**Simón dice**

Grupo 3

Leandro Delgado - Ramiro Averbuj - Daiana Rivera - Rodrigo Acevedo - Adrián Hidalgo

Especialidad en Computación, Escuela Técnica N°32 D.E. 14

4to2da: Proyecto Informático

Gonzalo N. Consorti

# **Introducción**

Desde el inicio, este proyecto se perfiló como uno de los más accesibles en concepto y ejecución. Es un juego que todos conocemos bien y que, en teoría, no parece muy complicado de desarrollar. El grupo rápidamente se mostró capaz y motivado para completarlo dentro del tiempo asignado. Este optimismo inicial se basaba en la familiaridad del juego y en la confianza en las habilidades técnicas del equipo.

# **¿Qué es el juego de mesa "Simón dice"?**

"Simón dice" es un popular juego de mesa electrónico que se basa en la memoria y la atención. Este juego, que ha cautivado a jugadores de todas las edades, se centra en seguir secuencias de luces y sonidos. Es ideal para fiestas, reuniones familiares o simplemente para disfrutar en casa con amigos.

## **Objetivo del Juego**

El objetivo principal de "Simón dice" es recordar y repetir correctamente una secuencia que el dispositivo presenta. A medida que avanza el juego, las secuencias se vuelven más largas y complejas, desafiando la memoria y la rapidez de reacción de los jugadores. La emoción aumenta con cada ronda, ya que cada error puede llevar a la eliminación o a perder una vida.

## **Cómo Jugar**

1. Inicio del Juego:
   * Para comenzar, el jugador activa el dispositivo. Generalmente, hay un botón de inicio que inicia la primera ronda.
   * El juego está diseñado para ser intuitivo, lo que permite que jugadores de diferentes edades comprendan rápidamente cómo funciona.
2. Observación de la Secuencia:
   * Una vez iniciado, el dispositivo ilumina una serie de botones en un orden específico, acompañado de sonidos característicos.
   * Los jugadores deben observar atentamente la secuencia presentada. Esta fase es crucial, ya que los jugadores deben memorizar tanto las luces como los sonidos.
3. Repetición de la Secuencia:
   * Después de que la secuencia se ha mostrado, los jugadores deben intentar reproducirla presionando los botones en el mismo orden.
   * La presión correcta de los botones activa las luces y sonidos correspondientes, lo que proporciona una retroalimentación inmediata sobre si han tenido éxito.
4. Aumento de Dificultad:
   * Con cada ronda exitosa, el juego añade una nueva luz y sonido a la secuencia, haciendo que sea más larga y compleja.
   * Este aumento progresivo en la dificultad mantiene a los jugadores comprometidos y desafiados. La sensación de logro al recordar secuencias más largas es muy gratificante.
5. Eliminación por Error:
   * Si un jugador comete un error al intentar repetir la secuencia (por ejemplo, presionando un botón incorrecto), puede ser eliminado o perder una vida, dependiendo de las reglas específicas del juego.
   * En algunas versiones del juego, se permite continuar jugando a pesar del error, pero con penalizaciones como perder puntos o tener que reiniciar desde una secuencia más corta.

## **Variantes del Juego**

* Modos de Juego: Muchas versiones del juego ofrecen diferentes modos, como jugar en solitario o en grupo. En el modo multijugador, los jugadores pueden competir entre sí para ver quién puede recordar las secuencias más largas.
* Niveles de Dificultad: Algunos dispositivos permiten ajustar los niveles de dificultad para adaptarse a diferentes edades y habilidades. Esto hace que el juego sea accesible tanto para niños como para adultos.
* Temáticas Especiales: Existen versiones temáticas que pueden incluir música popular o gráficos coloridos que hacen el juego aún más atractivo.

