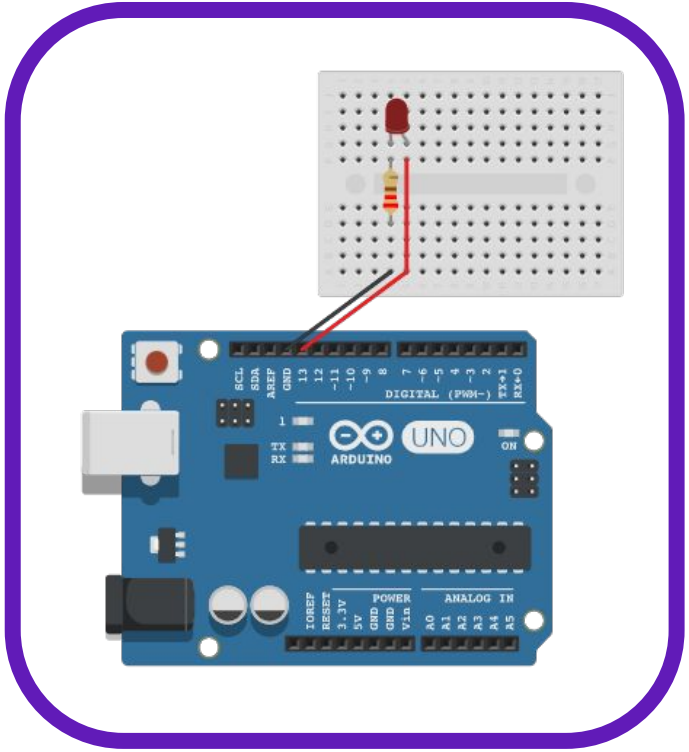
1. Para conectar un led necesitamos:

- El arduino UNO
- La protoboard
- Un led del color que queramos
- Una resistencia de 220Ω

Conexiones del LED:

- La patilla más larga se conecta con el **pin 13.** ■
- La patilla más corta se conecta **a la resistencia** y esta a **tierra** ■

(La resistencia limita la corriente que pasa a través del LED para evitar que se queme)



LED

Este led lo hemos conectado al pin 13. Por lo que queremos hacer el código para:

- Encender el led,
- Esperar un segundo,
- Apagar el led,
- Esperar un segundo.

Y esto se hace con:

pinMode(Numero_de_pin, tipo_de_elemento)

- Número de pin al que está conectado el led ___
- Tipo de elemento: **Output** emite, Input recibe información
  - ¿Que hace una luz, **emitir** o recibir? ____

Para que se encienda el led hay que “escribir” sobre él, utilizando:

- **digitalWrite**(Numero_de_pin, intensidad)
- Siendo **HIGH**, encendido y **LOW** apagado.

```
// Decidimos el pin del LED
int ledPin = 13;

void setup() {
     // Configurar el pin del LED
     pinMode(_______, _______);
}
void loop() {
     // Encender el LED
     digitalWrite(_______, _______);
     delay(1000); // Esperar un
segundo
     // Apagar el LED
     digitalWrite(ledPin, _______);
     delay(1000); // Esperar un
segundo
}
```

Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

Prueba de Componentes

Conectar y programar un Botón

Botón

Un pulsador (o botón) es un componente que permite o bloquea el paso de electricidad cuando lo presionas.

Botón

¿Cómo funciona en el Arduino?

- Cuando el botón **NO** está pulsado, el pin está en **HIGH** (encendido).
- Cuando el botón **SÍ** se pulsa, se conecta a tierra (GND) y el pin cambia a **LOW** (apagado).

Esto se consigue gracias a una configuración especial llamada **INPUT_PULLUP** que le dice al Arduino:

“Mantén el pin en HIGH por defecto, y si lo bajo a GND, es que el botón se ha pulsado”.

Para poder utilizarlo necesitamos lo que llamamos **CONDICIONALES**

¿Qué son los condicionales en español?

En programación, los condicionales son como las decisiones.

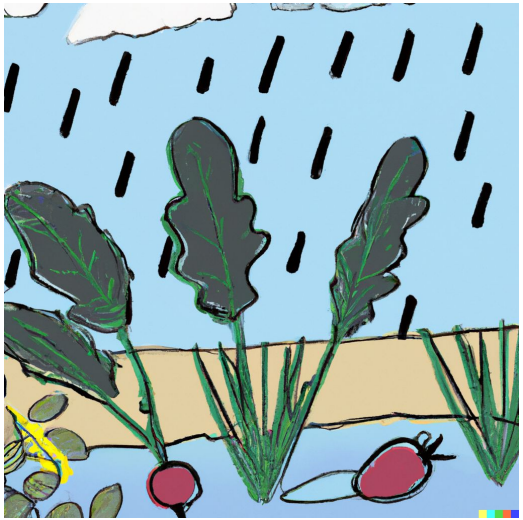
por ejemplo:

```
si (tienes hambre) {
    come una fruta;
} sino si (tienes sed) {
    bebe agua;
} sino {
    vete a jugar y diviértete;
}
```

- Pero en programación no podemos usar “si”, “si no” … hay que hacerlo en inglés…

  ¿Cómo se dice “**Si** tienes hambre, come algo” en inglés?

  If you are hungry, eat something



- Pues esto es exactamente igual:

```
if (está lloviendo){
    muestra por pantalla que está
    lloviendo
}
```

Botón

Y ahora nos quedarían los **símbolos de comparación.**

- ❑ > Mayor que
- ❑ < Menor que
- ❑ == Igual a
- ❑ != Distinto de

3 es … que 6
6 es … que 6

```
int edad = 15;
if (edad > 18) {
   // Acción si la edad es mayor a 18
   imprimir "Eres mayor de edad"
} else {
   // Acción si la edad no es mayor a 18
   imprimir "Eres menor de edad"
}
```

Botón

¿Cómo podemos utilizar los condicionales con los botones?

