



ugr

Universidad
de Granada

PRÁCTICA 3

EXPERIMENTACIÓN CON ARDUINO

Periféricos y Dispositivos de Interfaz Humana

Autores:

Clara Sola Ruiz
Mario Casas Pérez

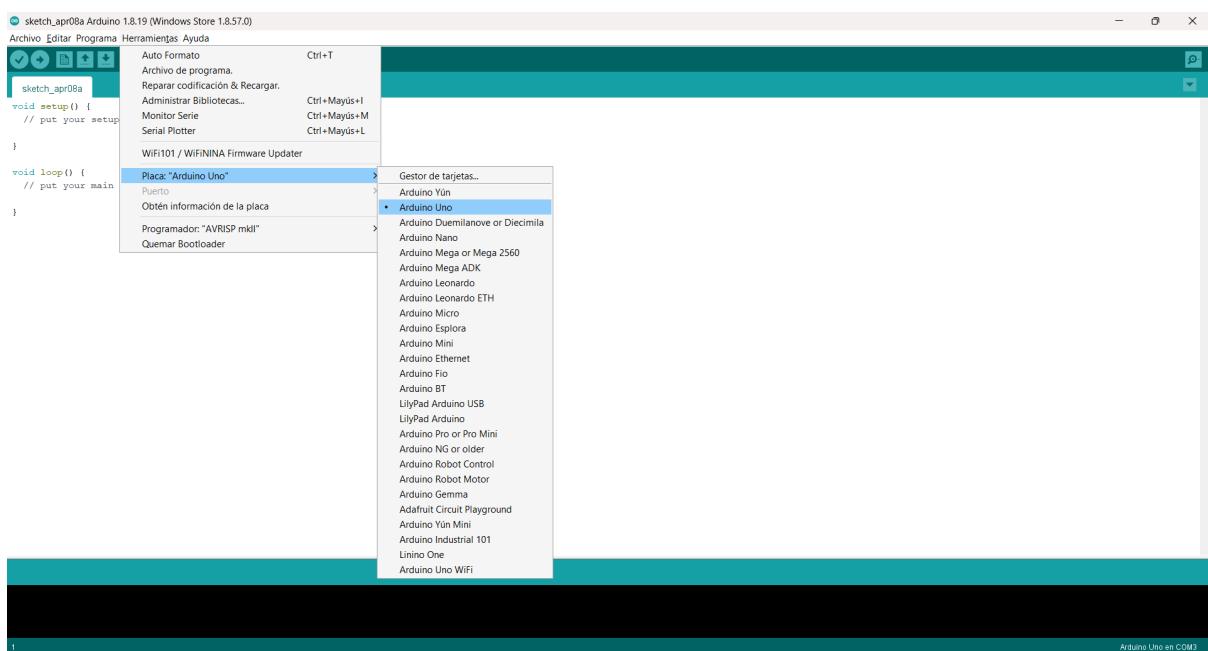
1. Introducción	3
2. Ejercicio mínimo 1	5
2.1. Tinkercad	5
2.2. Arduino IDE	7
3. Ejercicio mínimo 2	9
3.1. Tinkercad	9
3.2. Arduino IDE	11
4. Ejercicio opcional 1	13
4.1. Tinkercad	13
4.2. Arduino IDE	15
5. Ejercicio opcional 2	17
5.1. Tinkercad	17
5.2. Arduino IDE	20
6. Ejercicio opcional 3	21
6.1. Tinkercad	21
6.2. Arduino IDE	23
7. Ejercicio opcional 4	24
7.1. Tinkercad	24
7.2. Arduino IDE	26
8. Ejercicios adicionales	27
8.1. Sensor de temperatura (ejercicio 1 adicional)	27
8.1.1. Tinkercad	27
8.1.2. Arduino IDE	29
8.2. Juego de reflejos (ejercicio 2 adicional)	32
8.2.1. Tinkercad	32
8.2.2. Arduino IDE	34
8.3. Hola mundo en código morse (ejercicio 3 adicional)	36
8.3.1. Tinkercad	36
8.3.2. Arduino IDE	38
8.4. Termómetro de colores (ejercicio 4 adicional)	40
8.4.1. Tinkercad	40
8.4.2. Arduino IDE	42
9. Bibliografía	43

1. Introducción

Vamos a crear y comprobar el funcionamiento de varios sistemas de control basados en Arduino. Para ello, vamos a usar Tinkercad, la cual es una aplicación web gratuita y muy sencilla de usar. Aquí será donde diseñaremos nuestros circuitos de Arduino a la hora de realizar los diversos ejercicios de esta práctica.

La realización de cada ejercicio se hará tanto en el simulador Tinkercad como físicamente mediante Arduino. Además, para cada uno de los ejercicios se subirá a Github el video de su correcto funcionamiento y un enlace a lo realizado en el simulador.

Vamos a usar el programa Arduino IDE para poder implementar las funciones que queremos que se hagan cuando ya tengamos cada uno de los ejercicios construidos. Para ello, lo primero de todo es configurar dicho programa para que se conecte a la placa. Nos vamos a la parte del menú donde pone Herramientas y en *Placa* seleccionamos *Arduino Uno*.



Además, cuando conectemos la placa, debemos de especificar el puerto que se va a utilizar.



Luego, debemos de cargar el programa. Para ello primero debemos de pinchar el botón de *Verificar* que está en la parte superior izquierda y luego a *Subir* (al lado de *Verificar*) para que el programa empiece a funcionar en la placa.



The image shows two side-by-side screenshots of the Arduino IDE interface. Both windows have the title "sketch_apr08a Arduino 1.8.19 (Windows Store 1.8.57.0)". The left window shows the "Verificar" (Verify) button highlighted in green. The right window shows the "Subir" (Upload) button highlighted in green. Both windows display the same code for "sketch_apr08a".

```
sketch_apr08a Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda
Verificar

sketch_apr08a

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

sketch_apr08a Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda
Subir

sketch_apr08a

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Esto es lo que vamos a hacer para cargar cada uno de los programas de Arduino a la hora de realizar los ejercicios con el kit.

2. Ejercicio mínimo 1

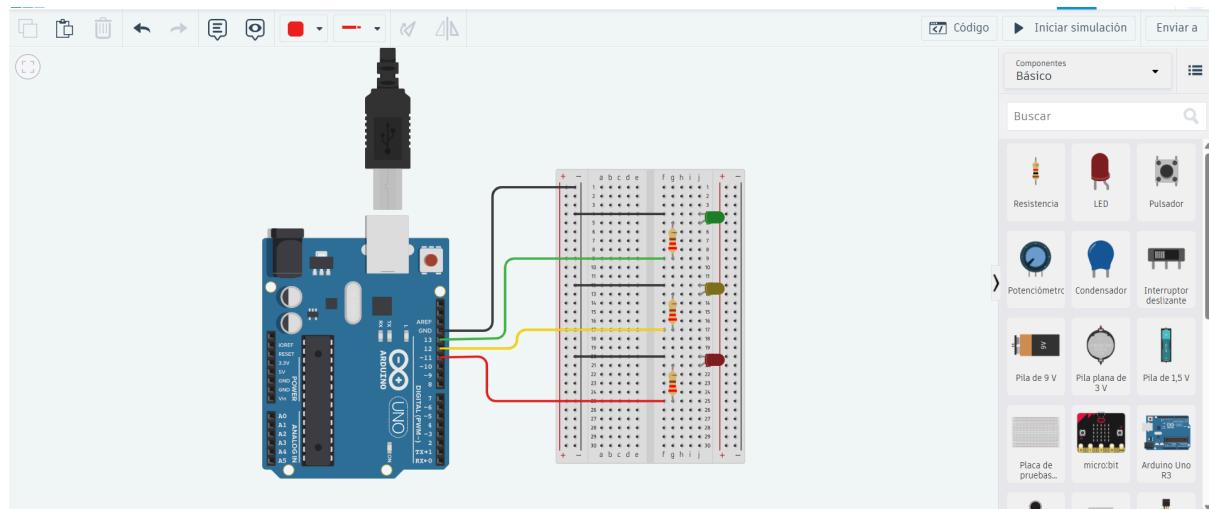
En este ejercicio vamos a implementar el programa de parpadeo de LED para que encienda y apague alternativamente 3 LEDs (uno rojo, otro amarillo y otro verde), conectados a las salidas digitales 11, 12 y 13 del Arduino, a un intervalo de 1.5 segundos. Además, se creará el esquema con Fritzing y se cargará el programa en Arduino IDE para comprobar que funciona correctamente.

Para hacer este ejercicio debemos de tener los siguientes componentes:

- Arduino UNO R3
- 3 resistencias de 220Ω
- 1 LED rojo
- 1 LED amarillo
- 1 LED verde
- 7 cables conectores
- 1 placa de pruebas

2.1. Tinkercad

Se ha cogido un *Arduino UNO R3* y una placa de pruebas para llevar a cabo esta tarea. Los LEDs están conectados a esta placa en donde los cátodos están conectados a tierra y los ánodos a las salidas digitales 11, 12 y 13. Se ha empleado una resistencia de 220Ω en cada ánodo de los LED para evitar sobretensiones en los diodos LED. El esquema es el siguiente:



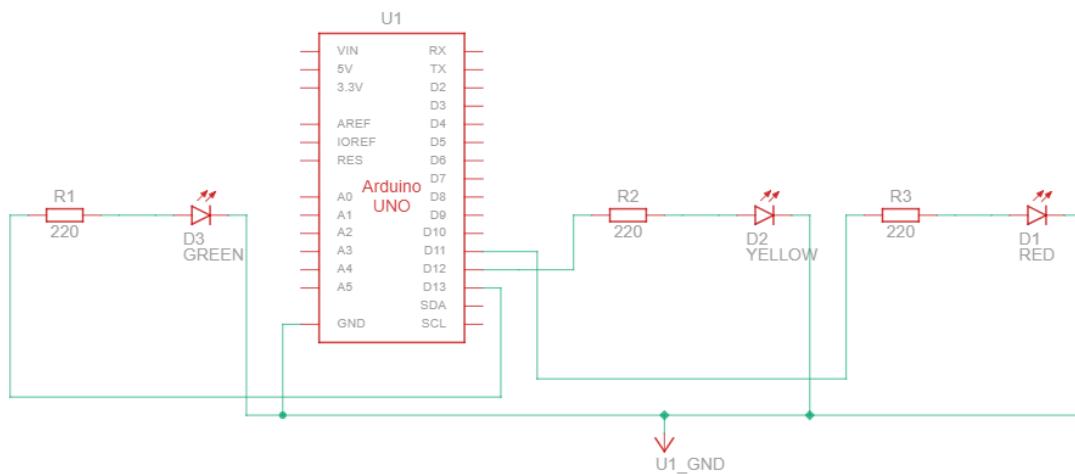
El código empleado para que se alterne los encendidos de los LEDs es el siguiente:

```
1 void setup()
2 {
3     pinMode(11, OUTPUT);
4     pinMode(12, OUTPUT);
5     pinMode(13, OUTPUT);
6 }
7
8 void loop()
9 {
10    digitalWrite(11, HIGH);
11    digitalWrite(12, LOW);
12    digitalWrite(13, LOW);
13    delay(1500); //Esperamos 1.5 segundos
14    digitalWrite(11, LOW);
15    digitalWrite(12, HIGH);
16    digitalWrite(13, LOW);
17    delay(1500); //Esperamos 1.5 segundos
18    digitalWrite(11, LOW);
19    digitalWrite(12, LOW);
20    digitalWrite(13, HIGH);
21    delay(1500); //Esperamos 1.5 segundos
22 }
```

En el primer método (setup) lo que hacemos es configurar los pines digitales 11, 12 y 13 como salida, lo que nos permite poder enviar señales para encender y apagar los LEDs.

En el segundo método (loop) se controla estos pines digitales 11, 12 y 13 de forma alternada para que se produzca el parpadeo, por lo que primeramente encendemos el pin 11 (LED rojo) mientras que el pin 12 y 13 están apagados. Tras 1500 milisegundos, encendemos el pin 12 (LED amarillo) dejando apagados el pin 11 y 13 para luego en otros 1500 milisegundos encender el pin 13 (LED verde) y dejar los los pines 11 y 12 apagados. Esto se hace indefinidamente.

El esquema con Fritzing es el siguiente:

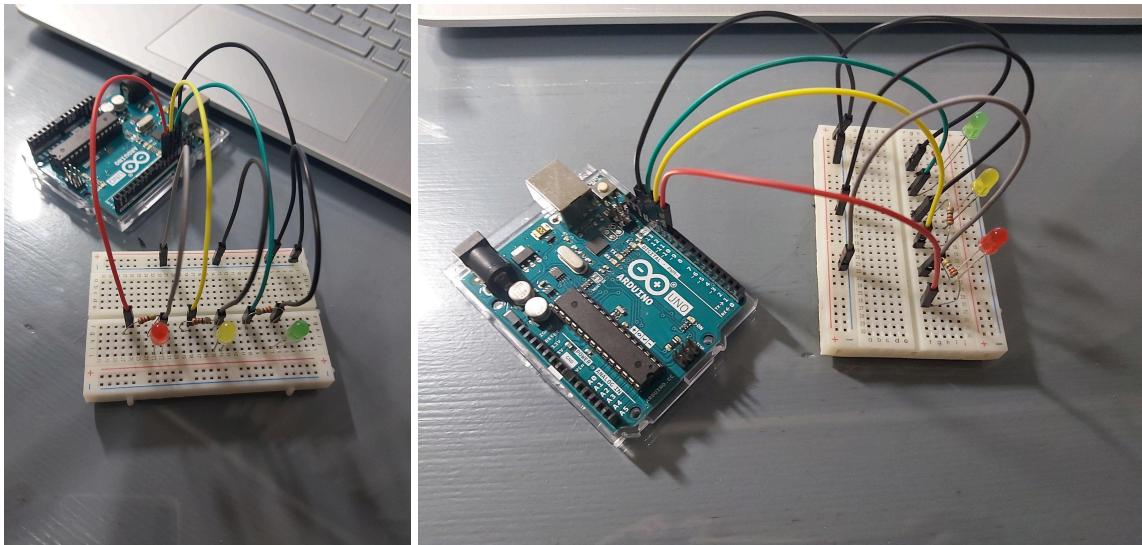


Para ver el proyecto desde Tinkercad, usar el enlace

https://www.tinkercad.com/things/8yglMyldOjV-ejercicio1-minimo?sharecode=enNHMF-4hd_UPv4UwMMZiKvPnM2M4zNJwAZeSBpMXAo

2.2. Arduino IDE

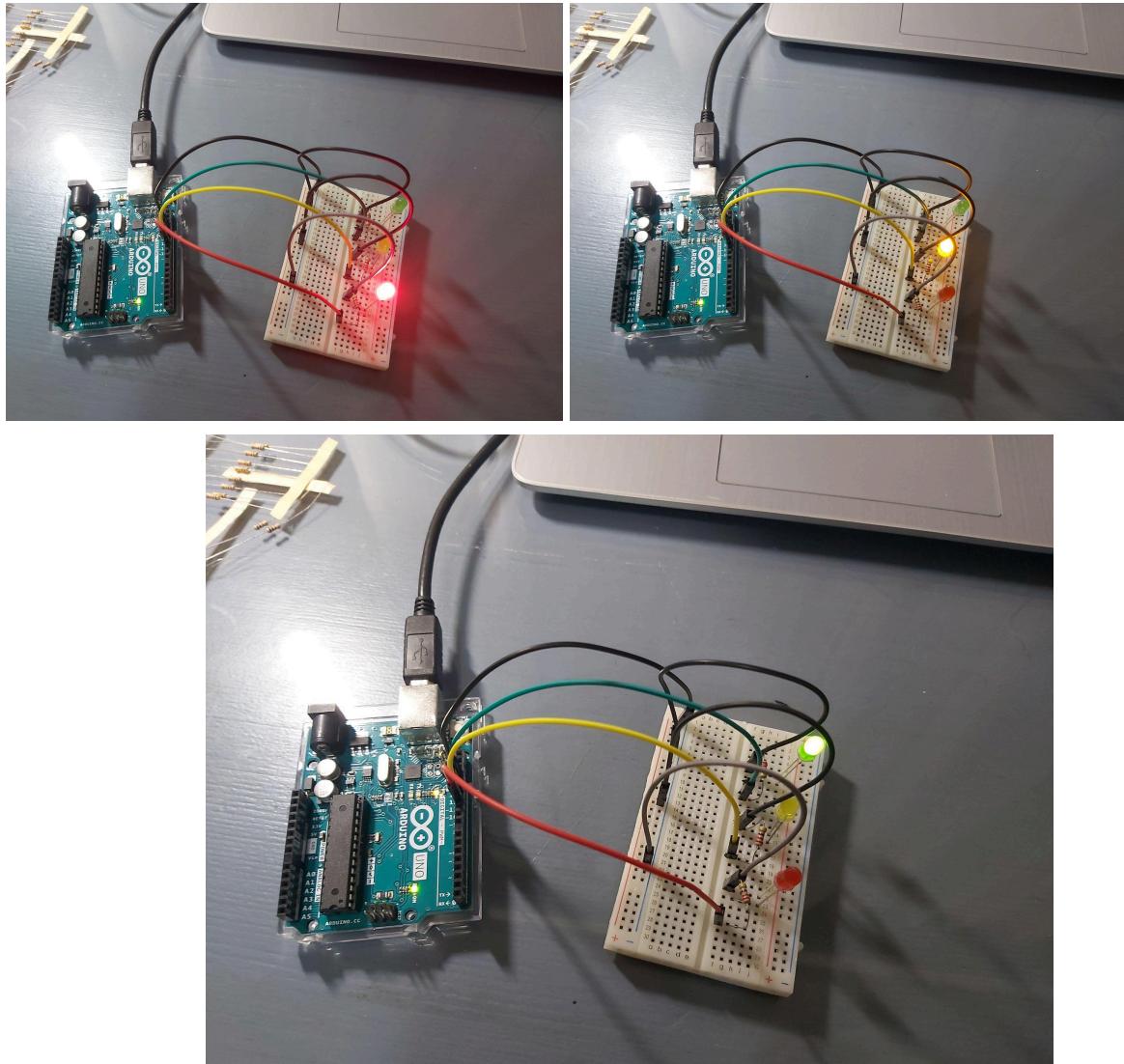
Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



En Arduino IDE cargamos el programa creado para hacer el ejercicio. Será el mismo código que se ha usado en el simulador Tinkercad.