## **Beneficios del Juego**

Los beneficios que trae jugar a este clásico juego, son las siguientes:

* Desarrollo Cognitivo: "Simón dice" es más que solo diversión; también es un excelente ejercicio mental. Ayuda a mejorar la memoria a corto plazo, la concentración y la agilidad mental. Los jugadores deben estar atentos y ser capaces de recordar información rápidamente.
* Interacción Social: Este juego es ideal para jugar en grupo, fomentando la interacción social y el trabajo en equipo. Las risas y la competencia amistosa crean un ambiente divertido y dinámico.
* Diversión Atractiva: La combinación de luces brillantes y sonidos emocionantes hace que "Simón dice" sea un juego atractivo para todos. La naturaleza interactiva del juego mantiene a los jugadores comprometidos y entretenidos durante largos períodos.

### **Primeros Pasos**

Al comenzar el proyecto, no nos resultó difícil identificar y definir las funcionalidades y características que queríamos implementar. El juego "Simón dice" es un clásico que, en su esencia, es sencillo: generar secuencias de luces y sonidos que el jugador debe repetir correctamente. Sin embargo, pronto nos dimos cuenta de que, aunque el concepto es simple, los detalles técnicos y las mejoras adicionales podrían presentar desafíos interesantes.

### **Planificación del Proyecto**

Iniciamos con una fase de planificación detallada, donde cada miembro del grupo aportó ideas sobre cómo enriquecer el proyecto con funcionalidades adicionales. La primera tarea fue descomponer el juego en sus componentes básicos: generación de secuencias, reproducción de secuencias, lectura de las entradas del jugador, y manejo de errores. Esta etapa también incluyó la definición de las melodías asociadas a cada color y a los fallos, un detalle que, aunque parecía menor, añadía una capa de complejidad y entretenimiento al juego.

**15 de octubre de 2024**

Comencé la carpeta de campo una o dos clases después de empezar el proyecto como tal, pero tengo frescas las ideas y lo que hemos hecho.

Al momento de agarrar este proyecto, el profe Consorti nos indicó que al ser un proyecto más o menos fácil, íbamos a tener ciertos objetivos que debíamos cumplir, sí o sí, para aprobar.

**¿Cuáles son estos objetivos?**

1. Aumentar la Dificultad Progresivamente

Uno de los objetivos clave es que la dificultad del juego aumente a medida que los jugadores completen niveles. Nuestra propuesta es que, cada ciertos niveles (por el momento, cada 10), la dificultad se incremente gradualmente. Esto se logrará a través de varias modificaciones:

* Incremento de la Cantidad de Luces: A medida que los jugadores avanzan, el número de luces que parpadean en cada secuencia aumentará, lo que requerirá una mayor atención y memoria.
* Cambio de Colores: Se introducirán diferentes colores en las secuencias para aumentar la complejidad visual del juego.
* Intercambio de Sonidos: Los sonidos asociados a cada luz también variarán, añadiendo un nivel adicional de desafío.

La estructura de niveles será la siguiente:

* Nivel 1 (Muy Fácil): Comenzará con una secuencia simple para familiarizar al jugador con el juego.
* Niveles 10 a 20 (Fácil): La dificultad aumentará ligeramente, presentando secuencias un poco más largas pero aún manejables.
* Niveles 20 a 30 (Normal o Medio): Aquí, los jugadores enfrentarán un reto moderado, con secuencias que requerirán más concentración.
* Niveles 30 a 40 (Difícil): En esta etapa, se invertirán los colores y sonidos, y las secuencias serán considerablemente más largas y confusas.
* Niveles 40 a 50 (Muy Difícil): Los niveles finales exigirán la máxima concentración del jugador, con secuencias extensas y complejas que pondrán a prueba todas sus habilidades. El objetivo es que estos niveles sean lo más desafiantes posible.

