PRÁCTICA DE ARDUINO - SEMAFORO

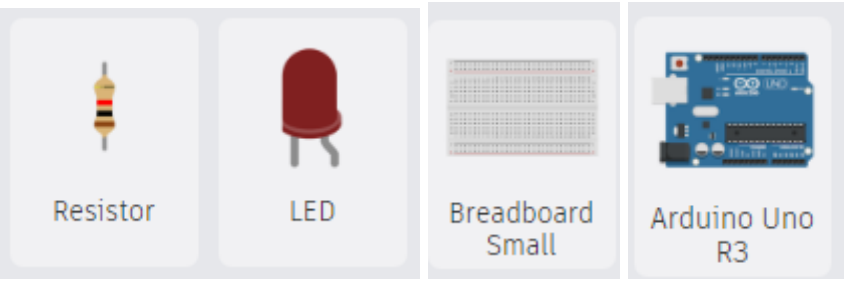
NIVEL: 1
DURACIÓN: 2 horas
OBJETIVO: Los estudiantes aprenderán a simular un semáforo utilizando tres LEDs (rojo, amarillo y verde), controlados por temporizadores a través de una placa Arduino.
MATERIALES

- Placa Arduino Uno (o similar)
- 1 LED rojo
- 1 LED amarillo
- 1 LED verde
- 3 resistencias de 220 ohmios
- Cables de conexión (jumpers)
- Protoboard
- Computadora con el software Arduino IDE instalado
- ThinkerCad