- Nos ahorra poner resistencias externas.
- Hace que el montaje sea más fácil y limpio.
- El código es sencillo de entender

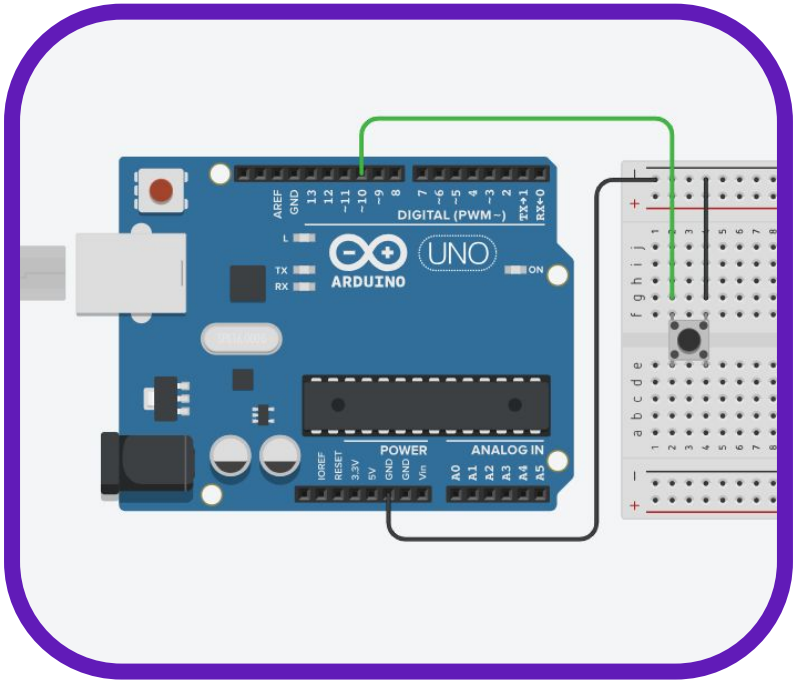
```
if (digitalRead(boton) == LOW) {
  // El botón ha sido pulsado
}
```

Botón

1. Conectamos el botón al arduino:

Colocaremos el botón en mitad de la protoboard, y:

- Una de las patillas las conectaremos al **pin 10.**
- La otra patilla la conectaremos al **GND (tierra)**.



2. Hacemos el código que haga que funcione nuestro botón

El código que vamos a crear, va a encender y apagar el led que incluye el arduino, que se encuentra en la posición 13. Y vamos a hacer que cuando pulsemos el botón (y lo mantengamos pulsado) se encienda la luz.

- digitalRead(boton) == LOW → Se pulsa el botón
- digitalRead(boton) == HIGH → No se pulsa el botón
- digitalWrite(led, HIGH) → Se enciende el led
- digitalWrite(led, LOW) → Se apaga el led

boton

```
const int boton = 10;       // Pin donde conectamos el botón
const int led = 13;         // Pin del LED (puede ser el integrado)

void setup() {
  pinMode(boton, INPUT_PULLUP); // Activamos resistencia interna
  pinMode(led, OUTPUT);         // El LED será una salida
}
```

```
loop{
     Si (al leer el botón está pulsado)
          Escribimos en el led que se
          encienda
     Sí no
          Escribimos en el led que se
          apague
}
```

Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

Prueba de Componentes

Conectar y programar zumbador

ZUMBADOR

Un zumbador es un pequeño componente que emite sonidos o pitidos cuando le llega electricidad. En nuestro proyecto lo usaremos para:

✅ Aciertos

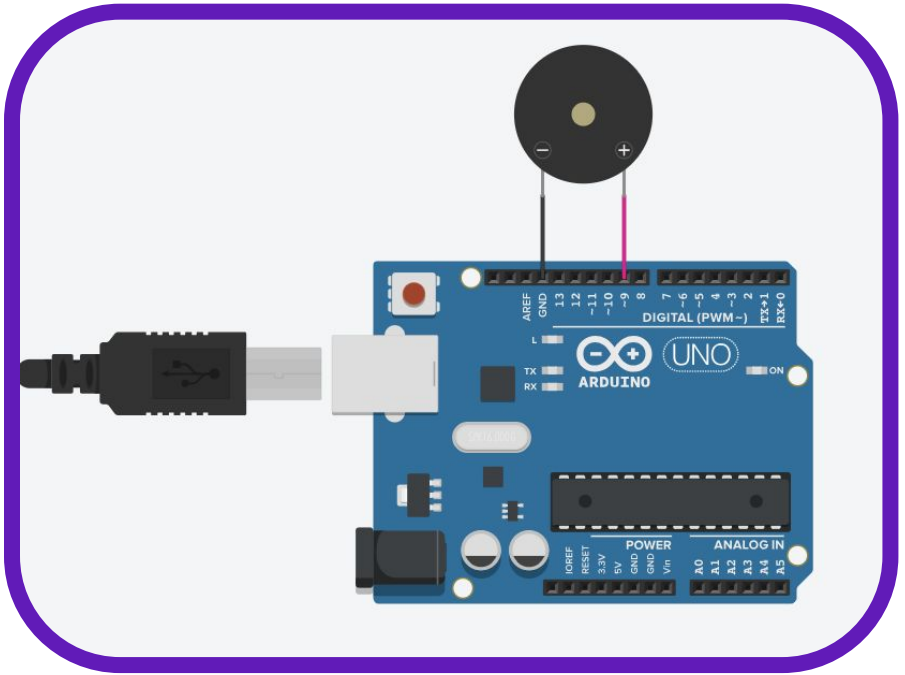
❌ Errores

🔔 Inicio o final del juego

Zumbador

1. Conectamos el sensor al arduino

   - ❏ Positivo **Pin 9**
   - ❏ Negativo - **Tierra**

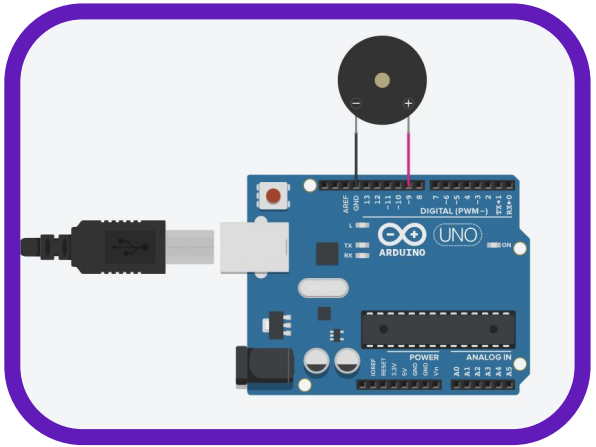


ZUMBADOR

- Utilizamos la función **tone()** para generar tonos con el zumbador.
  - Esta función toma como parámetros el pin de salida y la frecuencia del tono que deseamos generar.

```
En esta imagen, si ponemos:
     tone(9 ,1000)

Sonará, sin parar a 100hz
```



ZUMBADOR

2. Utilizamos el sensor:

- [ ] **tone(pin, tonoHz);**
- [ ] **noTone(pin)**
- [ ] **delay(1000);**

PruebaZumbador

```
const int zumbadorPin = 9;

void setup() {
  // Configurar el pin del zumbador como salida
  pinMode(zumbadorPin, OUTPUT);
}

void loop() {
```

```
ponemos el tono de 1000hz en el pin
esperamos 1 segundo

quitamos el tono del pin
esperamos dos segundos
```

```
}
```

Oficina de Software Libre Universidad de Granada

UNIVERSIDAD DE GRANADA

Prueba de Componentes

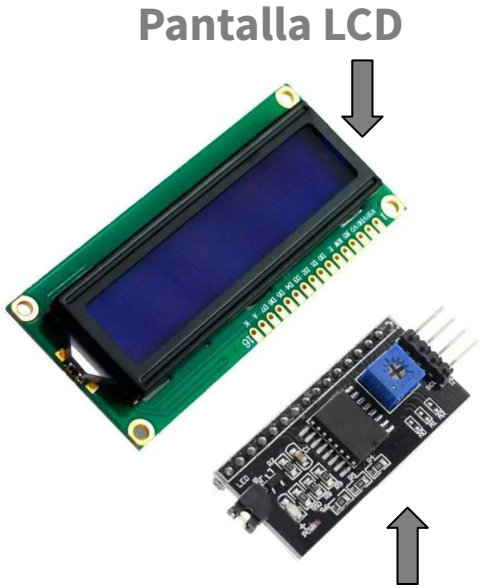
Conectar y programar pantalla LCD

Pantalla LCD

Una pantalla LCD (Liquid Crystal Display) es como una pantallita que muestra texto. En este proyecto usamos una LCD de 16 columnas y 2 filas, lo que significa que puede mostrar hasta 32 caracteres.