2. Implementar una Pantalla LED:

La incorporación de una pantalla LED es un objetivo importante y relativamente sencillo, ya que Ramiro tiene experiencia en este tema y ha estado investigando sobre su implementación. A través de la pantalla LED, podremos ofrecer al jugador información clave:

* Puntuación: Mostrar la puntuación actual del jugador para mantenerlo motivado.
* Nivel Actual: Indicar en qué nivel se encuentra el jugador para que pueda seguir su progreso.
* Dificultad: Informar sobre el nivel de dificultad actual para preparar al jugador para los desafíos que se avecinan.
* Mensajes Ocasionales: Incluir mensajes motivacionales o de advertencia durante el juego para enriquecer la experiencia.

3. Sonidos y Melodías:

Otro objetivo es implementar un buzzer que genere sonidos específicos para cada acción en el juego. Esto incluirá:

* Sonido Inicial: Una melodía que suene al comenzar el juego para crear una atmósfera atractiva.
* Transiciones entre Niveles: Un sonido distintivo al pasar de un nivel a otro, celebrando el avance del jugador.
* Error del Jugador: Un sonido específico para indicar cuando un jugador comete un error, lo que añade un elemento de retroalimentación inmediata.
* Finalización de Niveles: Una melodía especial al completar los 50 niveles, celebrando el logro del jugador.

4. Entretenimiento:

El objetivo principal es garantizar que el proyecto funcione correctamente; sin embargo, también queremos crear un juego verdaderamente entretenido. La idea es que "Simón dice" se convierta en una opción divertida para jugar en momentos de ocio. Queremos que los jugadores piensen en nuestro proyecto cuando busquen algo divertido para hacer, convirtiéndolo en una actividad agradable y adictiva que puedan disfrutar repetidamente.

**Investigación**

Empecé por mi cuenta en mi casa buscando información desde la inteligencia virtual ‘Microsoft Copilot’ que el profesor Consorti nos recomendó usar para investigar y darnos una idea de que podríamos necesitar a la hora de llevar a cabo el proyecto.

Gracias a Copilot, y a proyectos/trabajos realizados anteriormente, sabía que debíamos utilizar la biblioteca LiquidCrystal para poder utilizar la pantalla LCD correctamente.

Los componentes que vamos a usar son:

* 4 luces LED: son fundamentales para la dinámica del juego. Estos LEDs se iluminan de manera secuencial y aleatoria, creando un desafío atractivo para los jugadores. 
* 4 pulsadores: son componentes esenciales para interactuar con las luces del juego "Simón dice". Cada pulsador está diseñado para activar una luz y un sonido específicos, lo que permite a los jugadores participar de manera dinámica en el juego.



* Display LCD: Muestra el nivel actual en el que te encuentras, así como el tiempo que has estado jugando hasta que pierdes. Al finalizar, el contador se reiniciará a cero, comenzando de nuevo la partida.

****

* Buzzer: emite un sonido cada vez que pierdes, así como diferentes tonos para cada color y un sonido especial cuando ganas.

****

* 5 resistencias: principalmente utilizaremos resistencias de 220 ohmios. Estas resistencias desempeñan un papel fundamental en la protección y el funcionamiento adecuado de los demás componentes electrónicos.



Ya en clase, comenzamos a trabajar en Tinkercad, donde nos organizamos para instalar los componentes necesarios y explorar cada uno de ellos de manera individual. Al mismo tiempo, debatimos sobre la posibilidad de agregar elementos adicionales y discutimos cómo implementar cada función que teníamos en mente. Vimos varios videos que mostraban cómo otras personas habían construido su propio "Simón dice" utilizando Arduino. En esos videos, observamos las funciones que habían incorporado, si habían utilizado alguna biblioteca que nosotros no conocíamos, así como la manera en que colocaban los componentes en la protoboard y cómo realizaban las conexiones.

En un trabajo colaborativo, Ramiro, Daiana y yo diseñamos un circuito en Tinkercad con las conexiones que considerábamos más efectivas. Por otro lado, Adrián, Franco y Rodrigo realizaron un diseño similar de manera independiente. Al finalizar, comparamos nuestros diseños y decidimos optar por el Tinkercad que habían creado ellos, aunque con algunas modificaciones. Entre los cambios que realizamos, incluimos ajustes en las conexiones y añadimos una pantalla LCD que no estaba presente en su versión inicial.