```
ejercicio1-minimo
void setup()
{
    pinMode(11, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);
}

void loop()
{
    digitalWrite(11, HIGH);
    digitalWrite(12, LOW);
    digitalWrite(13, LOW);
    delay(1500); //Esperamos 1.5 segundos
    digitalWrite(11, LOW);
    digitalWrite(12, HIGH);
    digitalWrite(13, LOW);
    delay(1500); //Esperamos 1.5 segundos
    digitalWrite(11, LOW);
    digitalWrite(12, LOW);
    digitalWrite(13, HIGH);
    delay(1500); //Esperamos 1.5 segundos
}
```



También se dispone de un vídeo en la carpeta de github.

3. Ejercicio mínimo 2

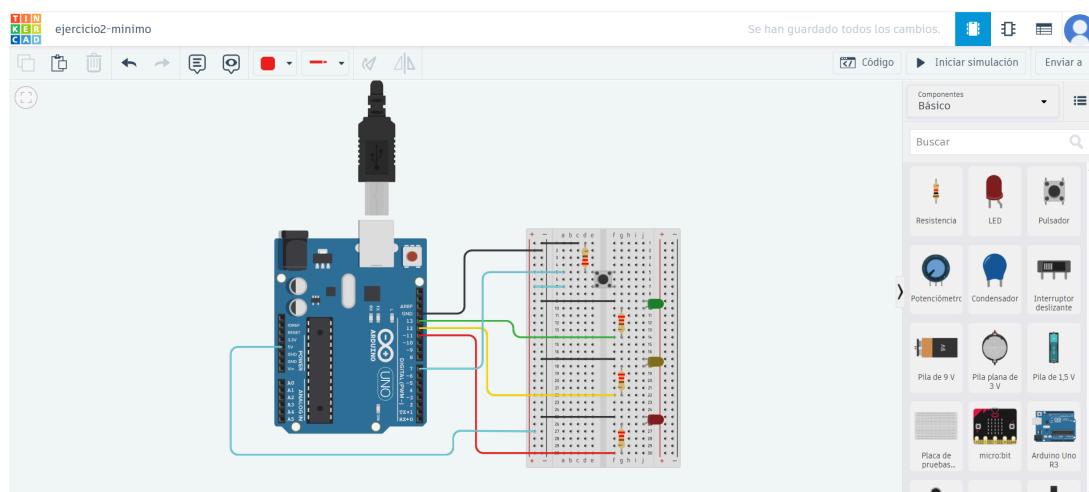
Para este ejercicio debemos de modificar el programa de parpadeo de LEDs para que se encienda el LED rojo solo cuando se pulse un interruptor conectado a la entrada digital 7, y en ese momento se apaguen los LEDs amarillo y verde. Además, se creará el esquema con Fritzing y se cargará el programa en Arduino IDE para comprobar que funciona correctamente.

Para hacer este ejercicio debemos de tener los siguientes componentes:

- Arduino UNO R3
- 4 resistencias de 220Ω
- 1 LED rojo
- 1 LED amarillo
- 1 LED verde
- 11 cables conectores
- 1 placa de pruebas
- 1 pulsador

3.1. Tinkercad

Se ha cogido un *Arduino UNO R3* y una placa de pruebas para llevar a cabo esta tarea. Los LED están conectados a esta placa en donde los cátodos están conectados a tierra y los ánodos a las salidas digitales 11, 12 y 13. Se ha empleado una resistencia de 220Ω en cada ánodo de los LED para evitar sobretensiones en los diodos LED. Además, se ha conectado un pulsador a la placa de pruebas en donde una de las patillas la tiene conectada a tierra y al pin digital 7, mientras que a la otra se le ha conectado un voltaje de 5V. El esquema es el siguiente:



El código empleado para que se encienda el LED rojo cuando se pulsa el pulsador y se apaguen los LEDs amarillo y verde es el siguiente:

```

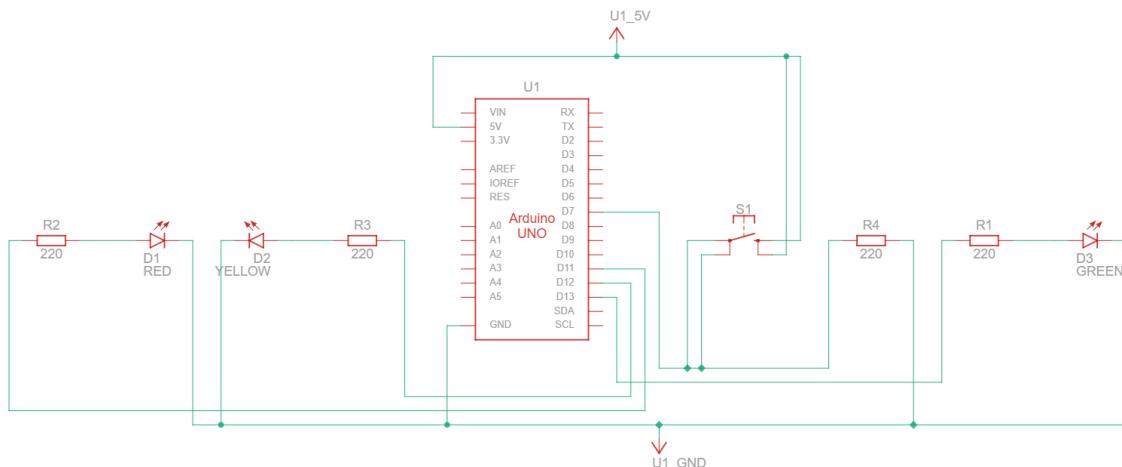
1 int pulsador = 0;
2
3 void setup()
4 {
5   pinMode(7, INPUT);
6   pinMode(11, OUTPUT);
7   pinMode(12, OUTPUT);
8   pinMode(13, OUTPUT);
9 }
10
11 void loop()
12 {
13   if (digitalRead(7) == HIGH) {
14     digitalWrite(11, HIGH);
15     digitalWrite(12, LOW);
16     digitalWrite(13, LOW);
17   } else {
18     digitalWrite(11, LOW);
19     digitalWrite(12, HIGH);
20     digitalWrite(13, HIGH);
21   }
22   delay(10); //Esperamos un poco para mejorar la simulación
23 }
```

En el primer método (`setup`) lo que hacemos es configurar el pin digital 7 como entrada, permitiendo leer el estado del pulsador que hemos conectado a ese pin. Además, configuraremos los pines digitales 11, 12 y 13 como salida, permitiéndonos enviar señales para encender y apagar los LEDs conectados a esos pines.

El segundo método (`loop`) controlamos el estado del pin 7 (`pulsador`) para hacer lo siguiente:

- Si se detecta un nivel alto (HIGH) en el pin 7, esto indica que el pulsador ha sido presionado. En este caso, el pin 11 (LED rojo) se enciende, mientras que los pines 12 (LED amarillo) y 13 (LED verde) se apagan.
- Si se detecta un nivel bajo (LOW) en el pin 7 significará que no se está pulsando el pulsador, por lo que los pines 12 (LED amarillo) y el pin 13 (LED verde) estarán encendidos, mientras que el pin 11 (LED rojo) estará apagado.

El esquema con Fritzing es el siguiente:

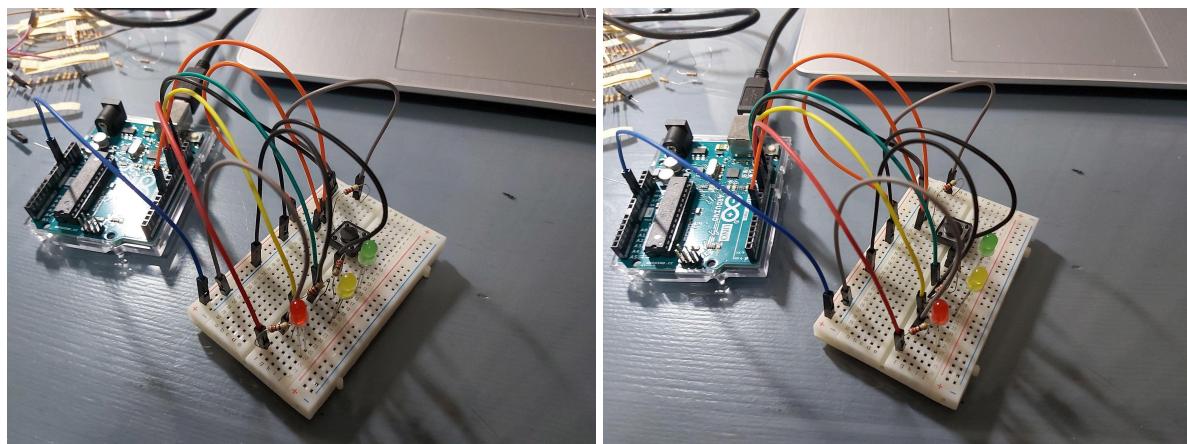


Para ver el proyecto desde Tinkercad, usar el enlace

https://www.tinkercad.com/things/hfpLveMf0vq-ejercicio2-minimo?sharecode=Tw-ja_L EgKa m2Xjmzm0iVvIlcRJyz7NkWhXFX69aK2Q

3.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



En Arduino IDE cargamos el programa creado para hacer el ejercicio. Será el mismo código que se ha usado en el simulador Tinkercad.



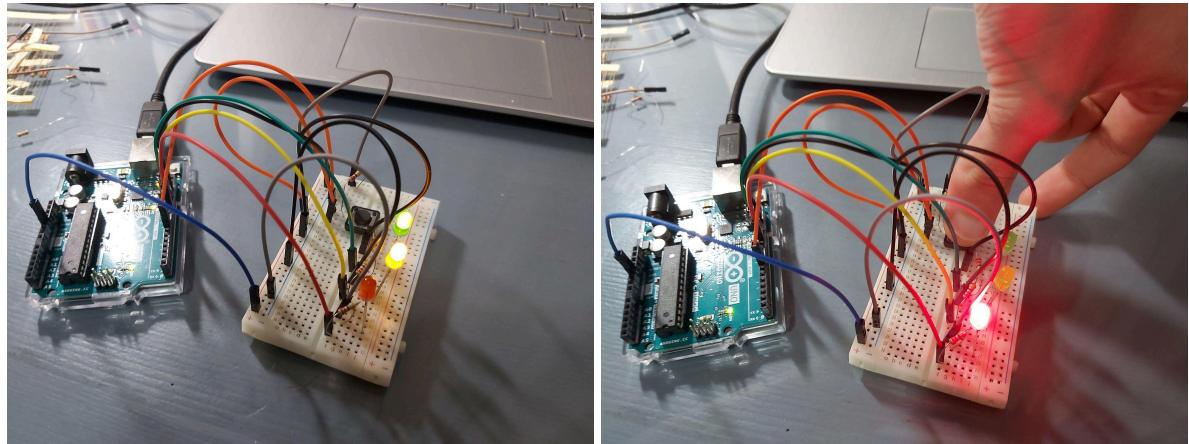
```
ejercicio2-minimo Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda

ejercicio2-minimo

int pulsador = 0;

void setup()
{
    pinMode(7, INPUT);
    pinMode(11, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);
}