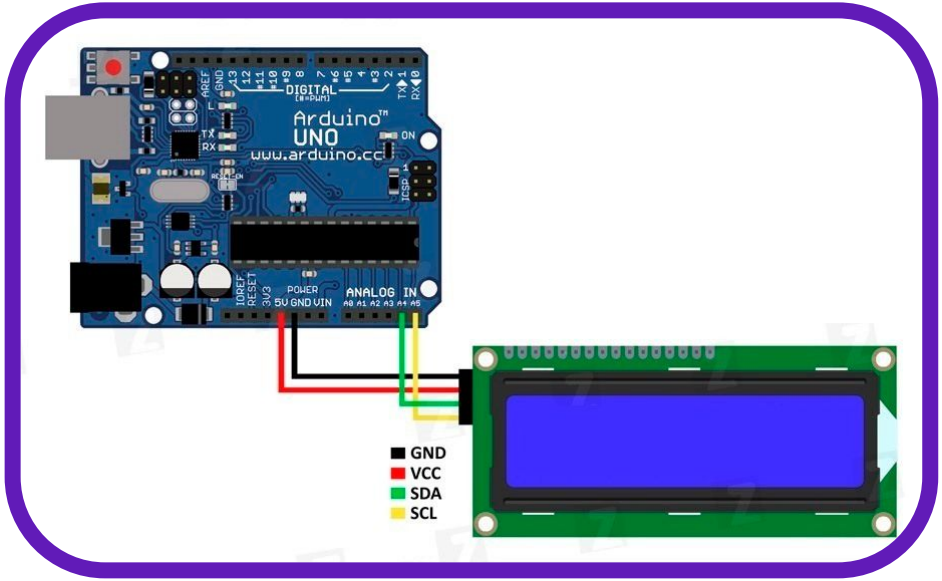
Sirve para mostrar mensajes como "¡Bienvenida!", "Nivel 1", "Game Over", etc., así que es super útil para que el juego te diga lo que está pasando en cada momento

- Además, nuestra pantalla tiene un adaptador I2C, para que solo necesitemos 4 cables.



Pantalla LCD

1. Conectamos la pantalla LCD al arduino:
   - VCC al **pin de 5V**, positivo
   - GND al **pin GND**, negativo
   - SDA al **pin A4**
   - SCL al **pin A5**



Pantalla LCD

Para utilizar la pantalla LCD hace falta incluir las librerías necesarias:

- Hay **dos formas de incluir librerías** a la biblioteca de nuestro arduino.
  - Si tenemos el archivo comprimido con extensión .zip
  - Buscarlas en el repositorio de Arduino

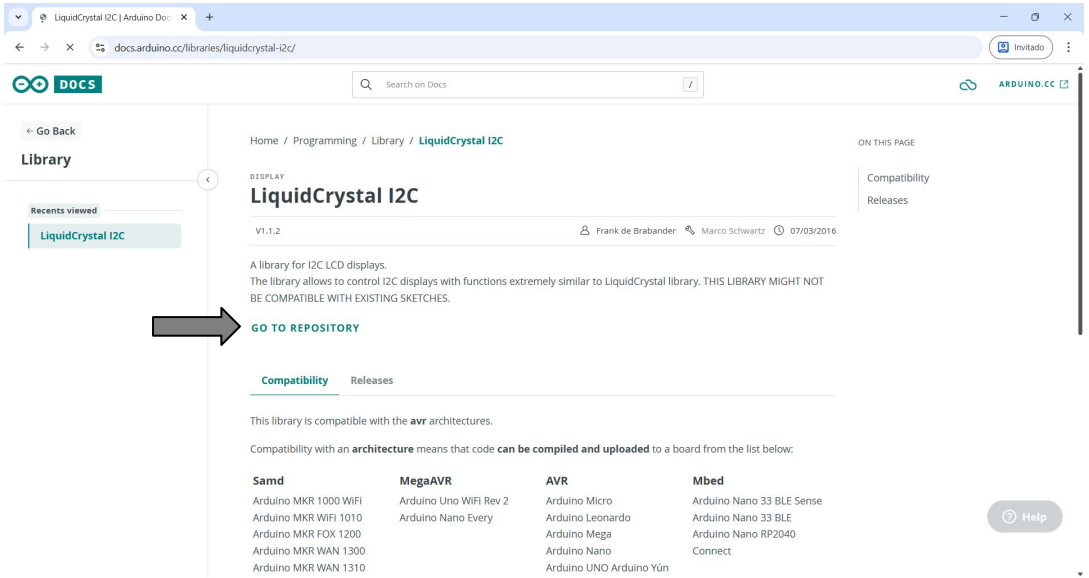
- Para la pantalla tenemos el archivo comprimido con el nombre **Arduino-LiquidCrystal-I2C.zip**
- La **descargamos** y la incluimos de la 1º forma.