Este proceso de colaboración y comparación fue fundamental para enriquecer nuestro proyecto, ya que nos permitió integrar diferentes ideas y mejorar el diseño final del juego "Simón dice".

# **15 de octubre**

Hoy decidimos avanzar significativamente en el desarrollo del código, después de haber instalado correctamente todos los componentes y verificar su funcionamiento adecuado. Comenzamos evaluando las características principales que abordaríamos inicialmente, y decidimos, en primer lugar, establecer la base del juego. Implementamos las funciones esenciales para generar secuencias, reproducirlas, leerlas, detectar errores y asignar las melodías correspondientes tanto a las luces como a los fallos.

## **Desarrollo Inicial**

Con la base del juego establecida, procedimos a instalar y codificar funcionalidades adicionales que mejorarían la experiencia del usuario. Cada función fue diseñada y probada meticulosamente para garantizar su correcto funcionamiento antes de avanzar al siguiente paso.

## **Configuración de la Pantalla LCD**

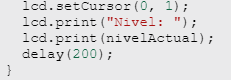
Junto a Ramiro Averbuj, trabajamos en la configuración de la pantalla LCD para que mostrara el nivel actual y el tiempo de juego transcurrido. Al principio, nos enfrentamos a varios problemas, ya que la pantalla no mostraba ningún contenido a pesar de que el código parecía estar correcto. Para resolver esta situación, consultamos al profesor Consorti, quien nos explicó cómo funcionaba el contraste en la pantalla LCD.

Descubrimos que habíamos utilizado una resistencia de 220 ohmios, lo que impedía que la imagen se mostrara adecuadamente. El profesor nos recomendó usar un potenciómetro, lo cual facilitaría la modificación precisa de la intensidad del contraste. Con esta valiosa información, logramos configurar correctamente la pantalla LCD.

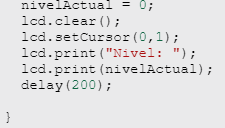
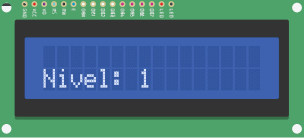
## **Implementación de Funciones**

Una vez que la pantalla estaba operativa, comenzamos a codificar las funciones necesarias para mostrar el nivel y el tiempo en el LCD. Para ello, revisamos tutoriales y practicamos en otros proyectos en Tinkercad. Durante la implementación de la función para mostrar el nivel, encontramos un inconveniente: al cometer un error, el nivel volvía a 1, pero no se mostraba correctamente; aparecía el nivel fallido junto con la nueva secuencia desde el nivel 1. Solo cuando superábamos el primer nivel, la pantalla mostraba "Nivel: 2".

Para resolver este problema, aplicamos técnicas como lcd.clear() y lcd.setCursor(), lo que finalmente solucionó el inconveniente de manera satisfactoria. Con esto, logramos que la pantalla mostrara correctamente el nivel actual y permitiera una experiencia más fluida para los jugadores.



Esta parte del código es para cuando se avance de nivel, vaya cambiando el nivel que se muestra.



En cambio, aquí se muestra cómo vuelve al primer nivel al momento de fallar.

Por el momento, no hemos logrado implementar correctamente la función que controla el tiempo en el juego. Este aspecto es crucial para la dinámica del juego, ya que se espera que los jugadores tengan un tiempo limitado para recordar y repetir las secuencias de luces. A medida que avanzamos en el desarrollo, es fundamental solucionar este inconveniente para mejorar la experiencia del usuario.