void loop()
{
    if (digitalRead(7) == HIGH) {
        digitalWrite(11, HIGH);
        digitalWrite(12, LOW);
        digitalWrite(13, LOW);
    } else {
        digitalWrite(11, LOW);
        digitalWrite(12, HIGH);
        digitalWrite(13, HIGH);
    }
    delay(10); // Esperamos un poco para mejorar la simulación
}
```



También se dispone de un vídeo en la carpeta de github.

4. Ejercicio opcional 1

En este ejercicio se nos pide hacer una secuencia de LEDs, encendiendo y apagando 4 LEDs secuencialmente, de una forma parecida a las luces de “El coche fantástico”.

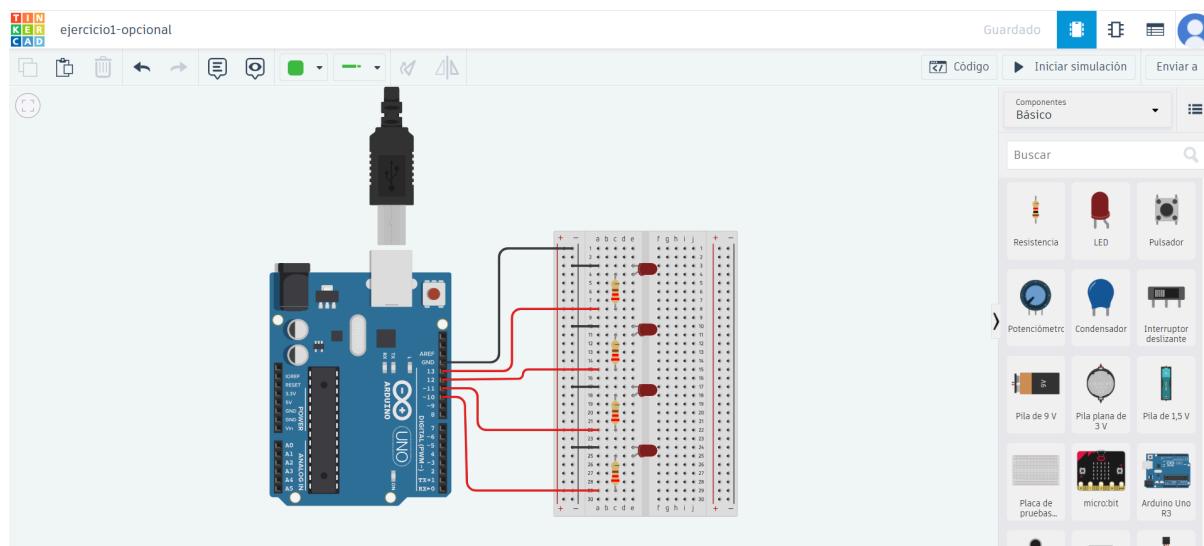
Para hacer este ejercicio debemos de tener los siguientes componentes:

- Arduino UNO R3
- 4 resistencias de 220Ω
- 4 LEDs rojos
- 9 cables conectores
- 1 placa de pruebas

Para la realización de este ejercicio, vamos a conectar los 4 LEDs a la salidas digitales 13, 12, 11 y 10 a un intervalo relativamente pequeño (150 milisegundos) para dar ese parecido a las luces del coche fantástico.

4.1. Tinkercad

Se ha cogido un *Arduino UNO R3* y una placa de pruebas para llevar a cabo esta tarea. Los LEDs están conectados a esta placa en donde los cátodos están conectados a tierra y los ánodos a las salidas digitales 13, 12, 11 y 10. Se ha empleado una resistencia de 220Ω en cada ánodo de los LED para evitar sobretensiones en los diodos LED. El esquema es el siguiente:



El código empleado para que se alterne los encendidos de los LEDs es el siguiente:

```
1 |void setup()
2 |
3 |  pinMode(13, OUTPUT);
4 |  pinMode(12, OUTPUT);
5 |  pinMode(11, OUTPUT);
6 |  pinMode(10, OUTPUT);
7 |
8 |
9 |void loop()
10 |
11 |  digitalWrite(13, HIGH);
12 |  digitalWrite(11, LOW);
13 |  delay(150); //Espera de 150 ms
14 |
15 |  digitalWrite(12, LOW);
16 |  delay(150); //Espera de 150 ms
17 |
18 |  digitalWrite(12, HIGH);
19 |  digitalWrite(10, LOW);
20 |  delay(150); //Espera de 150 ms
21 |  digitalWrite(11, HIGH);
22 |  digitalWrite(13, LOW);
23 |  delay(150); //Espera de 150 ms
24 |  digitalWrite(10, HIGH);
25 |  digitalWrite(12, LOW);
26 |  delay(150); //Espera de 150 ms
27 |
28 |  digitalWrite(11, LOW);
29 |  delay(150); //Espera de 150 ms
30 |
31 |  digitalWrite(11, HIGH);
32 |  digitalWrite(13, LOW);
33 |  delay(150); //Espera de 150 ms
34 |  digitalWrite(12, HIGH);
35 |  digitalWrite(10, LOW);
36 |  delay(150); //Espera de 150 ms
37 |}
```

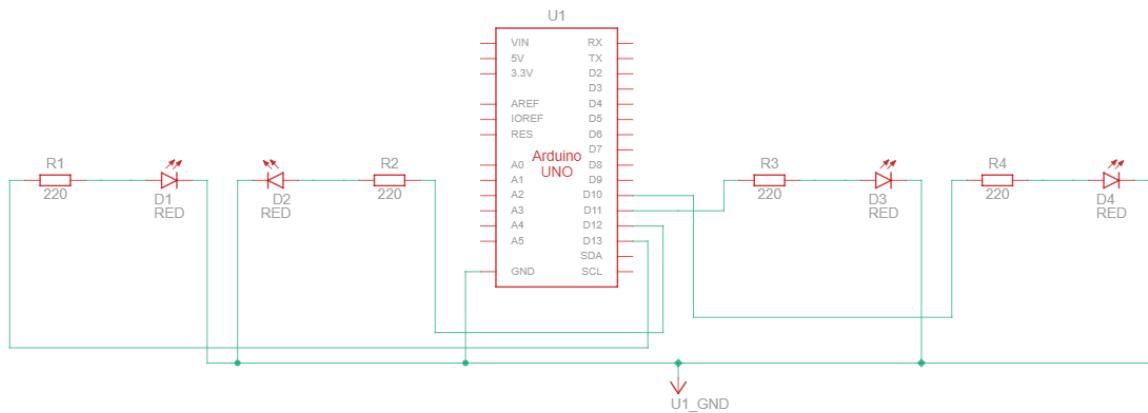
En el primer método (setup) se configuran los pines digitales 13, 12, 11 y 10 como salida, permitiendo enviar señales para encender y apagar los LEDs conectados a dichos pines.

En el segundo método (loop) hemos alternado el encendido y apagado de los LEDs conectados a los pines 13, 12, 11 y 10 con intervalos de tiempo de 150 milisegundos. La lógica que hemos seguido es la siguiente:

1. Encendemos el LED conectado al pin 13 y apagamos el LED conectado al pin 11. Luego, hacemos una espera de 150 milisegundos.
2. Apagamos el LED conectado al pin 12. Luego, hacemos una espera de 150 milisegundos.
3. Encendemos el LED conectado al pin 12 y apagamos el LED en el pin 10. Luego, hacemos una espera de 150 milisegundos.
4. Encendemos el LED conectado al pin 11 y apagamos el LED en el pin 13. Luego, hacemos una espera de 150 milisegundos.
5. Encendemos el LED conectado al pin 10 y apagamos el LED en el pin 12. Luego, hacemos una espera de 150 milisegundos.
6. Apagamos el LED conectado al pin 11. Luego, hacemos una espera de 150 milisegundos.

De esta manera podemos simular las luces que tiene el coche fantástico dado que estamos haciendo una alteración relativamente rápida de los LEDs de manera secuencial.

El esquema con Fritzing es el siguiente:

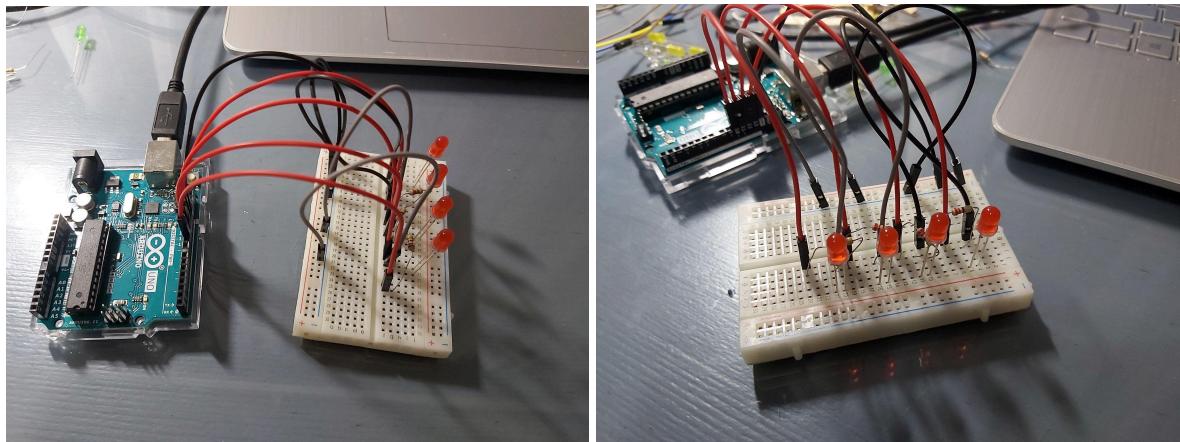


Para ver el proyecto desde Tinkercad, usar el enlace

https://www.tinkercad.com/things/kv7Aj5zTp8b-ejercicio1-opcional?sharecode=m_Bg4edMUl8jgJwLyq2oRv_iPzPC_Bfc08HP3CVM95Y

4.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



En Arduino IDE cargamos el programa creado para hacer el ejercicio. Será el mismo código que se ha usado en el simulador Tinkercad.

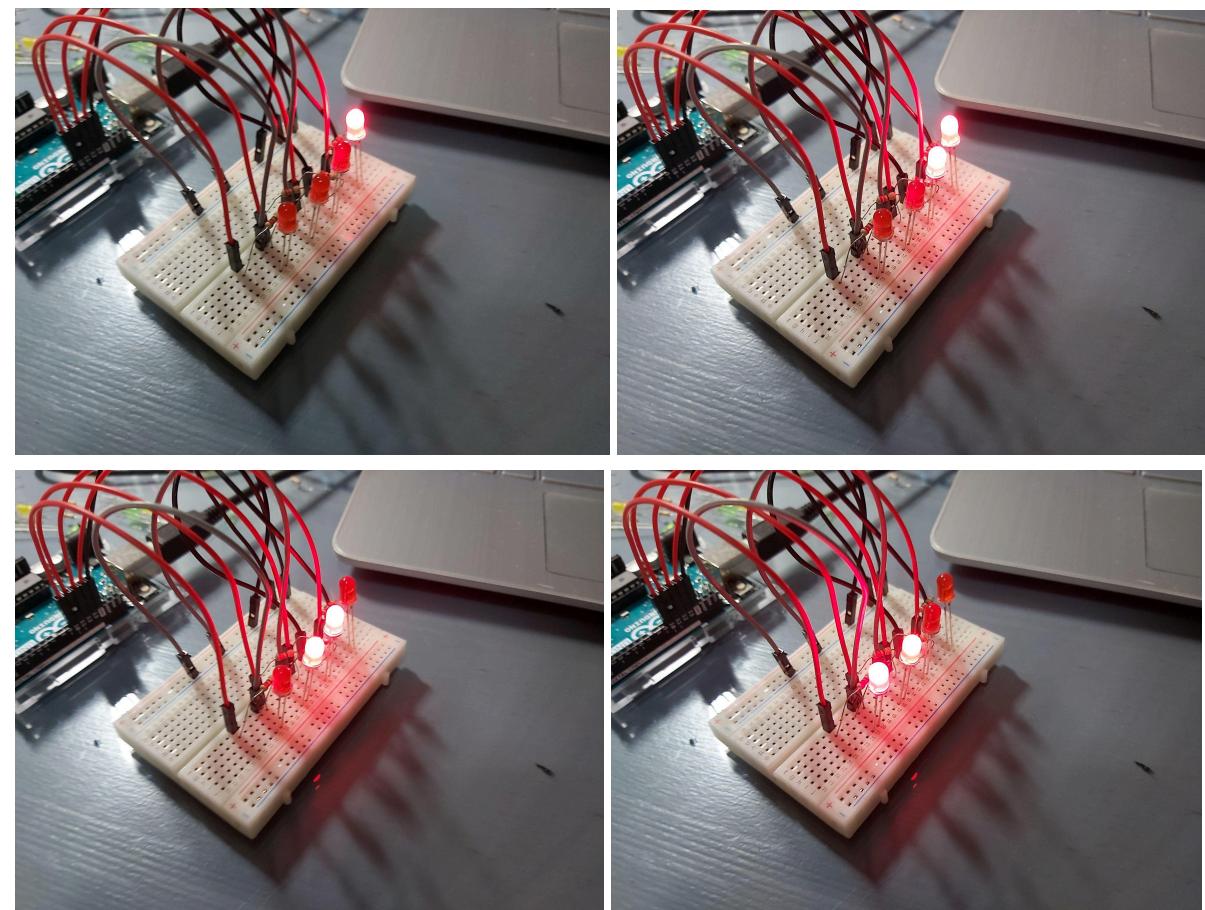
```
void loop()
{
    digitalWrite(13, HIGH);
    digitalWrite(11, LOW);
    delay(150); //Espera de 150 ms

    digitalWrite(12, LOW);
    delay(150); //Espera de 150 ms

    digitalWrite(12, HIGH);
    digitalWrite(10, LOW);
    delay(150); //Espera de 150 ms
    digitalWrite(11, HIGH);
    digitalWrite(13, LOW);
    delay(150); //Espera de 150 ms
    digitalWrite(10, HIGH);
    digitalWrite(12, LOW);
    delay(150); //Espera de 150 ms