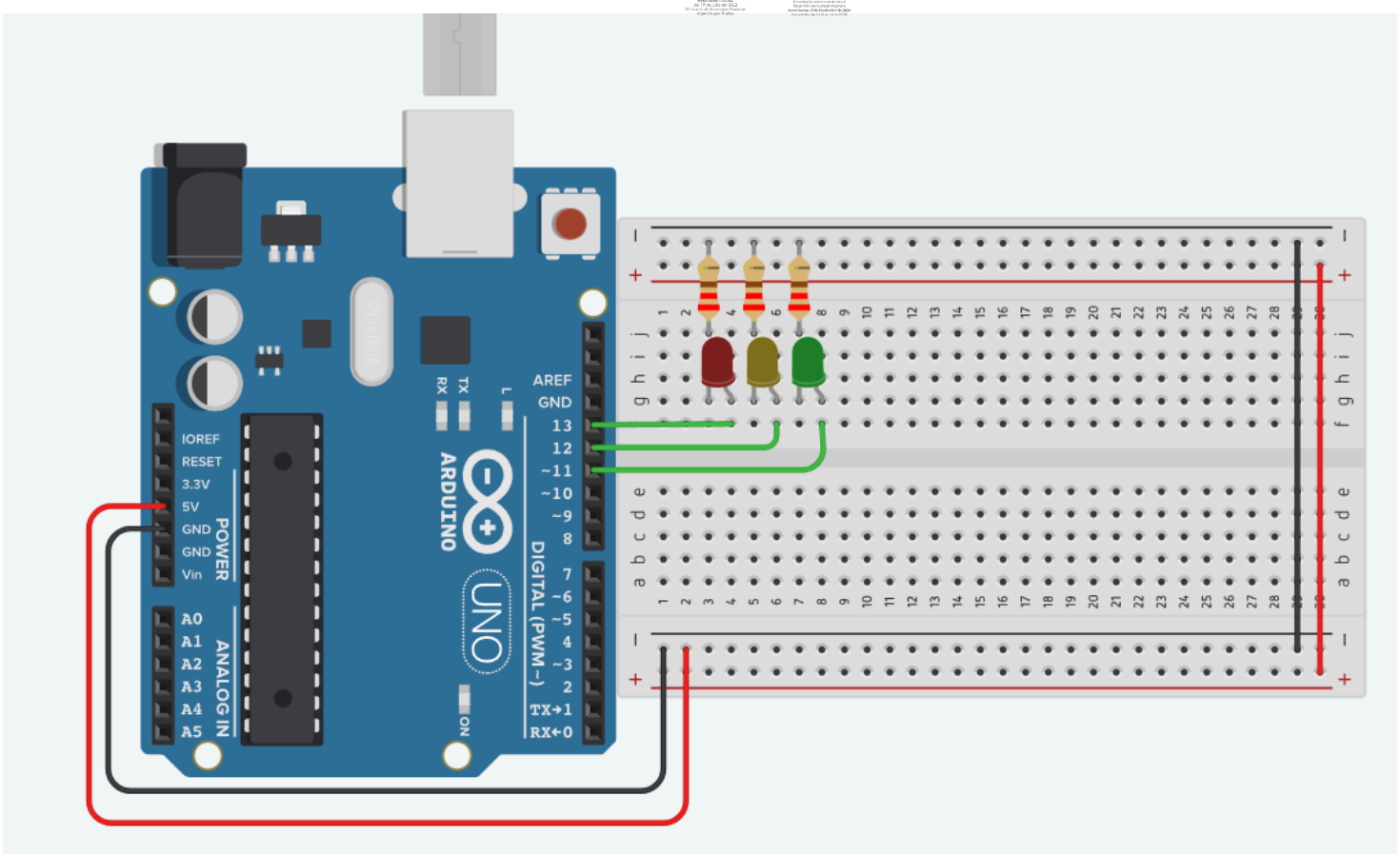
1. DESARROLLO DE LA PRÁCTICA

1.1 SIMULACIÓN EN THINKERCAD

1.1.1 COMPONENTES



1.1.2 MONTAJE



2.2 PROGRAMACIÓN EN ARDUINO IDE

```ino
// Definir los pines de los LEDs
const int redLED = 13;     // Pin del LED rojo
const int yellowLED = 12;  // Pin del LED amarillo
const int greenLED = 11;   // Pin del LED verde

void setup() {
  // Configurar los pines de los LEDs como salidas
  pinMode(redLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
}

void loop() {
  // Encender el LED rojo (Semáforo en rojo)
  digitalWrite(redLED, HIGH);
  delay(5000);  // Mantener la luz roja durante 5 segundos
  digitalWrite(redLED, LOW);

    // Encender el LED amarillo (Semáforo en amarillo)
  digitalWrite(yellowLED, HIGH);
  delay(2000);  // Mantener la luz amarilla durante 2 segundos
  digitalWrite(yellowLED, LOW);

  // Encender el LED verde (Semáforo en verde)
  digitalWrite(greenLED, HIGH);
  delay(5000);  // Mantener la luz verde durante 5 segundos
  digitalWrite(greenLED, LOW);

  // Encender el LED amarillo (Semáforo en amarillo)
  digitalWrite(yellowLED, HIGH);
  delay(2000);  // Mantener la luz amarilla durante 2 segundos
  digitalWrite(yellowLED, LOW);
}
```

Explicación:

- Cada LED está configurado como una salida digital.
- El ciclo de funcionamiento del semáforo se controla mediante temporizadores (`delay()`), que encienden y apagan los LEDs con tiempos específicos:
  - El LED rojo se enciende durante 5 segundos.
  - El LED verde se enciende durante 5 segundos.

- El LED amarillo se enciende durante 2 segundos.

- El ciclo se repite indefinidamente, simulando el funcionamiento continuo de un semáforo.

2. ANÁLISIS DE RESULTADOS

¿Por qué utilizamos resistencias en serie con los LEDs?

- Las resistencias limitan la corriente que pasa a través de los LEDs para evitar que se quemen. Sin ellas, los LEDs podrían recibir demasiada corriente, lo que dañaría los componentes.

¿Cómo afecta el valor del `delay()` al comportamiento del semáforo?

- El valor de `delay()` determina cuánto tiempo permanece encendida cada luz. Cambiar este valor cambiará el tiempo en que el semáforo se mantiene en rojo, verde o amarillo. Por ejemplo, si se aumenta el `delay()` del LED rojo a 10,000 milisegundos, la luz roja se mantendrá encendida por 10 segundos en lugar de 5.

¿Qué sucedería si conectamos un LED sin resistencia?

- Si conectamos un LED directamente sin resistencia, hay un alto riesgo de que el LED se queme debido a la alta corriente, ya que no hay ningún componente que limite el paso de corriente a través del LED.

¿Qué ocurre si no usamos temporizadores (`delay()`)?

- Si no usamos temporizadores, los LEDs cambiarían de estado (encendido y apagado) de manera extremadamente rápida, lo que haría imposible observar el ciclo del semáforo. Los temporizadores permiten que los LEDs se mantengan encendidos por un tiempo definido.

¿Cómo modificarías el código para cambiar el tiempo del semáforo?

3. Simplemente cambiando los valores de `delay()`. Por ejemplo, si quiero que el LED verde permanezca encendido durante 10 segundos, cambiaría `delay(5000)` por `delay(10000)` en la sección del LED verde.