Pantalla LCD

1. Incluir en el IDE la librería LiquidCrystal.h

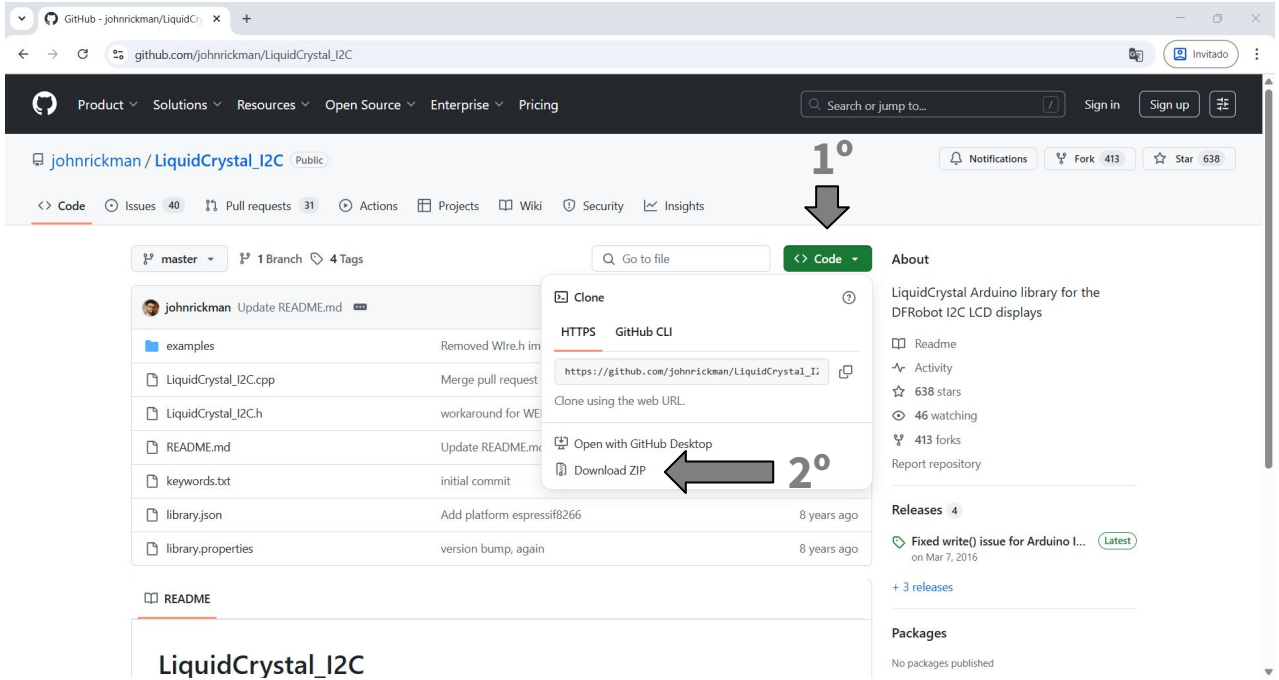
Vamos al enlace y descargamos la librería haciendo clic en download

https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/



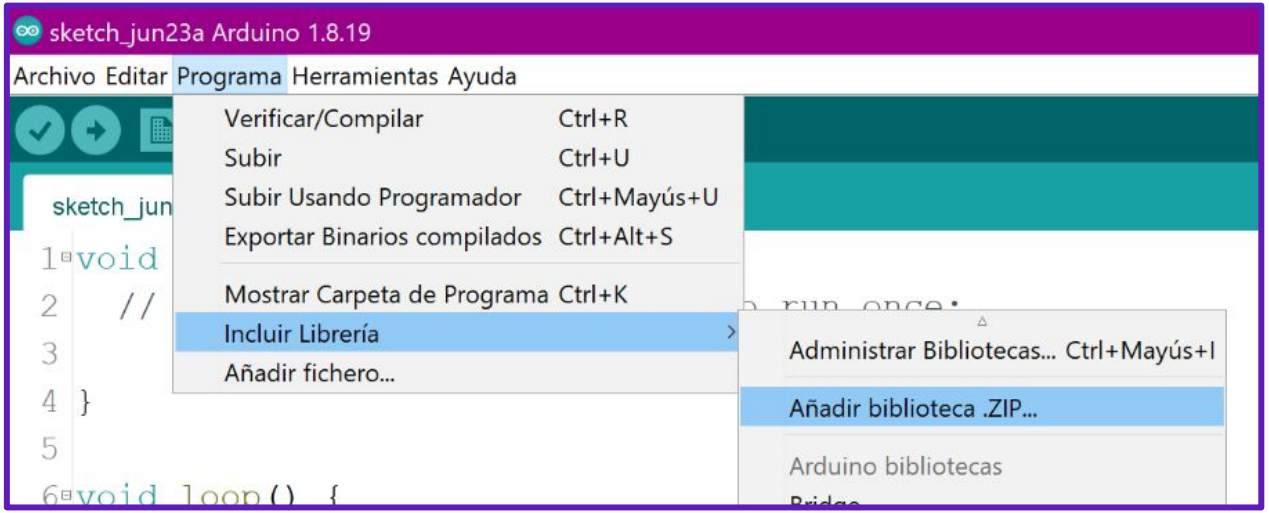
PANTALLA LCD

1. Incluir en el IDE la librería LiquidCrystal.h



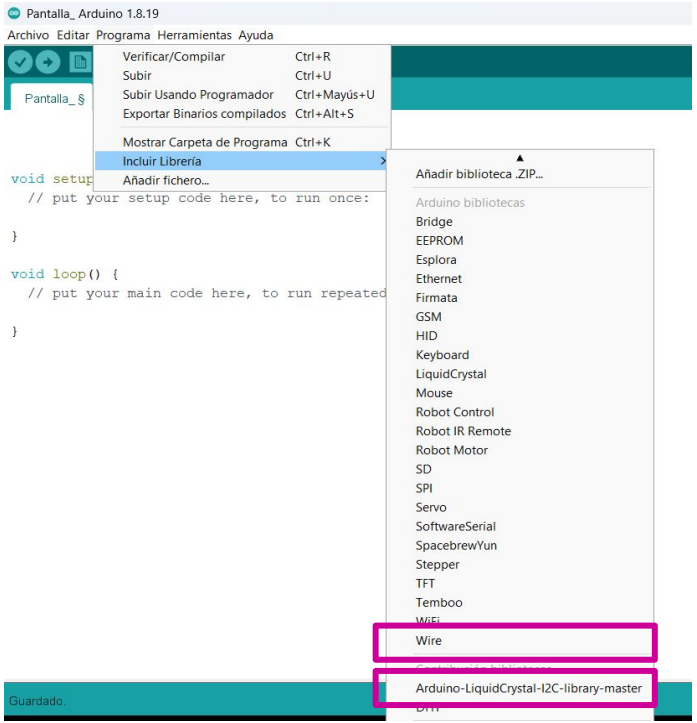
Pantalla LCD

1. Incluir en el IDE la librería LiquidCrystal.h



Pantalla LCD

2. **Para poder usar la librería tenemos que añadirla en el archivo. Vamos a añadir también wire.**



Pantalla LCD

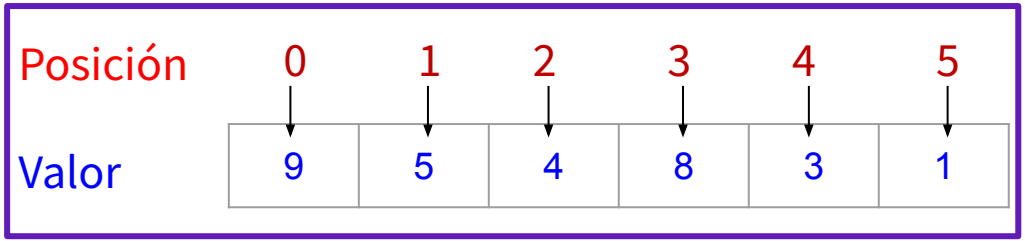
3. Declaramos un objeto basándonos en nuestra pantalla

- Para ello utilizaremos la siguiente **función de la librería LiquidCrystal_I2C** que acabamos de incluir.
  - **LiquidCrystal_I2C lcd(0x27, 16, 2);**
- Se inicializa con:
  - 0x27 porque se inicializa en esa dirección
  - 16 = número de caracteres por línea
  - 2 = número de líneas

- Vale pero, ¿qué tenemos que hacer y cómo? Vaya locura no entendí nada…
  - Poco a poco, paso a paso noseque

4. Aprendemos a usar la librería para utilizar cada punto de nuestra pantalla.

- ❑ Antes de empezar a usar las funciones de la librería necesitamos saber cómo usar las posiciones de cada línea.
- ❑ ¿Sabes qué es un **vector**?