    digitalWrite(11, LOW);
    delay(150); //Espera de 150 ms

    digitalWrite(11, HIGH);
    digitalWrite(13, LOW);
    delay(150); //Espera de 150 ms
    digitalWrite(12, HIGH);
    digitalWrite(10, LOW);
    delay(150); //Espera de 150 ms
}
```



También se dispone de un vídeo en la carpeta de github.

5. Ejercicio opcional 2

En este ejercicio se nos pide que usando un buzzer hagamos sonar un pitido en función de la distancia a la que esté un objeto detectado por un sensor de ultrasonidos.

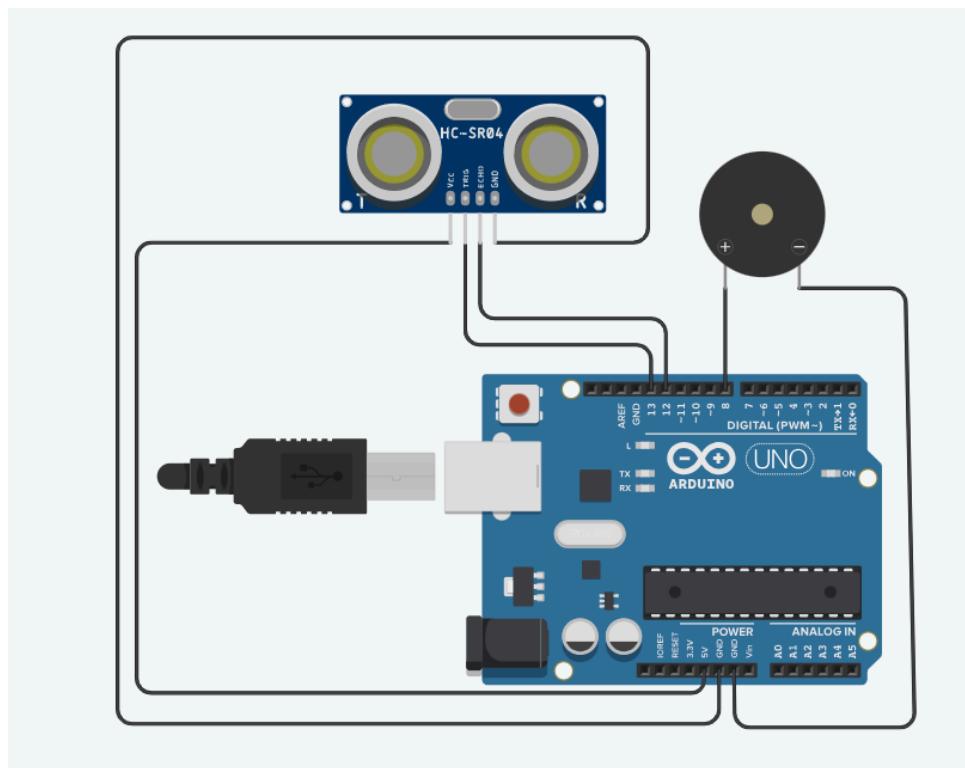
Para hacer este ejercicio tenemos que tener en cuenta los siguientes componentes:

- Sensor de distancia ultrasónico HC-SR04 (4 pines)
- Un piezo (buzzer)
- Arduino uno R3

5.1. Tinkercad

Para la realización del ejercicio en Tinkercad:

El sensor ultrasónico tiene 4 pines: VCC (conectado a 5V en el Arduino), Trig (conectado al pin digital 13 del Arduino), Echo (conectado al pin digital 12 del Arduino), GND (conectado a GND del Arduino). Por otro lado, el piezo pondremos una patilla al pin 8 y otra a GND (tierra).



El código empleado para que el buzzer funcione correctamente, es el siguiente:

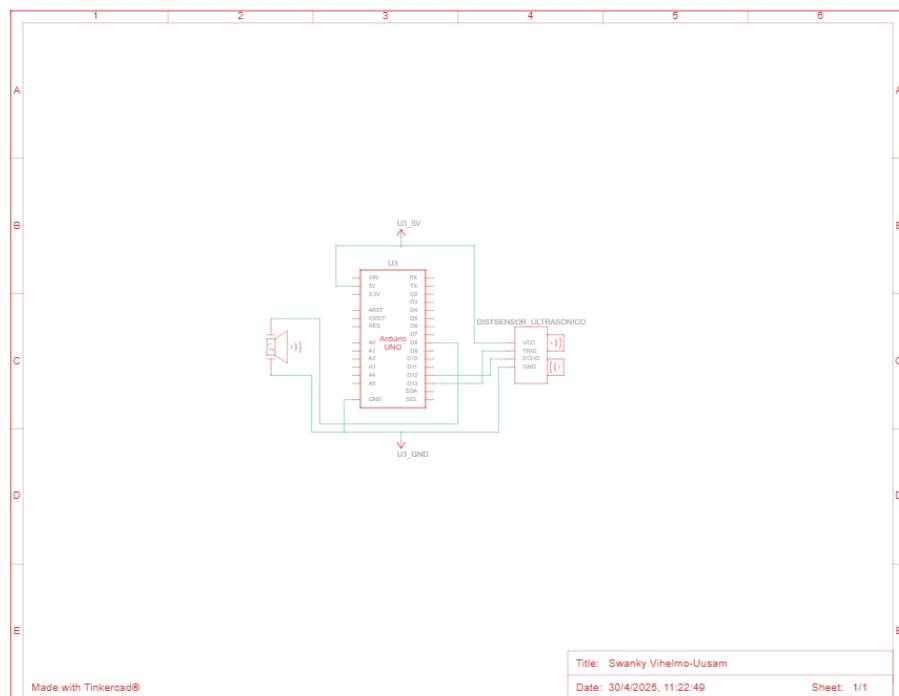
Primero, definimos los pines que vamos a utilizar, el *trigPin* es el pin que se usa para enviar un pulso ultrasónico desde el sensor, el *echoPin* recibe el eco de ese pulso, y el *buzzerPin* está conectado al zumbador, que emitirá un sonido cuando se detecte un objeto cercano.

En la función `setup()`, se inicializan los pines para que el Arduino sepa cuáles son de salida (para enviar señales) y cuáles son de entrada (para recibir datos). También se activa el monitor serial con `Serial.begin(9600)` para poder ver las distancias medidas en la pantalla del ordenador.

En el `loop()`, que se repite constantemente, primero se genera un pulso ultrasónico enviando una señal corta desde el pin `Trig`. Luego, se mide cuánto tiempo tarda el eco en regresar al pin `Echo`. Con ese tiempo, se calcula la distancia al objeto usando una fórmula basada en la velocidad del sonido (0.034 cm/ μ s). Esa distancia se muestra en el monitor serial.

Si la distancia medida es menor a 100 cm, el programa hace sonar el zumbador. La frecuencia del pitido depende de la cercanía del objeto: si está muy cerca, el buzzer pita más rápido; si está más lejos (pero aún dentro del rango), pita más lento. Esto se logra utilizando la función *map()* que traduce la distancia a un tiempo de retardo entre sonidos. Si el objeto está a más de 100cm, el zumbador se apaga completamente usando *noTone()*.

El esquema con Fritzing es el siguiente:

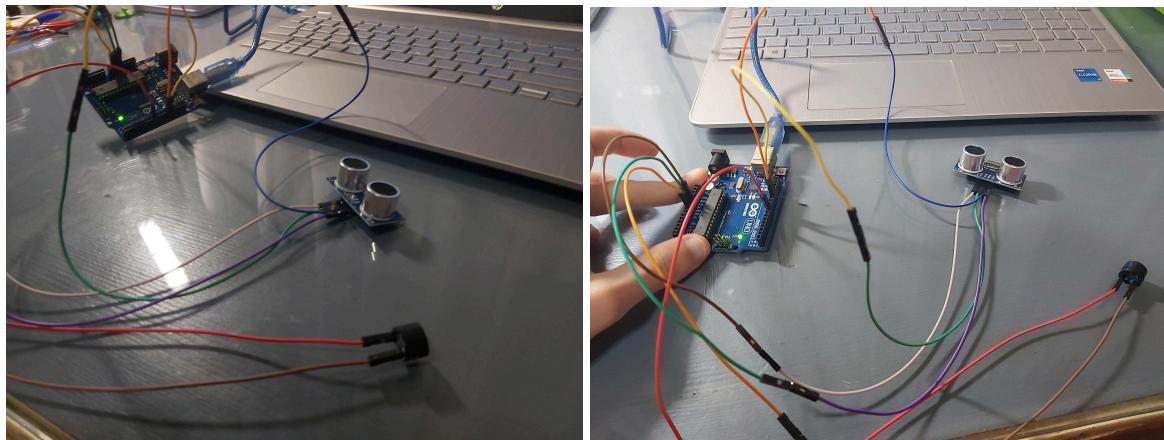


Para ver el proyecto desde Tinkercad usar el siguiente enlace:

<https://www.tinkercad.com/things/0nBF5qVjKzi-swanky-vihelmo-uusam/editel?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashboard&sharecode=NaFab6Nnt0LqE6nTwxNJ85R1bVclyUEwcxDtblZQrEA>

5.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



También se dispone de un vídeo en la carpeta de github.

6. Ejercicio opcional 3

En el tercer ejercicio se nos pide que usemos un LED que se ilumine más o menos en función de la cantidad de luz que se detecte en el fotosensor. Para ello hemos usado los siguientes elementos:

- Arduino Uno R3
- Rojo LED
- Resistencia 10 kΩ
- Resistencia 220 Ω
- Fotoresistencia

6.1. Tinkercad

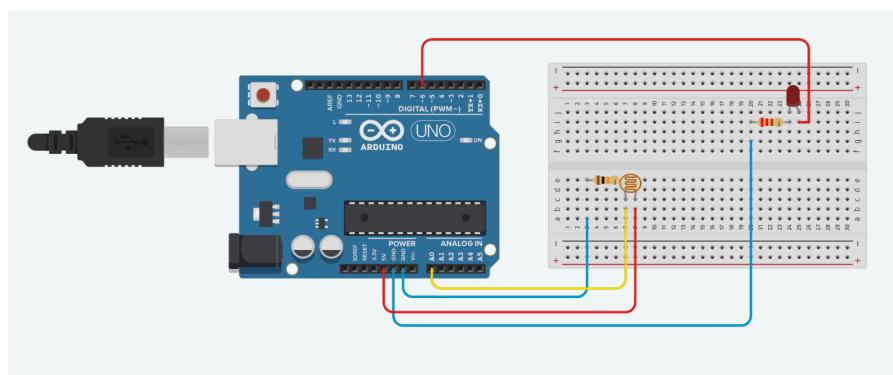
Para la realización del ejercicio en Tinkercad:

LED (con resistencia) colocado en la parte superior derecha:

- Ánodo (+) del LED (pata larga) → conectado a un cable rojo que va al pin -6.
- Cátodo (-) del LED (pata corta) → conectado a una resistencia de 220 Ω
- El otro extremo de la resistencia → conectado a un cable azul que va a tierra.

Fotoresistencia (LDR) colocada parte izquierda del protoboard:

- Un extremo derecho de la LDR → conectado a 5V a través de un cable rojo.
- El otro extremo de la LDR → conectado a un punto central de la protoboard, y también conectado a:
 - Una resistencia hacia GND (10kΩ).
 - Un cable amarillo que va al pin analógico A0 del Arduino.
 - El extremo de la resistencia opuesto a la LDR → va conectado a GND del Arduino.



El código utilizado para su funcionamiento es el siguiente:

```
const int ldrPin = A0;
const int ledPin = 6;

void setup() {
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int ldrValue = analogRead(ldrPin);
    int ledBrightness = map(ldrValue, 0, 1023, 255, 0); // inverso: n
    analogWrite(ledPin, ledBrightness);

    Serial.print("Luz: ");
    Serial.print(ldrValue);
    Serial.print(" | Brillo LED: ");
    Serial.println(ledBrightness);

    delay(200);
}
```

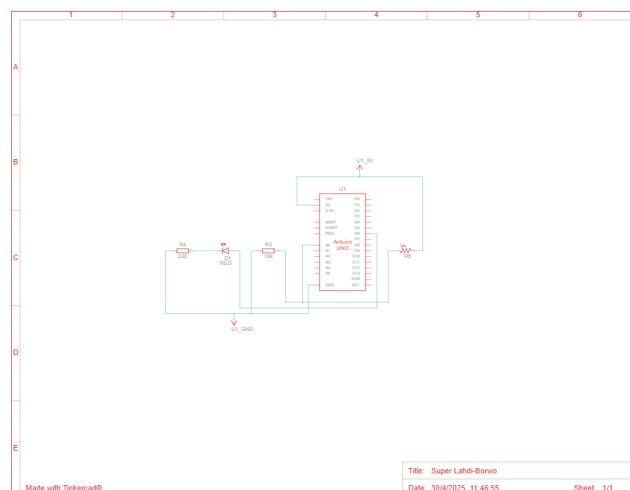
Primero, se definen dos variables: *ldrPin*, que indica el pin donde está conectada la fotoresistencia (A0), y *ledPin*, que corresponde al pin donde está conectado el LED (pin 6).

En la función *setup()*, se configura *ledPin* como salida (OUTPUT), ya que se va a enviar señal a un LED, y se inicia la comunicación con el monitor serial a 9600 baudios para poder visualizar los valores leídos.

Dentro de la función *loop()*, que se repite constantemente, se lee el valor de luz que capta la LDR usando *analogRead(ldrPin)*. Este valor puede ir desde 0 (oscuridad total) hasta 1023 (mucha luz). Después, con la función *map()*, ese valor se convierte en un valor de brillo para el LED, pero de forma inversa: cuando hay más luz, el LED se apaga; y cuando hay menos luz, el LED se enciende más fuerte.

Se usa *analogWrite(ledPin, ledBrightness)* para aplicar ese brillo al LED. También se imprime en el monitor serial el valor de luz y el nivel de brillo del LED para que el usuario pueda observar cómo responde el sistema. Finalmente, hay una pausa de 200 milisegundos (*delay(200)*) antes de repetir el proceso.

El esquema con Fritzing es el siguiente:

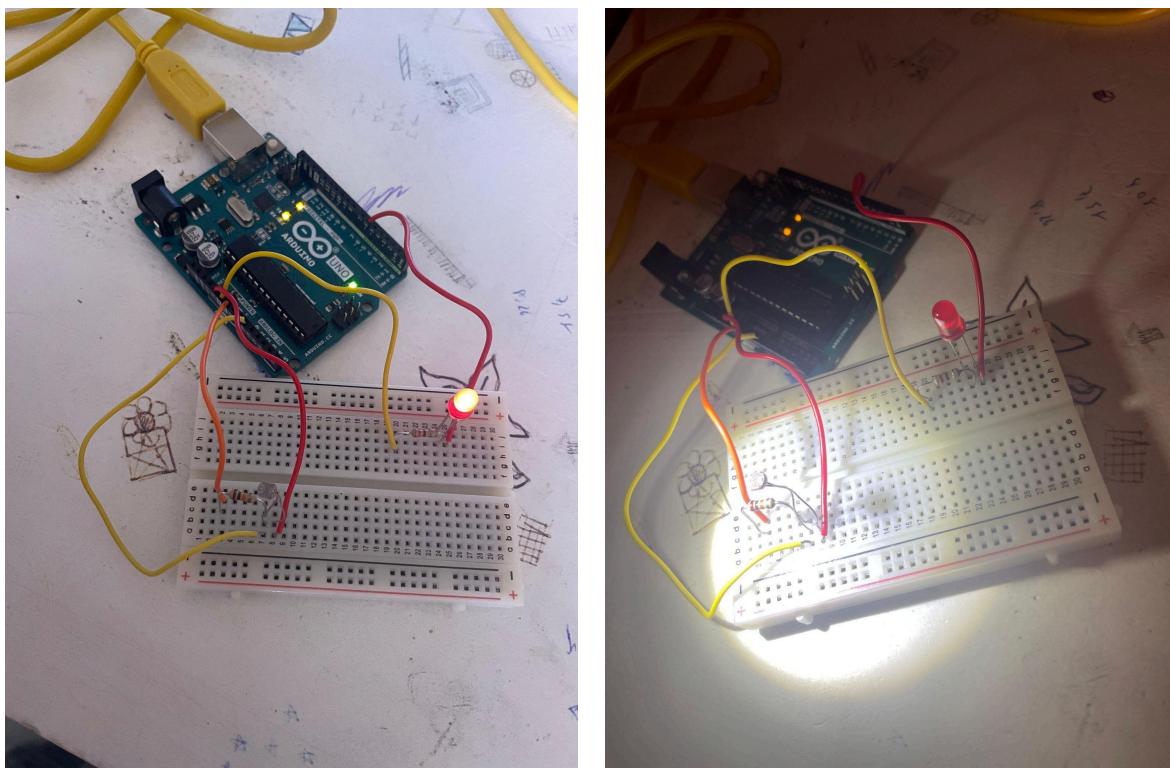


Para ver el proyecto en Tinkercad:

<https://www.tinkercad.com/things/7ckn3irK4Ko-super-lahdi-borwo/edit?returnTo=https%3A%2F%2Fwww.tinkercad.com%2Fdashbaord&sharecode=-EfW3rFbPnb7DiSNhu0bYhueCPFwQMyckrb1nfJWTW8>

6.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



También se dispone de un vídeo en la carpeta de github.

7. Ejercicio opcional 4

En el cuarto ejercicio se nos pide implementar un motor que se activa cuando se pulse un pulsador. Para ello, hemos hecho uso de los siguientes componentes:

- Arduino Uno R3
- Posicional MicroServoMotor
- Pulsador
- Resistencia 10 kΩ

7.1. Tinkercad

Para la realización del ejercicio en Tinkercad:

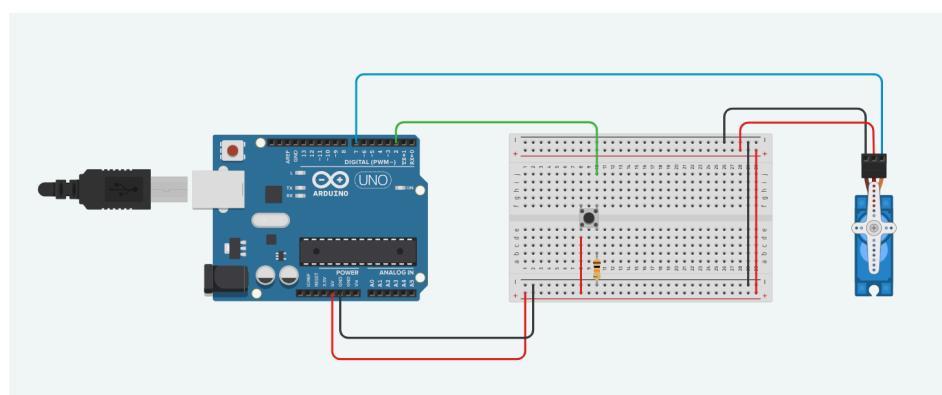
Pulsador

- Un extremo del pulsador está conectado al pin digital 2 del Arduino (cable verde).
- El otro extremo del pulsador:
 - Se conecta a 5V (línea roja de la protoboard).
 - También está conectado a GND a través de una resistencia de 10 kΩ

Cuando el botón no se pulsa, el pin está en LOW (0), y cuando se pulsa, cambia a HIGH (1).

ServoMotor

- Cable rojo (VCC del servo) → conectado a la línea positiva (roja) de la protoboard, que a su vez va al pin 5V del Arduino.
- Cable negro (GND del servo) → conectado a la línea negativa (negra) de la protoboard, que va al GND del Arduino.
- Cable naranja (señal del servo) → conectado al pin digital 7 del Arduino (cable azul).



El código utilizado para el funcionamiento del circuito es el siguiente:

```
#include <Servo.h>

Servo servo;

const int buttonPin = 2;      // Botón conectado al pin 2
const int servoPin = 7;       // Servo conectado al pin 7

bool lastButtonState = LOW;
bool currentButtonState;
bool servoAt90 = false;

void setup() {
    pinMode(buttonPin, INPUT);      // Usa resistencia externa (pul
    servo.attach(servoPin);
    servo.write(0);                // Posición inicial
    Serial.begin(9600);
}

void loop() {
    currentButtonState = digitalRead(buttonPin);

    // Detecta flanco de subida (cuando se presiona el botón)
    if (currentButtonState == HIGH && lastButtonState == LOW) {
        if (!servoAt90) {
            servo.write(90);        // Va a 90°
        }
        else {
            servo.write(0);        // Vuelve a 0°
            servoAt90 = false;
        }
        delay(200);   // Antirrebote
    }

    lastButtonState = currentButtonState;
}
```

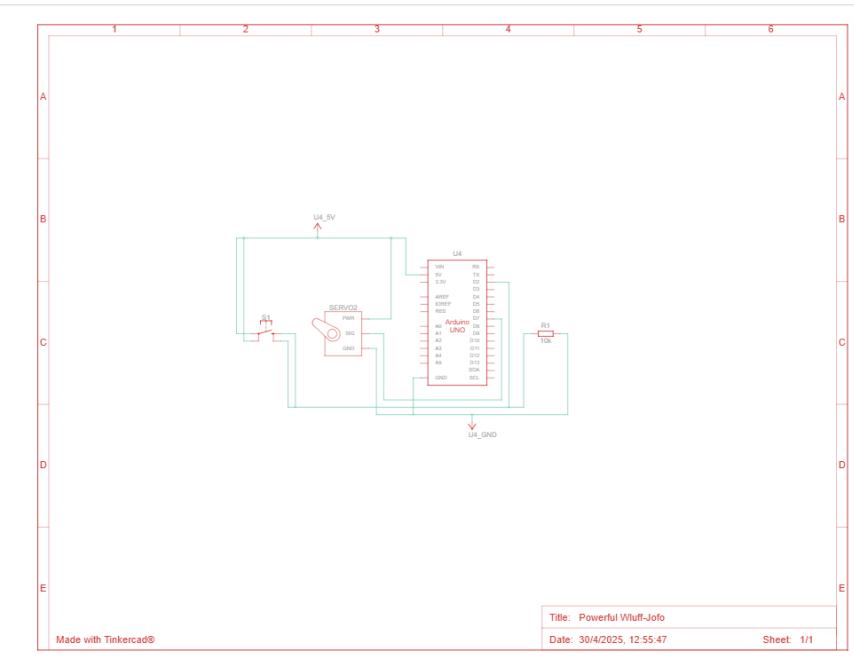
El botón está conectado al pin digital 2 y el servo al pin digital 7.

Se declara también una variable booleana llamada `servoAt90` para llevar el control de si el servo está o no en la posición de 90°. En la función `setup()`, se configura el botón como entrada, se conecta el servo al pin correspondiente con `servo.attach()`, y se inicializa su posición en 0 grados.

En el `loop()`, se lee continuamente el estado actual del botón con `digitalRead(buttonPin)`, y se compara con el estado anterior. Si el estado anterior era *LOW* (*no pulsado*) y el nuevo es *HIGH* (*pulsado*), entonces se detecta una pulsación. En ese momento se cambia la posición del servo: si estaba en 0°, se mueve a 90°, y si ya estaba en 90°, se vuelve a 0°.

Este cambio se controla usando la variable `servoAt90`, que se actualiza en cada pulsación. También se añade una pausa corta de 200 ms para evitar efectos de rebote.

El esquema con Fritzing es el siguiente:

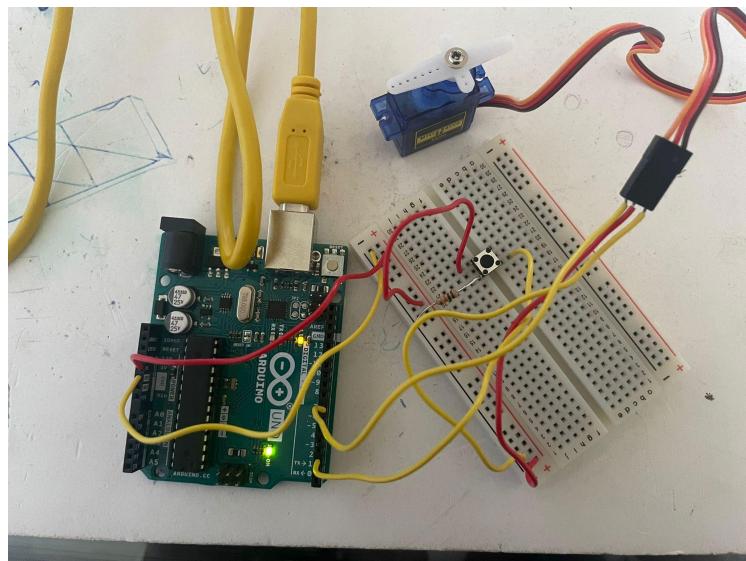


Para ver el proyecto en Tinkercad:

<https://www.tinkercad.com/things/irgY3PdlFpx/editel?returnTo=%2Fdashboard&sharecode=VLdMGGGurcwzzTRqUkLkb-qhR64KXSs6KqLlgLTE2ziQ>

7.2. Arduino IDE

El código que se usa para el Arduino es el mismo que el de Tinkercad:



También se dispone de un vídeo en la carpeta de github.

8. Ejercicios adicionales

8.1. Sensor de temperatura (ejercicio 1 adicional)

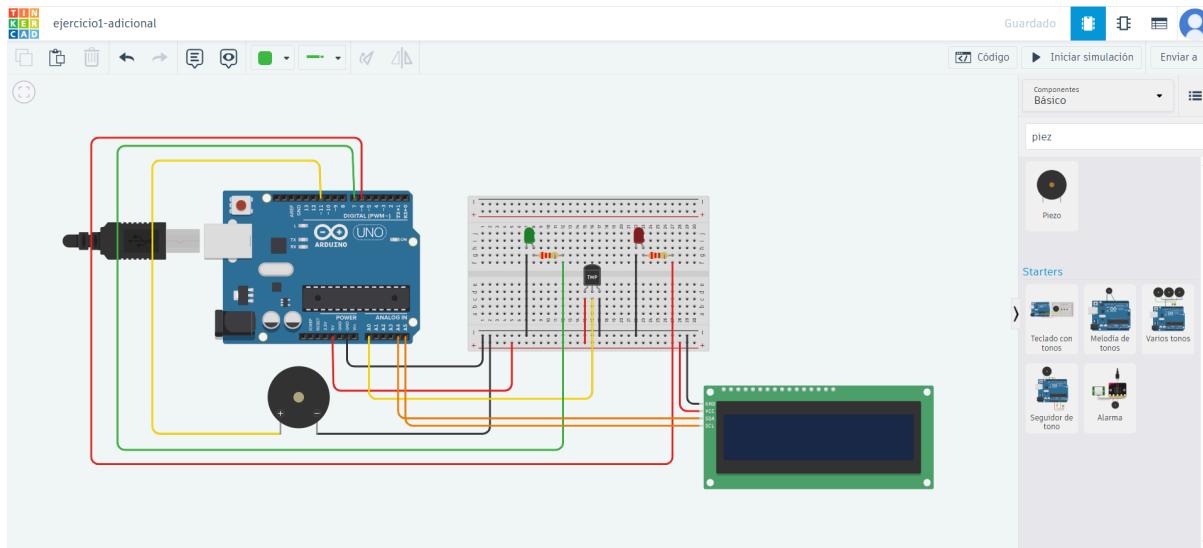
En este ejercicio vamos a realizar un programa en donde se tenga en cuenta la temperatura actual. Lo que hará será que, a partir de la temperatura que haya en cada momento, se indicará en un LCD 16x2 (I2C) en donde, si hace mucho calor (temperatura ≥ 30) o si hace mucho frío (temperatura < 10), sonará una alarma, indicando en el lcd que hace mucho calor o frío. Además, se creará el esquema con Fritzing y se cargará el programa en Arduino IDE para comprobar que funciona correctamente.

Para hacer este ejercicio debemos de tener los siguientes componentes:

- Arduino UNO R3
- 2 resistencias de 220Ω
- 1 LED rojo
- 1 LED verde
- 15 cables conectores
- 1 placa de pruebas
- 1 LCD 16x2 (I2C)
- 1 sensor de temperatura
- 1 piezo

8.1.1. Tinkercad

Se ha cogido un *Arduino UNO R3* y una placa de pruebas para llevar a cabo esta tarea. Los LEDs están conectados a esta placa en donde los cátodos están conectados a tierra y los ánodos a las salidas digitales 7 y 6. Se ha empleado una resistencia de 220Ω en cada ánodo de los LED para evitar sobretensiones en los diodos LED. El sensor de temperatura se ha metido en el pin analógico A0 mientras que el piezo se encuentra en el pin digital 11, poniendo sus correspondientes cables a tierra (en el caso del sensor de temperatura, también necesita un voltaje el cual pondremos que sea de 5V). El LCD se ha conectado sus correspondientes claves de tierra y potencia, además de poner el SDA al pin analógico A4 y el SCL al pin analógico A5. El esquema es el siguiente:



El código para el correcto funcionamiento del programa es el siguiente:

```

1 #include <Adafruit_LiquidCrystal.h>
2
3 const int pinPiezo = 11;
4 const int ledVerde = 7;
5 const int ledRojo = 6;
6 const int pinTemp = A0;
7
8 Adafruit_LiquidCrystal lcd_1(0);
9
10 void setup() {
11   lcd_1.begin(16, 2);
12   lcd_1.setCursor(0, 0);
13
14   pinMode(pinPiezo, OUTPUT);
15   pinMode(ledVerde, OUTPUT);
16   pinMode(ledRojo, OUTPUT);
17
18   noTone(pinPiezo);
19   digitalWrite(ledVerde, LOW);
20   digitalWrite(ledRojo, LOW);
21
22   Serial.begin(9600);
23 }
24
25 void loop() {
26   int sensorValue = analogRead(pinTemp);
27   float voltage = sensorValue * (5.0 / 1023.0);
28   float temperatura = (voltage - 0.5) * 100.0;
29
30   lcd_1.setCursor(0, 0);
31
32   lcd_1.print("T:");
33   lcd_1.print(temperatura, 1);
34
35   if (temperatura < 10.0) {
36     lcd_1.setCursor(0, 1);
37     lcd_1.print("HACE FRIO      ");
38     tone(pinPiezo, 1000);
39     digitalWrite(ledVerde, LOW);
40     digitalWrite(ledRojo, HIGH);
41   }
42   else if (temperatura < 30.0) {
43     lcd_1.setCursor(0, 1);
44     lcd_1.print("TEMP NORMAL    ");
45     noTone(pinPiezo);
46     digitalWrite(ledVerde, HIGH);
47     digitalWrite(ledRojo, LOW);
48   }
49   else {
50     lcd_1.setCursor(0, 1);
51     lcd_1.print("HACE CALOR      ");
52     tone(pinPiezo, 1000);
53     digitalWrite(ledVerde, LOW);
54     digitalWrite(ledRojo, HIGH);
55   }
56
57   //Depuración que hacemos en Serial
58   Serial.print("Temperatura: ");
59   Serial.print(temperatura, 1);
60
61   delay(500);
}