4. CONCLUSIONES

a. Los estudiantes han aprendido los principios básicos de temporización y control de salidas digitales con Arduino.

b. La simulación del semáforo es una aplicación realista de cómo se pueden controlar varios componentes (en este caso, LEDs) utilizando temporizadores, una funcionalidad fundamental en proyectos de electrónica y automatización.
c. Se ha mostrado la importancia de los retardos (delay()) en la programación de sistemas de control para asegurar que los estados de las señales cambien de manera visible y controlada.

5. ANEXO: SEMAFORO USANDO LA FUNCIÓN `millis()`

```
// Definir los pines de los LEDs
const int redLED = 13;
const int yellowLED = 12;
const int greenLED = 11;

// Variables para gestionar el tiempo
unsigned long previousMillis = 0;
const long redInterval = 5000;    // 5 segundos para el LED rojo
const long greenInterval = 5000;  // 5 segundos para el LED verde
const long yellowInterval = 2000; // 2 segundos para el LED amarillo

// Variable para rastrear el estado del semáforo
int ledState = 0;  // 0 = rojo, 1 = verde, 2 = amarillo

void setup() {
  // Configurar los pines de los LEDs como salidas
  pinMode(redLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
}
```

```
void loop() {
  // Obtener el tiempo actual
  unsigned long currentMillis = millis();

  // Cambio de estado del semáforo basado en el tiempo transcurrido
  switch (ledState) {
    case 0:  // Estado LED verde
      if (currentMillis - previousMillis >= greenInterval) {
        previousMillis = currentMillis;  // Reiniciar el tiempo
        digitalWrite(greenLED, LOW);     // Apagar el LED verde
        digitalWrite(yellowLED, HIGH);   // Encender el LED amarillo
        ledState = 1;  // Cambiar al siguiente estado (amarillo)
      }
      break;

    case 1:  // Estado LED amarillo (después de verde)
      if (currentMillis - previousMillis >= yellowInterval) {
        previousMillis = currentMillis;  // Reiniciar el tiempo
        digitalWrite(yellowLED, LOW);    // Apagar el LED amarillo
        digitalWrite(redLED, HIGH);      // Encender el LED rojo
        ledState = 2;  // Cambiar al siguiente estado (rojo)
      }
      break;

    case 2:  // Estado LED rojo
      if (currentMillis - previousMillis >= redInterval) {
        previousMillis = currentMillis;  // Reiniciar el tiempo
        digitalWrite(redLED, LOW);       // Apagar el LED rojo
        digitalWrite(yellowLED, HIGH);   // Encender el LED amarillo (precaución)
        ledState = 3;  // Cambiar al siguiente estado (amarillo después de rojo)
      }
      break;

    case 3:  // Estado LED amarillo (después de rojo)
      if (currentMillis - previousMillis >= yellowInterval) {
        previousMillis = currentMillis;  // Reiniciar el tiempo
        digitalWrite(yellowLED, LOW);    // Apagar el LED amarillo
        digitalWrite(greenLED, HIGH);    // Encender el LED verde
        ledState = 0;  // Volver al estado inicial (verde)
      }
      break;
  }
}
```

Explicación

La función `millis()` devuelve el tiempo en milisegundos. Utilizamos esta función para saber cuánto tiempo ha transcurrido sin detener el resto del programa.

Utilizamos una variable llamada `ledState` para llevar el control de qué LED debe estar encendido en cada momento:

- `0` para el LED rojo,
- `1` para el LED verde,
- `2` para el LED amarillo.

Temporización:

a. Utilizamos la variable `previousMillis` para almacenar el momento en el que cambiamos de estado por última vez.
b. Luego, comparamos el tiempo actual (`currentMillis`) con `previousMillis` para determinar si ha pasado el tiempo necesario (5 segundos para el rojo y verde, 2 segundos para el amarillo).
c. Cuando se cumple el tiempo necesario, cambiamos de estado y reiniciamos `previousMillis` para empezar a contar de nuevo.

Ventajas de `millis()` sobre `delay()`:

- A diferencia de `delay()`, que detiene el programa completo, `millis()` permite que el código siga ejecutándose mientras controlamos el tiempo.
- Usando `millis()`, es posible agregar más funciones al código (como monitorear botones o realizar otras acciones) sin que el tiempo de espera interfiera en ellas.