- El vector tiene **posiciones**
- La primera posición **siempre empieza en 0**
- **Cada posición** tiene **un valor** (en la imagen, la posición 0 tiene valor 9, la posición 4 tiene valor 3…)
- El tamaño máximo del vector es el número de posiciones que tiene. En este ejemplo el vector tiene **6 posiciones que van del 0 al 5**.
- En nuestra pantalla, habrá dos vectores, que irán del… **0 al 15**

Pantalla LDC

4. Aprendemos a usar la librería para mostrar mensajes por pantalla:

- Para poder utilizar nuestra pantalla vamos a tener que aprender las funciones de la librería:
  - lcd.begin() se utiliza para inicializar la comunicación con la pantalla LCD.
  - lcd.backlight() enciende la retroiluminación de la pantalla LCD.
  - lcd.setCursor(0, 0) establece la posición del cursor en la columna 0 y la fila 0.
  - lcd.print("Hola") muestra el texto "Hola" en la pantalla LCD.
  - lcd.clear() borra el contenido de la pantalla LCD.

¿Creéis que podríais hacer que la pantalla muestre en la primera línea “Hola”?

Pantalla LDC

4. Aprendemos a usar la librería para mostrar mensajes por pantalla:

- lcd.begin()
- lcd.backlight()
- lcd.setCursor(0, 0)
- lcd.print("Hola")
- lcd.clear()
- delay(2000)

pantalla

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
  // Inicia el LCD en la dirección 0x27, con 16 caracteres y 2 líneas

void setup()
{
}
```

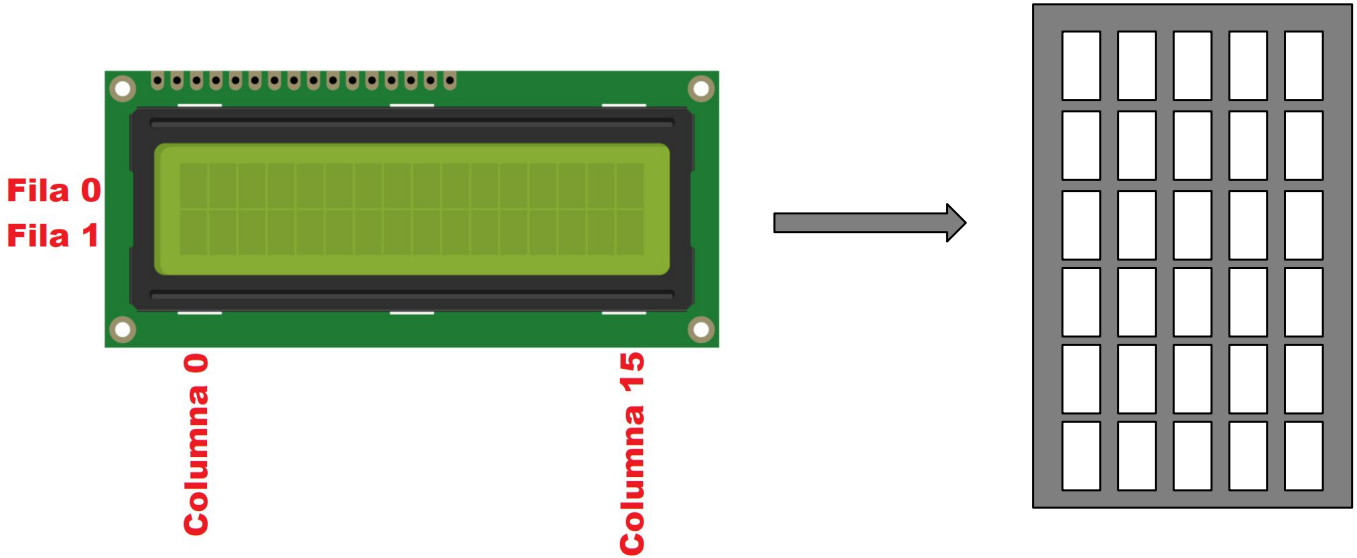
```
loop{
     inicio la pantalla
     luz de fondo
     empiezo a escribir en línea
     añado el texto que quiera
     espero 2 segundos
     limpio pantalla
}
```

Pantalla LCD

4. Aprendemos a usar la librería para mostrar mensajes por pantalla:

   EXTRA:

- ¿Molaría poder poner iconos un poco más complicados?
- Vamos a entrar más en profundidad en cómo se muestran las letras en pantalla:

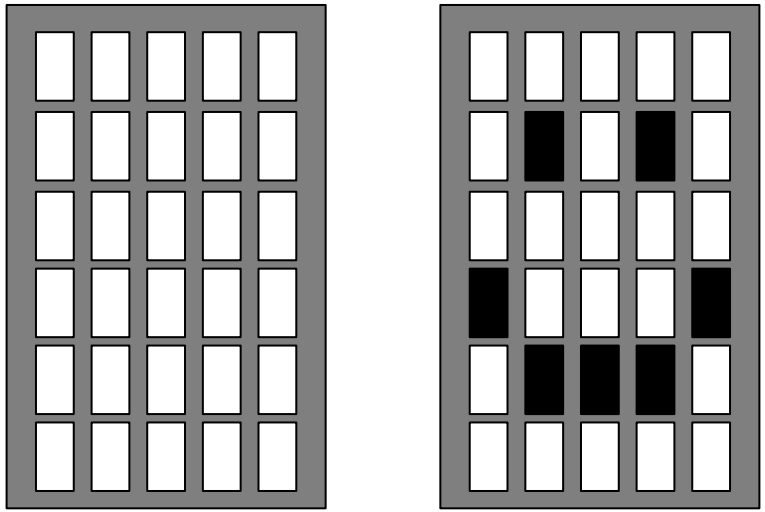


Pantalla LCD

4. Aprendemos a usar la librería para mostrar mensajes por pantalla:

   EXTRA:

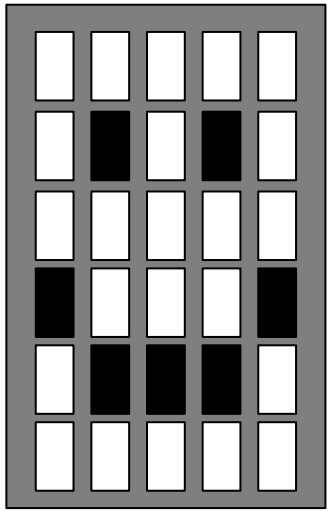
- ❑ Si quisiéramos poner una carita sonriente, ¿Qué puntos blancos colorearíais?



Pantalla LCD

4. Aprendemos a usar la librería para mostrar mensajes por pantalla:

EXTRA:



```
byte sonrisa[8] = {
  B00000,
  B01010,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
};

void setup()
{
  lcd.begin();
  lcd.createChar(0, sonrisa);
}

void loop()
{
  lcd.setCursor(0, 0);
  lcd.write((byte)0);
  lcd.setCursor(1, 0);
  lcd.print("Hola");
  lcd.setCursor(0, 1);
  lcd.print("Ingenieras!");
  delay(2000);
  lcd.clear();
  delay(1000);
}
```

Oficina de Software Libre Universidad de Granada

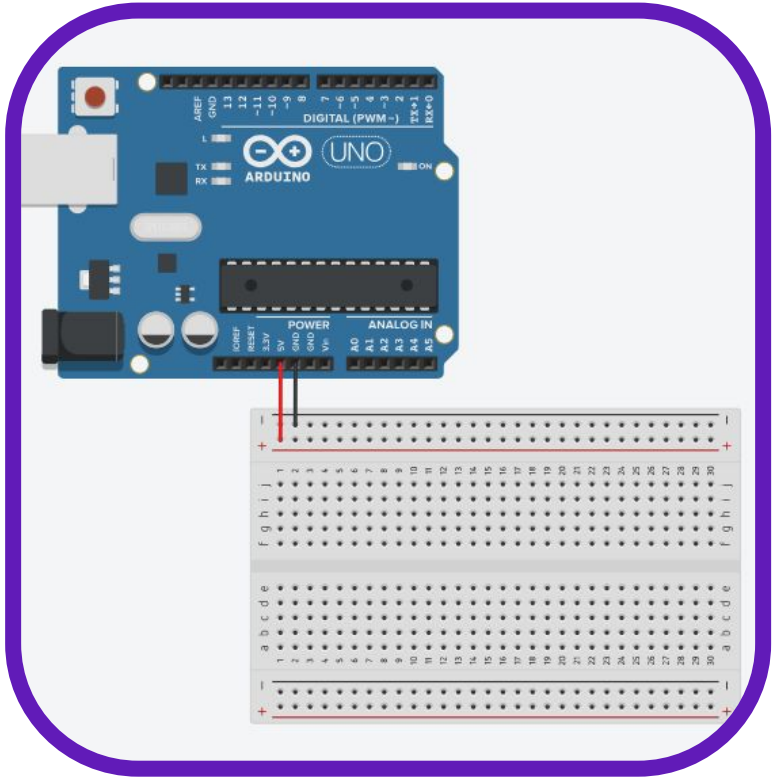
UNIVERSIDAD DE GRANADA

Prueba de Componentes

Conexión

Conexión

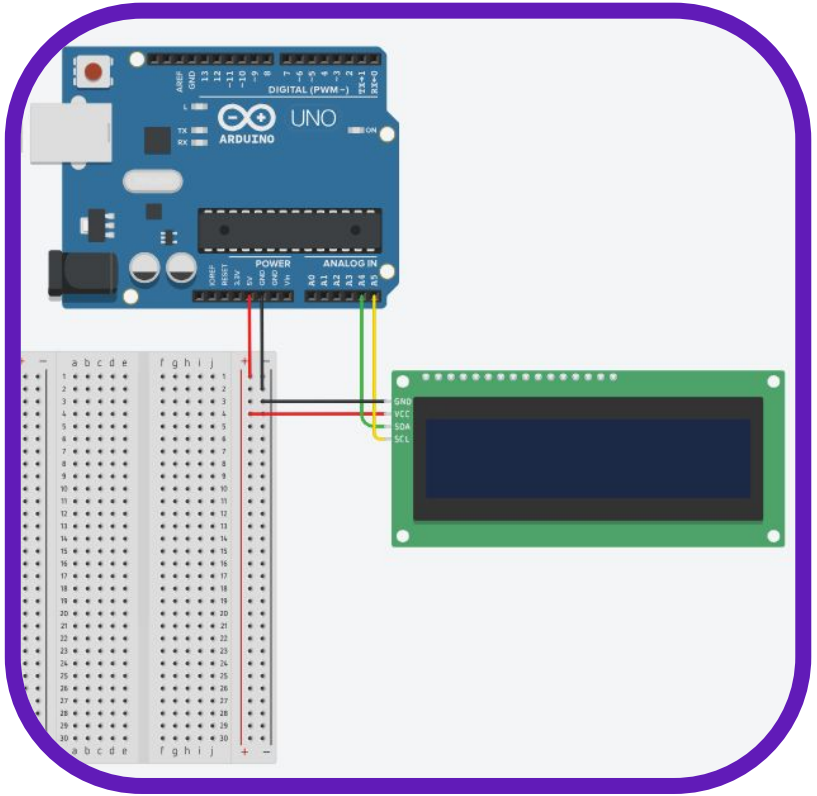
1. Protoboard:
   - Coloca la protoboard delante de ti
   - Conecta **5V** del Arduino **a la línea roja (+)** y **GND** a la **línea azul (-)** de la protoboard.
   - Así tendrás alimentación lista para todos los componentes



Conexión

2. El módulo I2C de la pantalla LCD tiene 4 pines:

- **GND** → tierra
- **VCC** → positivo
- **SDA** → A4 del Arduino
- **SCL** → A5 del Arduino

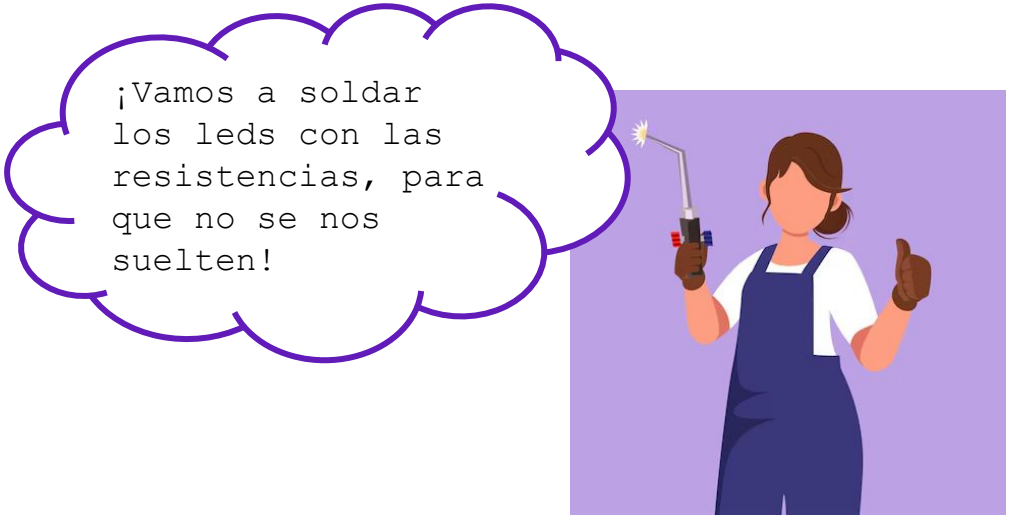


Conexión

3. Vamos a usar 6 LEDs, cada uno con su resistencia (220Ω)

Conecta el cátodo (la pata corta) a la línea azul (GND),

- LED1 → D2
- LED2 → D3
- LED3 → D4
- LED4 → D5
- LED5 → D6



Conexión

4. Coloca los botones sobre la protoboard. Conecta una patilla de cada botón a:

Botón 1 → D10

Botón 2 → A3

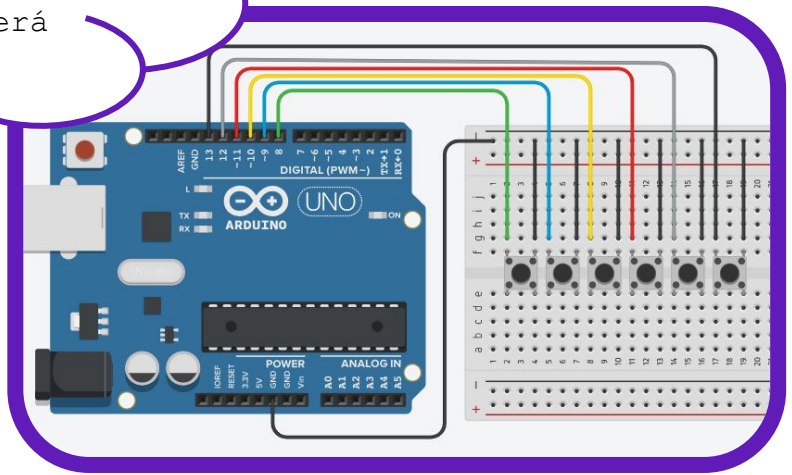
Botón 3 → A2

Botón 4 → A1

Botón 5 → A0

La otra patilla de todos los botones va a GND

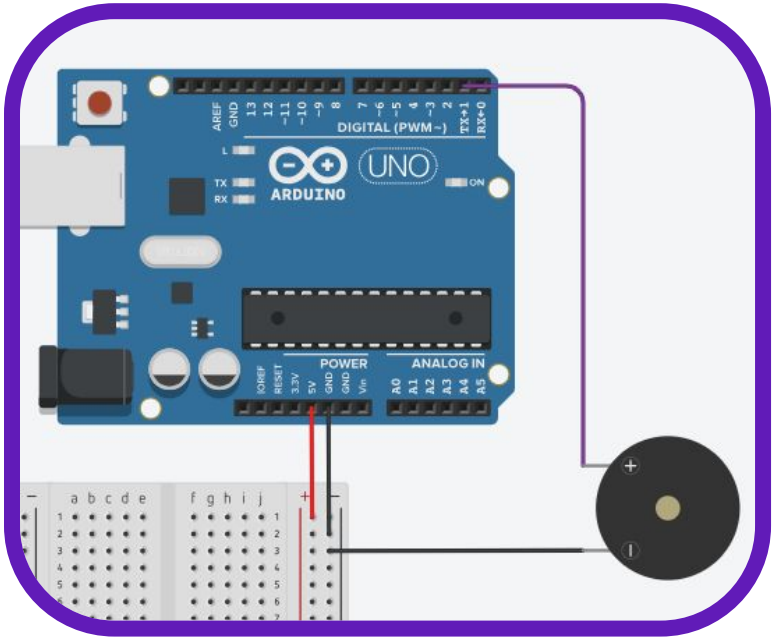
Recomendación de Ana:
Intentad que el color del botón sea el mismo que del cable, así será más fácil de identificar después