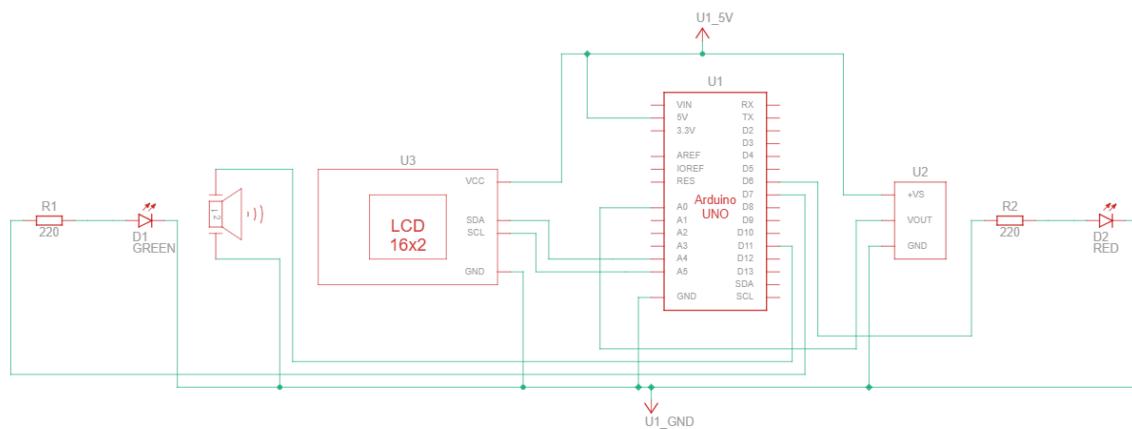
```

En el primer método (setup) configuramos el pin 11 como salida para poder emitir señales sonoras usando el piezo. Los pines 7 y 6 también los configuramos como salida (LEDs) lo cual nos permitirá controlar el encendido y apagado de los mismos. Además, iniciamos la pantalla del LCD con dimensiones de 16 columnas y 2 filas donde configuramos el cursor en la posición inicial (esta pantalla se usará para mostrar la temperatura y los mensajes correspondientes a esta). Finalmente, desactivamos el tono inicial del zumbador y apagamos los LEDs para que comience en estado LOW e iniciamos la comunicación serial para depuración (lo usamos para imprimir la temperatura y verificar los valores que registre el sensor).

En el segundo método (loop) lo que hacemos es hacer una lectura desde el sensor de temperatura conectado al pin A0, donde tenemos en cuenta el cálculo de la temperatura en grados Celsius y lo mostramos por la pantalla LCD. Luego, dependiendo de la temperatura

que haga, se mostrará el mensaje correspondiente por el LCD indicando si *HACE FRÍO* (temperatura < 10), si la temperatura es “normal” (*TEMP NORMAL*), o si *HACE CALOR* (temperatura ≥ 30). En el caso de que la temperatura sea mayor o igual a 30°C o menor a 10°C, el piezo empezará a sonar con un tono de 1000 Hz. Además, si la temperatura es “normal”, el LED verde estará encendido, mientras que en caso de que si el sensor de temperatura indica que hace mucho frío o calor, se apagará el LED verde y se encenderá el rojo. Finalmente, esperamos 500 milisegundos para repetir el ciclo de manera indefinida.

El esquema con Fritzing es el siguiente:

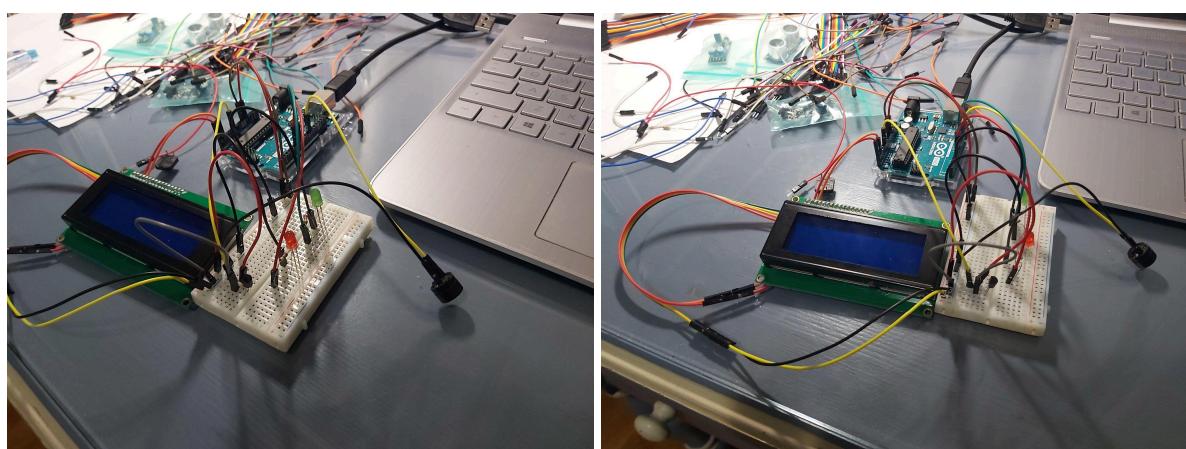


Para ver el proyecto desde Tinkercad, usar el enlace

<https://www.tinkercad.com/things/ep33luwe0aE-ejercicio1-adicional?sharecode=4pY0V7pKNlpFRLjJzmWyA651CKSCLNSqAbrQotV81c8>

8.1.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



En Arduino IDE cargamos el programa creado para hacer el ejercicio. El código ha sido modificado para configurar la dirección del LCD 16x2 (I2C) y, para que en el video se pudiera ver su funcionamiento, se ha modificado para que se pueda ver los cambios de temperatura de manera que se pueda apreciar los mensajes del LCD.

```
ejercicio1-adicional Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda

ejercicio1-adicional §
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int pinPiezo = 11;
const int ledVerde = 7;
const int ledRojo = 6;
const int pinTemp = A0;
const float VREF = 5.0;
LiquidCrystal_I2C lcd_1(0x27, 16, 2);
const int numReadings = 20;
int readings[numReadings];
int readIndex = 0;
long total = 0;
float average = 0;

void setup() {
  Wire.begin();
  lcd_1.init();
  lcd_1.backlight();

  pinMode(pinPiezo, OUTPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(ledRojo, OUTPUT);

  noTone(pinPiezo);
  digitalWrite(ledVerde, LOW);
  digitalWrite(ledRojo, LOW);

  Serial.begin(9600);
}

for (int i = 0; i < numReadings; i++) {
  readings[i] = 0;
}

void loop() {
  total = total - readings[readIndex];
  readings[readIndex] = analogRead(pinTemp);
  total = total + readings[readIndex];
  readIndex++;
  if (readIndex >= numReadings) {
    readIndex = 0;
  }
  average = total / (float) numReadings;

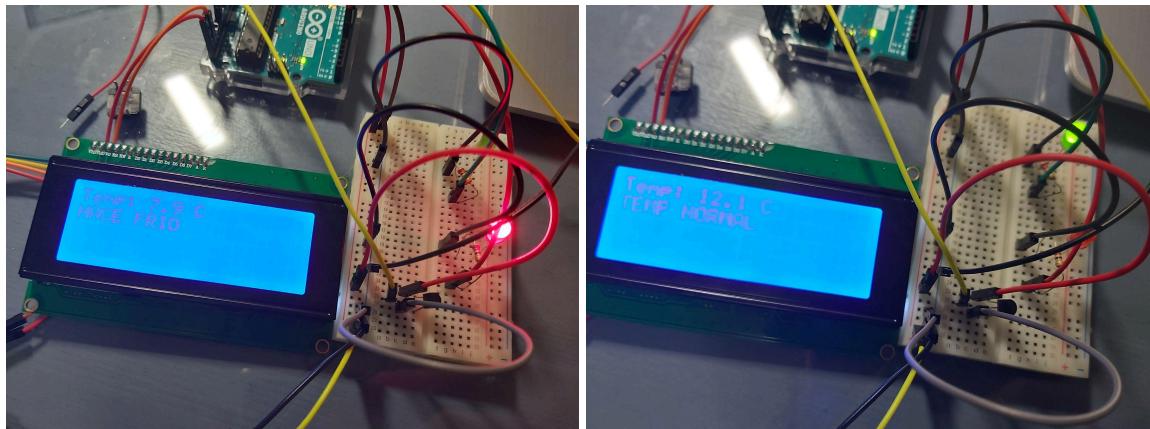
  float voltage = average * (VREF / 1023.0);
  float temperatura = (voltage - 0.5) * 100.0;

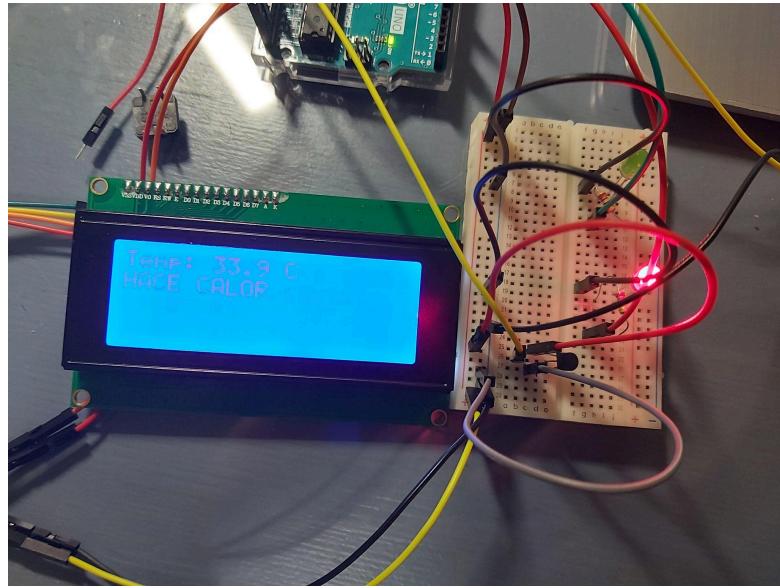
  Serial.print("Raw avg: ");
  Serial.print(average);
  Serial.print(" Voltaje: ");
  Serial.print(voltage, 3);
  Serial.print(" V Temperatura: ");
  Serial.print(temperatura, 1);
  Serial.println(" °C");

  lcd_1.clear();
  lcd_1.setCursor(0, 0);
  lcd_1.print("Temp: ");
  lcd_1.print(temperatura, 1);
  lcd_1.print(" °C");

  if (temperatura < 10.0) {
    lcd_1.setCursor(0, 1);
    lcd_1.print("HACE FRIO      ");
    tone(pinPiezo, 1000);
    digitalWrite(ledVerde, LOW);
    digitalWrite(ledRojo, HIGH);
  }
  else if (temperatura < 30.0) {
    lcd_1.setCursor(0, 1);
    lcd_1.print("TEMP NORMAL   ");
    noTone(pinPiezo);
    digitalWrite(ledVerde, HIGH);
    digitalWrite(ledRojo, LOW);
  }
  else {
    lcd_1.setCursor(0, 1);
    lcd_1.print("HACE CALOR     ");
    tone(pinPiezo, 1000);
    digitalWrite(ledVerde, LOW);
    digitalWrite(ledRojo, HIGH);
  }

  delay(1000);
}
```





También se dispone de un vídeo en la carpeta de github.

8.2. Juego de reflejos (ejercicio 2 adicional)

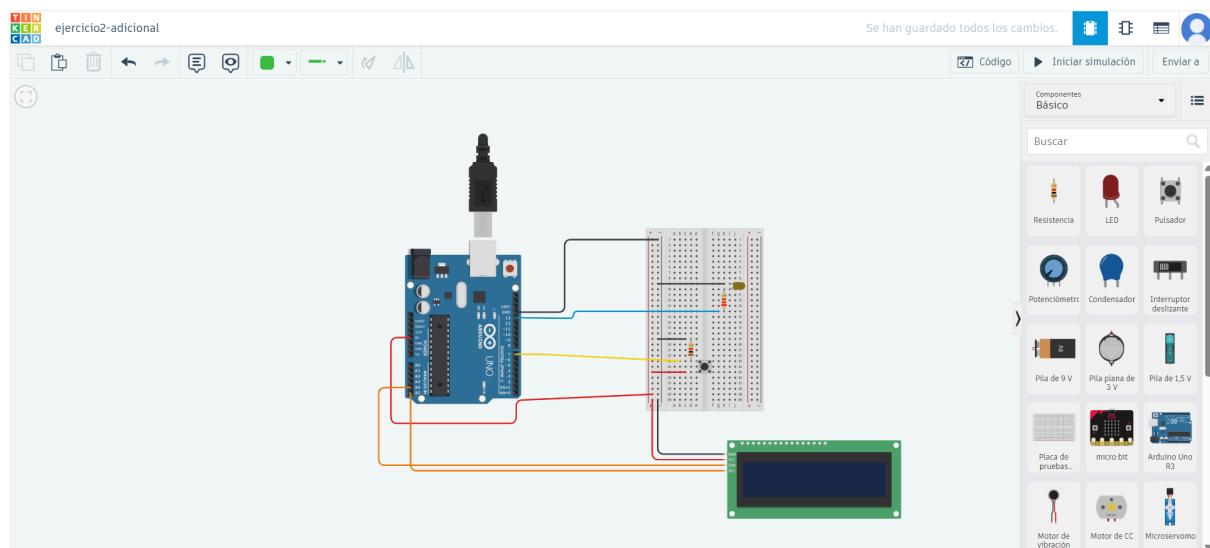
Para este ejercicio se ha hecho un programa en el cual se simula un juego de reflejos. En el tendremos que pulsar el pulsador para iniciar dicho juego y en una pantalla LCD 16x2 (I2C) saldrá el tiempo que tardamos en presionar otra vez el pulsador una vez se haya encendido el LED amarillo. El juego consiste en, una vez se encienda el LED amarillo, tarda el menor tiempo posible en pulsar el pulsador. Además, en el LCD saldrá el tiempo en milisegundos que hemos tardado en presionar dicho pulsador desde que se encendió el LED.

Para hacer este ejercicio debemos de tener los siguientes componentes:

- Arduino UNO R3
- 2 resistencias de 220Ω
- 1 LED amarillo
- 11 cables conectores
- 1 placa de pruebas
- 1 pulsador
- 1 LCD 16x2 (I2C)

8.2.1. Tinkercad

Se ha cogido un *Arduino UNO R3* y una placa de pruebas para llevar a cabo esta tarea. El LED está conectado a esta placa en donde el cátodo está conectado a tierra y el ánodo a la salida digital 13. Se ha empleado una resistencia de 220Ω en el ánodo del LED para evitar sobretensiones en los diodos LED. Además, se ha conectado un pulsador a la placa de pruebas en donde una de las patillas la tiene conectada a tierra y al pin digital 7, mientras que a la otra se le ha conectado un voltaje de 5V. El esquema es el siguiente:



El código empleado para la realización de este juego ha sido el siguiente:

```

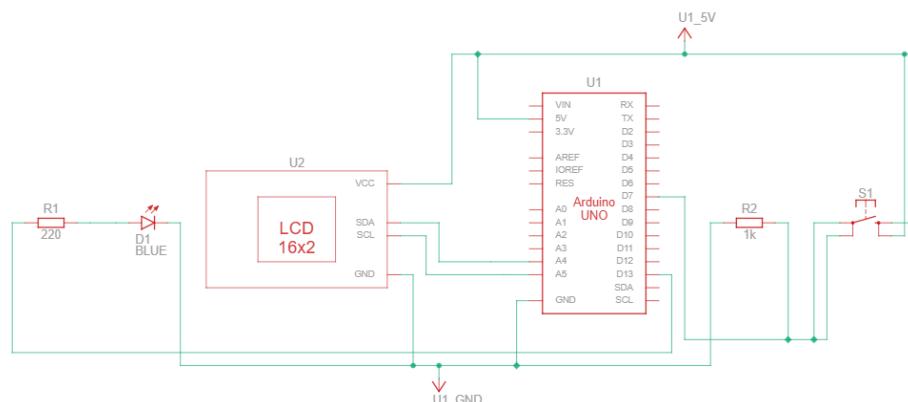
1 #include <Adafruit_LiquidCrystal.h>
2
3 Adafruit_LiquidCrystal lcd_1(0);
4
5 const int pinLed = 13;
6 const int pinBoton = 7;
7
8 const unsigned long retardoRebote = 50;
9
10 unsigned long tiempoInicio = 0;
11 unsigned long tiempoReaccion = 0;
12
13 void setup() {
14   pinMode(pinLed, OUTPUT);
15   pinMode(pinBoton, INPUT_PULLUP);
16
17   lcd_1.begin(16, 2);
18
19   lcd_1.clear();
20   lcd_1.setCursor(0, 0);
21   lcd_1.print("Juego Reaccion");
22   lcd_1.setCursor(0, 1);
23   lcd_1.print("Presiona boton");
24   delay(2000);
25   lcd_1.clear();
26   lcd_1.setCursor(0, 0);
27   lcd_1.print("Presiona boton");
28 }
29
30 void loop() {
31   if (digitalRead(pinBoton) == LOW) {
32     while (digitalRead(pinBoton) == LOW) { }
33     delay(retardoRebote);
34
35     lcd_1.clear();
36     lcd_1.setCursor(0, 0);
37     lcd_1.print("Preparado..."); 
38     lcd_1.setCursor(0, 1);
39     lcd_1.print("Esperando LED");
40
41     delay(1000);
42
43     int retardoAleatorio = random(2000, 5000);
44     delay(retardoAleatorio);
45
46     digitalWrite(pinLed, HIGH);
47     lcd_1.clear();
48     lcd_1.setCursor(0, 0);
49     lcd_1.print("YA!");
50
51     tiempoInicio = millis();
52
53     while(digitalRead(pinBoton) == LOW) { }
54     delay(retardoRebote);
55
56     while (digitalRead(pinBoton) != LOW) { }
57     delay(retardoRebote);
58
59     tiempoReaccion = millis() - tiempoInicio;
60
61     digitalWrite(pinLed, LOW);
62
63     lcd_1.clear();
64     lcd_1.setCursor(0, 0);
65     lcd_1.print("Tiempo:");
66     lcd_1.setCursor(0, 1);
67     lcd_1.print(tiempoReaccion);
68     lcd_1.print(" ms");
69
70     delay(2000);
71
72     lcd_1.clear();
73     lcd_1.setCursor(0, 0);
74     lcd_1.print("Presiona boton");
75
76     while (digitalRead(pinBoton) == LOW) { }
77     delay(retardoRebote);
78   }
79 }
80

```

En el primer método (setup) configuramos el pin 13 como salida permitiéndonos enviar señales para encender y apagar el LED conectado a ese pin. Por otro lado, configuramos el pin digital 7 como entrada para permitir leer el estado del pulsador conectado a dicho pin. Además, iniciamos el LCD para que funcione como pantalla 16x2 caracteres y mostramos un mensaje inicial del juego seguido del mensaje que indica al jugador que presione el botón para iniciar el juego.

En el segundo método (loop) controlamos el estado del pin digital 7 para implementar el juego de reacción. En el caso de que se detecte que el botón está presionado, el sistema espera hasta que se suelte el botón y se hace un pequeño retardo para evitar el rebote. Esto lo hacemos para calcular bien el tiempo que tarda entre que se enciende la luz y lo que tarda el jugador en pulsar el botón. Además, mostramos los mensajes en la pantalla indicando que el jugador esté preparado para cuando se encienda el LED pulsar el botón. Este se encenderá de manera aleatoria entre 2-5 segundos y, una vez que se encienda, el LCD escribe por pantalla un mensaje indicando que ya ha empezado el juego. El jugador deberá de pulsar el botón lo más rápido posible mientras el programa espera esa pulsación. Una vez que el jugador presione el pulsador, el LED se apagará y se le indicará por el LCD el tiempo en milisegundos que ha tardado en pulsar el botón. Finalmente, se vuelve a reiniciar el juego por si el jugador desea jugar otra partida.

El esquema con Fritzing es el siguiente:

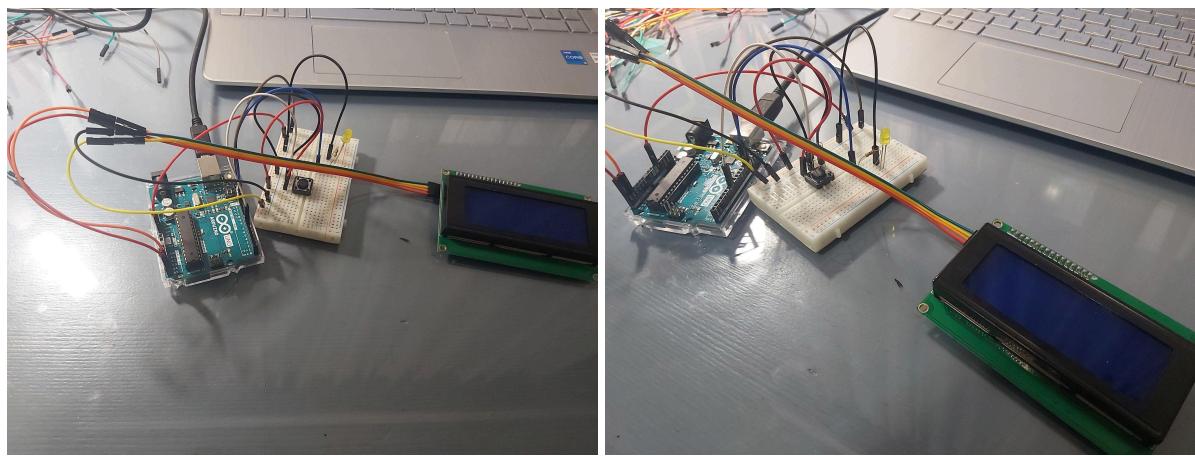


Para ver el proyecto desde Tinkercad, usar el enlace

<https://www.tinkercad.com/things/6cNmliHWdhr-ejercicio2-adicional?sharecode=esSE2iqhZFkFsbbbrmpbFql4UxbDU0QoeLXvhql0HgNw>

8.2.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



En Arduino IDE cargamos el programa creado para hacer el ejercicio. Se ha tenido que adaptar el código realizado en Tinkercad dado a la configuración de la dirección del LCD 16x2 (I2C).

```
ejercicio2-adicional Arduino 1.8.19 (Windows Store 1.8.57.0)
Archivo Editar Programa Herramientas Ayuda
ejercicio2-adicional
void loop() {
  if (digitalRead(pinBoton) == LOW) {
    while (digitalRead(pinBoton) == LOW) { }
    delay(retardoRebote);
    lcd_1.clear();
    lcd_1.setCursor(0, 0);
    lcd_1.print("Preparando...");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Esperando LED");
    delay(1000);

    unsigned long tiempoInicio = 0;
    unsigned long tiempoReaccion = 0;

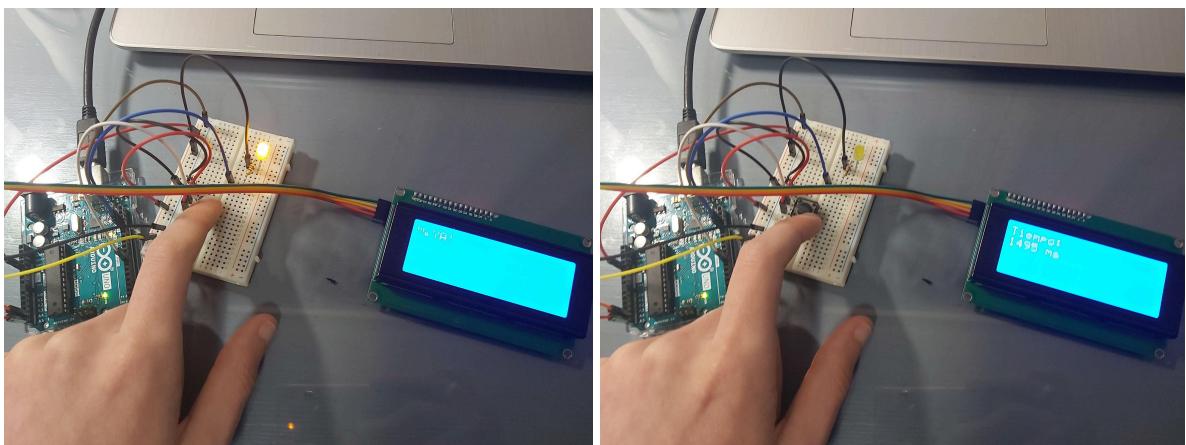
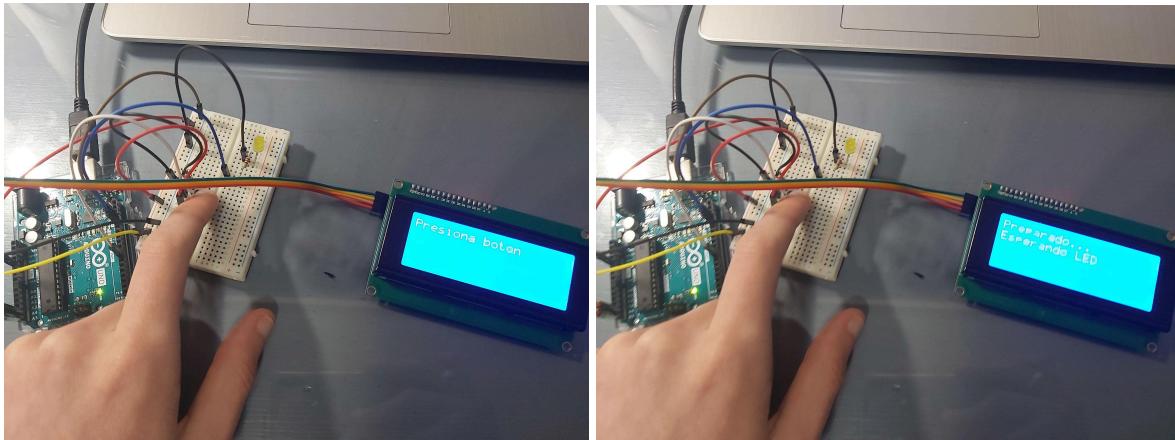
    void setup() {
      Wire.begin();
      pinMode(pinLed, OUTPUT);
      pinMode(pinBoton, INPUT_PULLUP);

      lcd_1.init(); // Inicializa el LCD
      lcd_1.backlight(); // Enciende la retroiluminación
    }

    lcd_1.clear();
    lcd_1.setCursor(0, 0);
    lcd_1.print("Juego Reaccion");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Presiona boton");
    delay(2000);
    lcd_1.clear();
    lcd_1.setCursor(0, 0);
    lcd_1.print("Presiona boton");
  }
}

void loop() {
  if (digitalRead(pinBoton) == LOW) {
    while (digitalRead(pinBoton) == LOW) { }
    delay(retardoRebote);
    lcd_1.clear();
    lcd_1.setCursor(0, 0);
    lcd_1.print("Tiempo:");
    lcd_1.setCursor(0, 1);
    lcd_1.print(tiempoReaccion);
    lcd_1.print(" ms");
    delay(2000);

    lcd_1.clear();
    lcd_1.setCursor(0, 0);
    lcd_1.print("Presiona boton");
  }
}
```



También se dispone de un vídeo en la carpeta de github.

8.3. Hola mundo en código morse (ejercicio 3 adicional)

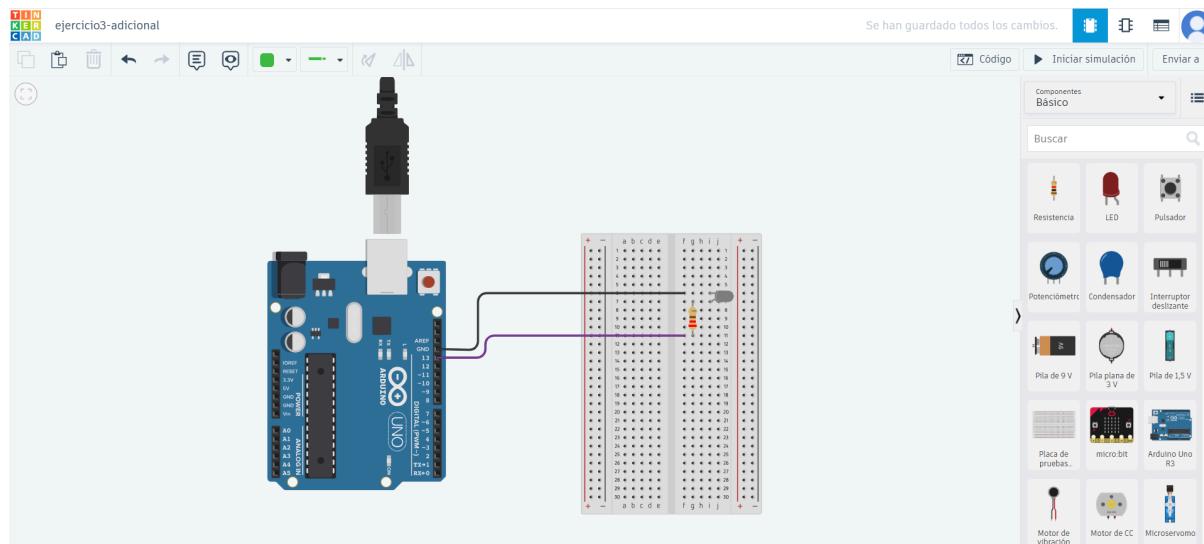
En este ejercicio vamos a implementar el programa que transmite en código morse el mensaje de “Hola mundo” mediante un LED para que encienda y apague alternativamente (uno blanco), conectado a la salida digital 13 del Arduino. Además, se creará el esquema con Fritzing y se cargará el programa en Arduino IDE para comprobar que funciona correctamente.

Para hacer este ejercicio debemos de tener los siguientes componentes:

- Arduino UNO R3
- 1 resistencia de 220Ω
- 1 LED blanco
- 2 cables conectores
- 1 placa de pruebas

8.3.1. Tinkercad

Se ha cogido un *Arduino UNO R3* y una placa de pruebas para llevar a cabo esta tarea. El LED está conectado a esta placa en donde el cátodo está conectado a tierra y el ánodo a la salida digital 13. Se ha empleado una resistencia de 220Ω en el ánodo del LED para evitar sobretensiones en los diodos LED. El esquema es el siguiente:



El código empleado para transmitir el mensaje ha sido el siguiente:

```
1 void setup() {
2   pinMode(13, OUTPUT);
3 }
4
5 void loop() {
6   //Letra H (****)
7   for (int i = 0; i < 4; i++) {
8     digitalWrite(13, HIGH);
9     delay(200);
10    digitalWrite(13, LOW);
11    delay(200);
12  }
13  delay(600);
14
15 //Letra O (---)
16 for (int i = 0; i < 3; i++) {
17   digitalWrite(13, HIGH);
18   delay(600);
19   digitalWrite(13, LOW);
20   delay(200);
21 }
22 delay(600);
23
24 //Letra L (-**)
25 digitalWrite(13, HIGH);
26 delay(200);
27 digitalWrite(13, LOW);
28 delay(200);
29 digitalWrite(13, HIGH);
30 delay(600);
31 digitalWrite(13, LOW);
32 delay(200);
33 for (int i = 0; i < 2; i++) {
34   digitalWrite(13, HIGH);
35   delay(200);
36   digitalWrite(13, LOW);
37   delay(200);
38 }
39 delay(600);
40
41 //Letra A (*-)
42 digitalWrite(13, HIGH);
43 delay(200);
44 digitalWrite(13, LOW);
45 delay(200);
46 digitalWrite(13, HIGH);
47 delay(600);
48 digitalWrite(13, LOW);
49 delay(600);
50
51 //Dejamos un espacio entre palabras
52 delay(1400);
53
54 //Letra M (--)
55 for (int i = 0; i < 2; i++) {
56   digitalWrite(13, HIGH);
57   delay(600);
58   digitalWrite(13, LOW);
59   delay(200);
60 }
61 delay(600);
62
63 //Letra U (**-)
64 for (int i = 0; i < 2; i++) {
65   digitalWrite(13, HIGH);
66   delay(200);
67   digitalWrite(13, LOW);
68   delay(200);
69 }
70 digitalWrite(13, HIGH);
71 delay(600);
72 digitalWrite(13, LOW);
73 delay(600);
74
75 //Letra N (-*)
76 digitalWrite(13, HIGH);
77 delay(600);
78 digitalWrite(13, LOW);
79 delay(200);
80 digitalWrite(13, HIGH);
81 delay(200);
82 digitalWrite(13, LOW);
83 delay(600);
84
85 //Letra D (-**)
86 digitalWrite(13, HIGH);
87 delay(600);
88 digitalWrite(13, LOW);
89 delay(200);
90 for (int i = 0; i < 2; i++) {
91   digitalWrite(13, HIGH);
92   delay(200);
93   digitalWrite(13, LOW);
94   delay(200);
95 }
96 delay(600);
97
98 //Letra O (---)
99 for (int i = 0; i < 3; i++) {
100   digitalWrite(13, HIGH);
101   delay(600);
102   digitalWrite(13, LOW);
103   delay(200);
104 }
105 delay(5000);
106
107 }
108 }
```

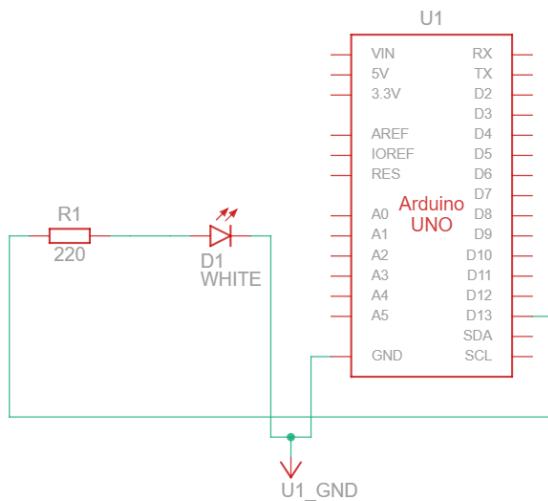
En el primer método (setup) lo que hacemos es configurar el pin digital 13 como salida, lo que nos permite poder enviar señales para encender y apagar el LED.

En el segundo método (loop) se hará el programa en donde se diga “Hola mundo” en código morse. Para ello, nos hemos basado en el lenguaje morse.

A ● -	J ● ---	S ● ● ●
B - ● ● ●	K - ● -	T -
C - ● - ●	L ● - ● ●	U ● ● -
D - ● ●	M --	V ● ● ● -
E ●	N - ●	W ● - -
F ● ● - ●	O ---	X - ● ● -
G - - ●	P ● - - ●	Y - ● - -
H ● ● ● ●	Q - - - ● -	Z - - - ● ●
I @ ●	R ● - -	

Lo que se ha hecho es que el LED parpadee más rápido cuando son puntos (200 milisegundos) que cuando son rayas (600 milisegundos). Además, se deja un mensaje entre cada letra (600 milisegundos) y cada palabra (1400 milisegundos). Finalmente, se deja una espera de 5 segundos para luego volver a repetir el mensaje.

El esquema con Fritzing es el siguiente:

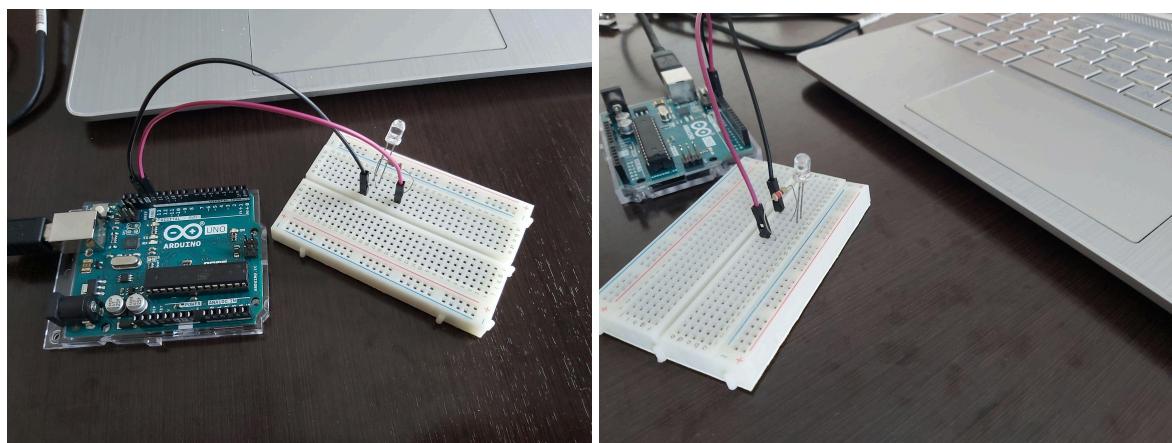


Para ver el proyecto desde Tinkercad, usar el enlace

https://www.tinkercad.com/things/bYzaulsOteT-ejercicio3-adicional?sharecode=eZ6_fuN_7rHkb6K3_xLnLtdYZVG9Or_gacWLf92LL8M

8.3.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



En Arduino IDE cargamos el programa creado para hacer el ejercicio. Será el mismo código que se ha usado en el simulador Tinkercad.

```
ejercicio3-adicional Arduino 1.8.19 (Windows Store 1.8.57.0) — ejercicio3-adicional
Archivo Editar Programa Herramientas Ayuda
ejercicio3-adicional
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  //Letra H (****)
  for (int i = 0; i < 4; i++) {
    digitalWrite(13, HIGH);
    delay(200);
    digitalWrite(13, LOW);
    delay(200);
  }
  delay(600);

  //Letra O (---)
  for (int i = 0; i < 3; i++) {
    digitalWrite(13, HIGH);
    delay(600);
    digitalWrite(13, LOW);
    delay(200);
  }
  delay(600);

  //Letra L (●-●)
  digitalWrite(13, HIGH);
  delay(200);
  digitalWrite(13, LOW);
  delay(200);
  digitalWrite(13, HIGH);
  delay(600);
  digitalWrite(13, LOW);
  delay(200);
  for (int i = 0; i < 2; i++) {
    digitalWrite(13, HIGH);
    delay(200);
    digitalWrite(13, LOW);
    delay(200);
  }
  delay(600);

  //Letra U (●●-)
  for (int i = 0; i < 2; i++) {
    digitalWrite(13, HIGH);
    delay(200);
    digitalWrite(13, LOW);
    delay(200);
  }
  delay(600);

  //Letra O (---)
  for (int i = 0; i < 3; i++) {
    digitalWrite(13, HIGH);
    delay(600);
    digitalWrite(13, LOW);
    delay(200);
  }
  delay(5000);
}

delay(200);
digitalWrite(13, LOW);
delay(200);
digitalWrite(13, HIGH);
delay(600);
digitalWrite(13, LOW);
delay(600);

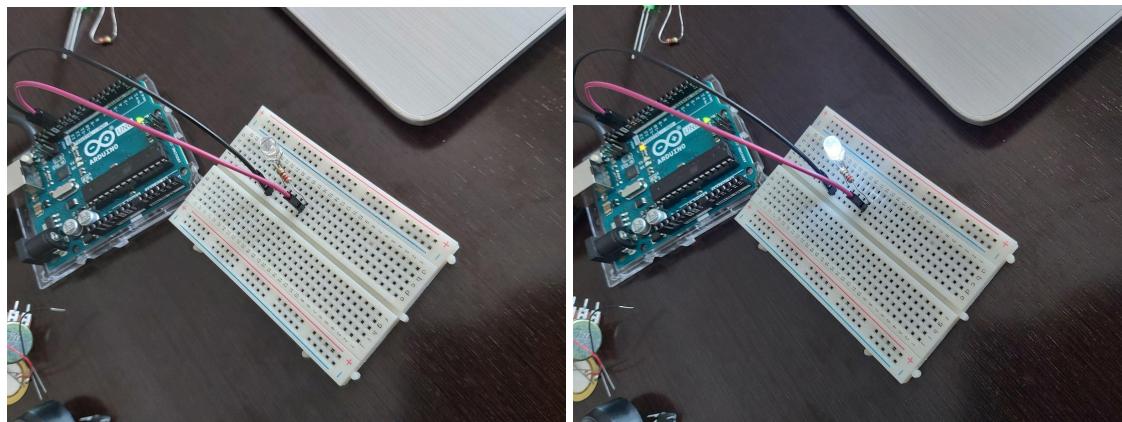
//Letra A (●-)
digitalWrite(13, HIGH);
delay(200);
digitalWrite(13, LOW);
delay(200);
digitalWrite(13, HIGH);
delay(600);
digitalWrite(13, LOW);
delay(600);

//Letra N (-●)
digitalWrite(13, HIGH);
delay(600);
digitalWrite(13, LOW);
delay(200);
digitalWrite(13, HIGH);
delay(200);
digitalWrite(13, LOW);
delay(600);

//Letra D (-●●)
digitalWrite(13, HIGH);
delay(600);
digitalWrite(13, LOW);
delay(200);
for (int i = 0; i < 2; i++) {
  digitalWrite(13, HIGH);
  delay(200);
  digitalWrite(13, LOW);
  delay(200);
}
delay(600);

}

delay(200);
digitalWrite(13, LOW);
delay(200);
digitalWrite(13, HIGH);
delay(600);
digitalWrite(13, LOW);
delay(600);
```



También se dispone de un vídeo en la carpeta de github.

8.4. Termómetro de colores (ejercicio 4 adicional)

Este ejercicio consiste en hacer un termómetro con tres colores, usando un sensor de temperatura y 3 LEDs (rojo, verde, azul) y se iluminan según el valor de la temperatura.

Se necesitan los siguientes componentes:

- Arduino Uno R3
- Sensor de temperatura [TMP36]
- LED Azul
- LED Rojo
- LED Verde
- Tres resistencias de $220\ \Omega$

8.4.1. Tinkercad

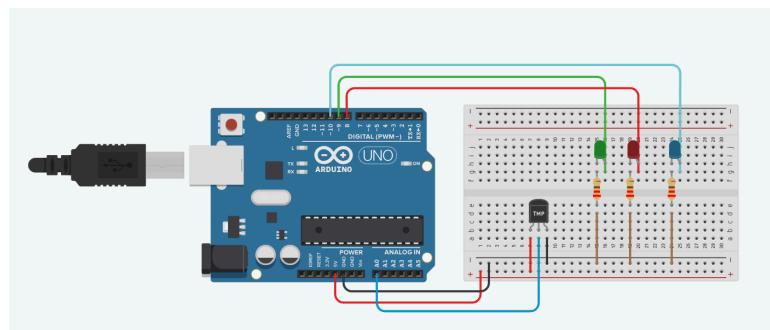
Para la realización del ejercicio en Tinkercad:

Sensor de temperatura (TMP36):

- Pata izquierda (VCC) → 5V del Arduino
- Pata del medio (salida) → pin A0 del Arduino
- Pata derecha (GND) → GND del Arduino

LEDs:

- LED rojo (indica calor):
 - Ánode (+) → pin 8 del Arduino (con resistencia de $220\ \Omega$)
 - Cátodo (-) → GND
- LED verde (temperatura normal):
 - Ánode → pin 9
 - Cátodo → GND
- LED azul (frío):
 - Ánode → pin 10
 - Cátodo → GND



El código que hace funcionar el circuito es:

```
const int tempPin = A0;      // Sensor de temperatura conectado a
const int ledRojo = 8;        // LED rojo
const int ledVerde = 9;        // LED verde
const int ledAzul = 10;       // LED azul

void setup() {
    pinMode(ledRojo, OUTPUT);
    pinMode(ledVerde, OUTPUT);
    pinMode(ledAzul, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int lectura = analogRead(tempPin);
    float voltaje = lectura * 5.0 / 1023.0;
    float temperaturaC = (voltaje - 0.5) * 100; |

    Serial.print("Temperatura: ");
    Serial.print(temperaturaC);
    Serial.println(" °C");

21   // Apagamos todos los LEDs al inicio
22   digitalWrite(ledRojo, LOW);
23   digitalWrite(ledVerde, LOW);
24   digitalWrite(ledAzul, LOW);
25
26   // Encendemos según el rango de temperatura
27   if (temperaturaC < 18) {
28       digitalWrite(ledAzul, HIGH); // Hace frío
29   } else if (temperaturaC >= 18 && temperaturaC < 25) {
30       digitalWrite(ledVerde, HIGH); // Temperatura normal
31   } else {
32       digitalWrite(ledRojo, HIGH); // Hace calor
33   }
34
35   delay(500);
36 }
37
38 }
```

En la función `setup()`, se configuran los tres pines donde están conectados los LEDs (8, 9 y 10) como salidas. Además, se inicia el monitor serial con `Serial.begin(9600)` para poder ver en pantalla los valores de temperatura que el Arduino está midiendo en tiempo real.

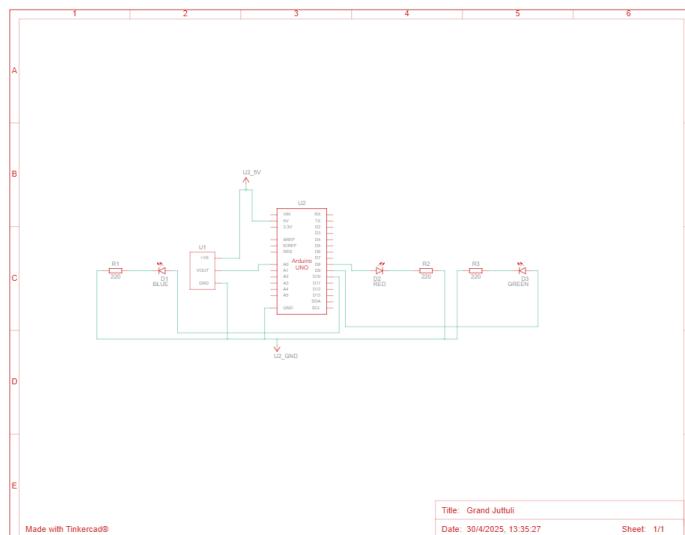
Dentro de la función `loop()`, se empieza leyendo el valor del sensor con `analogRead(tempPin)`. Este valor analógico varía entre 0 y 1023 y representa un voltaje entre 0V y 5V. Por eso, se convierte primero a voltaje usando la fórmula: $voltaje = lectura * 5.0 / 1023.0$.

Después, ese voltaje se convierte en temperatura en grados Celsius con la fórmula $temperaturaC = (voltaje - 0.5) * 100$. Esto funciona para el sensor *TMP36*, que da 0.5V cuando la temperatura es 0°C, y cada aumento de 0.01V representa un aumento de 1°C.

Una vez obtenida la temperatura, el programa apaga los tres LEDs con `digitalWrite(..., LOW)` para evitar que se queden encendidos de una lectura anterior. Luego, según la temperatura:

- Si es menor a 18°C, se enciende el LED azul (hace frío).
- Si está entre 18°C y 25°C, se enciende el LED verde (temperatura normal).
- Si es mayor a 25°C, se enciende el LED rojo (hace calor).

El esquema con Fritzing es el siguiente:

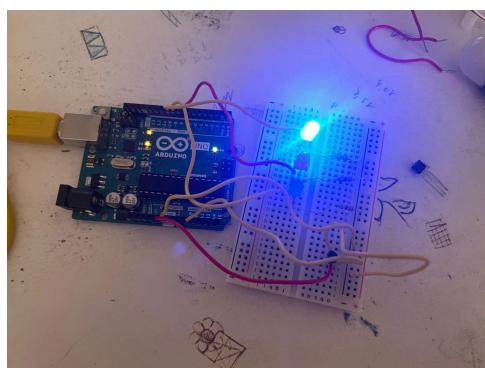


Para ver el proyecto desde Tinkercad, usar el enlace:

<https://www.tinkercad.com/things/3HMhygmHpJ5/edit#design>

8.4.2. Arduino IDE

Ahora vamos a hacer el ejercicio con componentes reales de Arduino. Lo primero de todo es realizar el ejercicio con una placa de pruebas y con un *Arduino UNO R3*, montándolo con los componentes correspondientes. El ejercicio montado quedaría de la siguiente manera:



9. Bibliografía

https://swad.ugr.es/swad/tmp/wt/5oIWX801QnFNWGIldAk_OOqrsJTYEbXA_st7RBYN8/P3-Arduino.pdf