Conexión

5. Colocamos el buzzer/zumbador

- Pin positivo → pin 7
- Pin negativo → tierra



Prueba de Componentes

Funcionamiento de nuestro juego

Funcionamiento

Ahora que tenemos muchas partes del código sueltas (LEDs, botones, pantalla…), ha llegado el momento de organizarlo bien.

💡 ¿Y cómo lo hacemos? → Usando **funciones**: bloques de código que hacen tareas específicas.

¿Por qué son útiles las funciones?
✅ **Nos ayudan a no repetir código**
✅ **Hacen que todo se entienda mejor**
✅ **Si hay un error, sabemos dónde buscar**
✅ **Podemos usar y reutilizar partes del programa fácilmente**
✅ **Y el programa queda más limpio y pro** 💻✨

Funcionamiento

¿Con qué es comparable una función en vuestro día a día?

Imagina que estás cocinando y necesitas hacer varias veces una receta, por ejemplo, una ensalada de frutas.

Ingredientes(Entrada): Manzanas, plátanos, fresas, uvas, zumo de naranja.
Instrucciones (Cuerpo de la Función):

Lavar y cortar las frutas.
Mezclar las frutas en un bol.
Añadir el zumo de naranja.
Remover todo bien.

Plato Final (Salida): Una ensalada de frutas.
Reutilización: Cada vez que quieras hacer una ensalada de frutas, sigues la misma receta y no tienes que volver a inventarla.

Funcionamiento

De esa misma manera utilizamos las funciones. Hacen que el código sea más limpio y nos deja poder reutilizarlo.

Una función tiene dos elementos principalmente:

- Declaración: Definimos el nombre de la función.
- Cuerpo: Contiene el código que se ejecuta cuando usamos la función.

```
void decirHola() {
   Serial.println("¡Hola, Mundo!");
}
```

Funcionamiento

- Y para utilizarla, en lugar de poner:
  - `Serial.println("¡Hola, Mundo!");`
- Solo haría falta escribir:
  - `decirHola();`

Así podemos usar funciones para cualquier tarea que queramos repetir en nuestro programa.

Nota: Estas funciones se escriben **fuera del setup{} y fuera del loop{}, pero se llaman desde el loop{}.**

Esto ahora mismo parece muy complejo para algo tan cortito pero os va a venir bien :)

Tipos de funciones – ¿Todas hacen lo mismo?

No todas las funciones son iguales. Algunas **hacen algo pero no devuelven nada** (como un aviso), y otras **te devuelven un resultado.**

Tipo	¿Qué hace?	Ejemplo
`void`	Sólo ejecuta instrucciones	`void decirHola()`
`int - números`	Devuelve un número entero	`int sumar(int a, int b)`
`String - palabras`	Devuelve una frase o palabra	`String saludar(String nom)`
`boolean - solo dice si o no`	Devuelve verdadero o falso	`boolean esCorrecto()`

Funcionamiento

Si una función tiene que devolver un valor, usamos la palabra mágica **return (ella, bilingüe)**

```
int sumar(int a, int b) {
  return a + b;
}
```

```
int → Lo que queremos devolver, un número
sumar → Cómo se llama la función
(int a, int b) → Lo que queremos sumar
return a + b; → Nos devuelve la suma de a y b
```

```
Ejemplo:
    int resultado = sumar(4, 5);  // resultado va a ser 9
```

Funcionamiento

¿Por qué organizar el código en funciones?

📦 **Divide y vencerás.** Unificar el código en funciones ayuda a:

- Evitar repetir líneas mil veces
- Que el programa se entienda mejor
- Poder hacer cambios sin romper todo
- Localizar errores más fácilmente

PARA NUESTRO JUEGO…

```
void mostrarSecuencia() { ... }
void leerBotones() { ... }
void mostrarGameOver() { ... }
```

Prueba de Componentes

Creación del código unificado

Creación del código unificado

1. Antes de preparar el juego, necesitamos marcar qué componentes vamos a usar y en qué pines están conectados.

   Esto se hace fuera del setup(), al principio del programa.

   ¿Qué cosas declaramos?

   📟 La pantalla LCD

   💡 Los pines de los LEDs (rojo, amarillo, verde, azul, blanco)

   🔘 Los pines de los botones

   📢 El pin del zumbador

   🎮 Algunas variables que controlan el juego:

Creación del código unificado

Esto es lo que tenéis que poner pero escrito en “modo código”, como hemos hecho antes…

Importar librería de pantalla LCD
Crear pantalla LCD con dirección 0x27 y tamaño 16x2
Asignar nombres a los pines de los LEDs:
- Rojo → pin 2
- Amarillo → pin 3
- Verde → pin 4
- Azul → pin 5
- Blanco → pin 6

Asignar nombres a los pines de los botones
- Botón 0 → pin 10
- Botón 1 → pin A3
- Botón 2 → pin A2
- Botón 3 → pin A1
- Botón 4 → pin A0

Asignar nombre al pin del zumbador → pin 7

Creación del código unificado

¡¡¡¡¡¡ESPERAD!!!!!!

Añadid esto también
(podeis copiarlo tal cual)

```
// Variables del juego
int secuencia[100];
int nivel = 1;
bool jugando = false;

// Nombres de colores
String colores[] = {"Rojo",
"Amarillo", "Verde", "Azul",
"Blanco"};
```

Creación del código unificado

2. **Dentro del setup:**
   - Iniciar la comunicación con la pantalla LCD y encenderla
   - Mostrar el mensaje de bienvenida en la pantalla
   - Esperar un momento y limpiar la pantalla
   - Mostrar el mensaje "Pulsa para jugar" en la pantalla
   - Configurar los pines de los LEDs como salidas
   - Configurar los pines de los botones como entradas con resistencia interna activada (INPUT_PULLUP)
   - Configurar el pin del zumbador como salida
   - **Generar una semilla aleatoria usando el pin A0**
     - **Esto sirve para que las secuencias cambien en cada partida (pasad a la siguiente diapositiva si no sabéis cómo)**

Creación del código unificado

¿Qué es eso de “semilla aleatoria”?

Una semilla es como un punto de partida.
Le dice al Arduino:
→ “Empieza a inventar desde este número”.
Así conseguimos que el juego cambie cada vez que lo usamos, y no sea siempre igual

```
randomSeed(analogRead(0));
```

- Leemos el valor de un pin analógico sin conectar (A0).
- Como ese pin capta pequeñas señales del ambiente, su valor cambia solo.
- Lo usamos como punto de partida para crear secuencias distintas cada vez

Creación del código unificado

3. **Dentro del loop:**
   1. Si aún no estamos jugando:
      - Esperar a que la jugadora pulse cualquier botón
      - Cuando lo haga, empezamos el nivel 1
      - Mostramos el mensaje de nivel por pantalla y por monitor

Creación del código unificado

2. Si ya estamos jugando:
   - Generar una secuencia de colores aleatoria para este nivel
   - Mostrar esa secuencia encendiendo los LEDs uno por uno con sonido

   - Luego, esperar a que la jugadora repita la secuencia:
   - Por cada color:
     - Esperar hasta que pulse un botón o pase el tiempo
       - Si se equivoca, mostrar “Game Over” y volver a empezar
       - Si acierta, seguir con el siguiente color

   - Si completa todo correctamente:
   - Subir al siguiente nivel
   - Mostrar mensaje de “¡Bien hecho!”

Creación del código unificado

4. Las funciones, lo ultimísimo:

Creación del código unificado

Función: leerBoton()

Detecta si se ha pulsado alguno de los 5 botones del juego y devuelve el número del botón que se ha pulsado (de 0 a 4).

```
Revisar cada botón del 0 al 4:
  Si el botón está pulsado:
    Esperar a que se suelte (para no contar doble)
    Devolver el número del botón
Si no hay ningún botón pulsado:
  Devolver -1
```

Creación del código unificado

Función: esperaSoltar(pin)

Evita que un botón cuente varias veces por una sola pulsación (esto se llama rebote del botón).

```
Esperar un poco
Mientras el botón siga pulsado:
  No hacer nada
```

Creación del código unificado

Función: generarSecuencia(nivel)

Crea una lista de colores aleatorios para que la jugadora tenga que memorizar.

```
Para cada paso del nivel:
  Elegir un número aleatorio entre 0 y 4
  Guardarlo en la secuencia
  Mostrar el número por el monitor
```

Creación del código unificado

Función: leerYVerificar(nivel)

Comprueba si la jugadora repite bien la secuencia.

```
Para cada paso del nivel:
  Esperar hasta que pulse un botón o pase el tiempo
  Si se equivoca o no pulsa a tiempo:
    Mostrar "Game Over"
    Hacer sonido de error
    Devolver false
  Si acierta:
    Mostrar mensaje y seguir
Si acierta todo:
  Devolver true
```

Creación del código unificado

Función: obtenerPinLed(numero)

Evita que un botón cuente varias veces por una sola pulsación (esto se llama rebote del botón).

```
Si el número es 0 → devolver pin 2 (rojo)
Si el número es 1 → devolver pin 3 (amarillo)
Si el número es 2 → devolver pin 4 (verde)
Si el número es 3 → devolver pin 5 (azul)
Si el número es 4 → devolver pin 6 (blanco)
```

Creación del código unificado

Función: tonoVictoria() y tonoError()

Crea una lista de colores aleatorios para que la jugadora tenga que memorizar.

```
Tocar varias notas alegres con el zumbador
Esperar un poco entre cada una
```

```
Tocar un tono grave
Esperar
Tocar otro tono grave más bajo
```