## **Ajuste del Nivel de Dificultad**

Simultáneamente, Acevedo e Hidalgo trabajaron en otra computadora para desarrollar la función de ajuste de dificultad. Esta función es esencial, ya que permite que con cada nivel la secuencia se ejecute más rápido y se alargue de manera progresiva. El ajuste automático del nivel de dificultad añade un mayor desafío al juego, manteniendo a los jugadores comprometidos y motivados.

* Incremento de Velocidad: A medida que los jugadores avanzan en los niveles, la velocidad a la que se presentan las secuencias aumentará, lo que requerirá una respuesta más rápida.
* Alargamiento de Secuencias: La longitud de las secuencias también se incrementará gradualmente, lo que exigirá una mayor capacidad de memoria y atención por parte de los jugadores.

## **Resultados y Próximos Pasos**

Después de superar varios desafíos técnicos, logramos configurar correctamente la pantalla LCD y ajustar el código para manejar los niveles y la dificultad de manera eficiente. Aunque aún no hemos implementado completamente la función de tiempo, estamos en buen camino para finalizar este aspecto del proyecto.

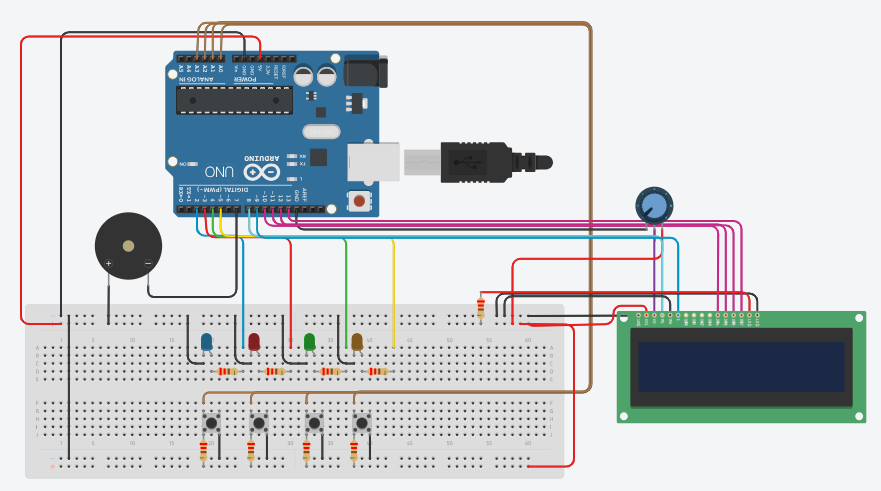
En resumen, nuestro enfoque meticuloso y colaborativo nos permitió avanzar significativamente en el desarrollo del juego "Simón dice", incorporando características avanzadas que mejoran la jugabilidad y la experiencia del usuario. Continuaremos trabajando en la función de tiempo y refinando otras áreas del código para completar el proyecto con éxito.

## **Reflexión final del día**

Hoy fue un día muy productivo; logramos avanzar considerablemente en el código y corregir varios errores. Al final del día, discutimos cómo continuar con el desarrollo del código y qué funciones deberíamos priorizar. Por ahora, nos queda por hacer lo siguiente:

1. Corregir la Función de Tiempo: Es esencial resolver este problema para garantizar una experiencia de juego adecuada.
2. Terminar el Diseño 3D: Completar el diseño físico del proyecto será crucial para su presentación final.

* Con estos pasos claros, estamos motivados para seguir adelante y culminar con éxito nuestro proyecto "Simón dice".

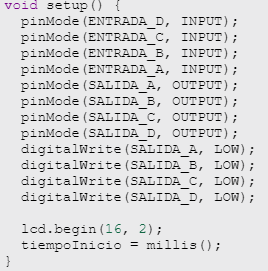
Foto de como terminó todo conectado.

# **22 de octubre de 2024**

Hoy finalizamos la codificación de la función de tiempo. Hidalgo y yo trabajamos en ello por separado y, al comparar nuestros códigos, descubrimos que ambos presentábamos el mismo error: el tiempo se mostraba correctamente, pero no se reiniciaba al perder.

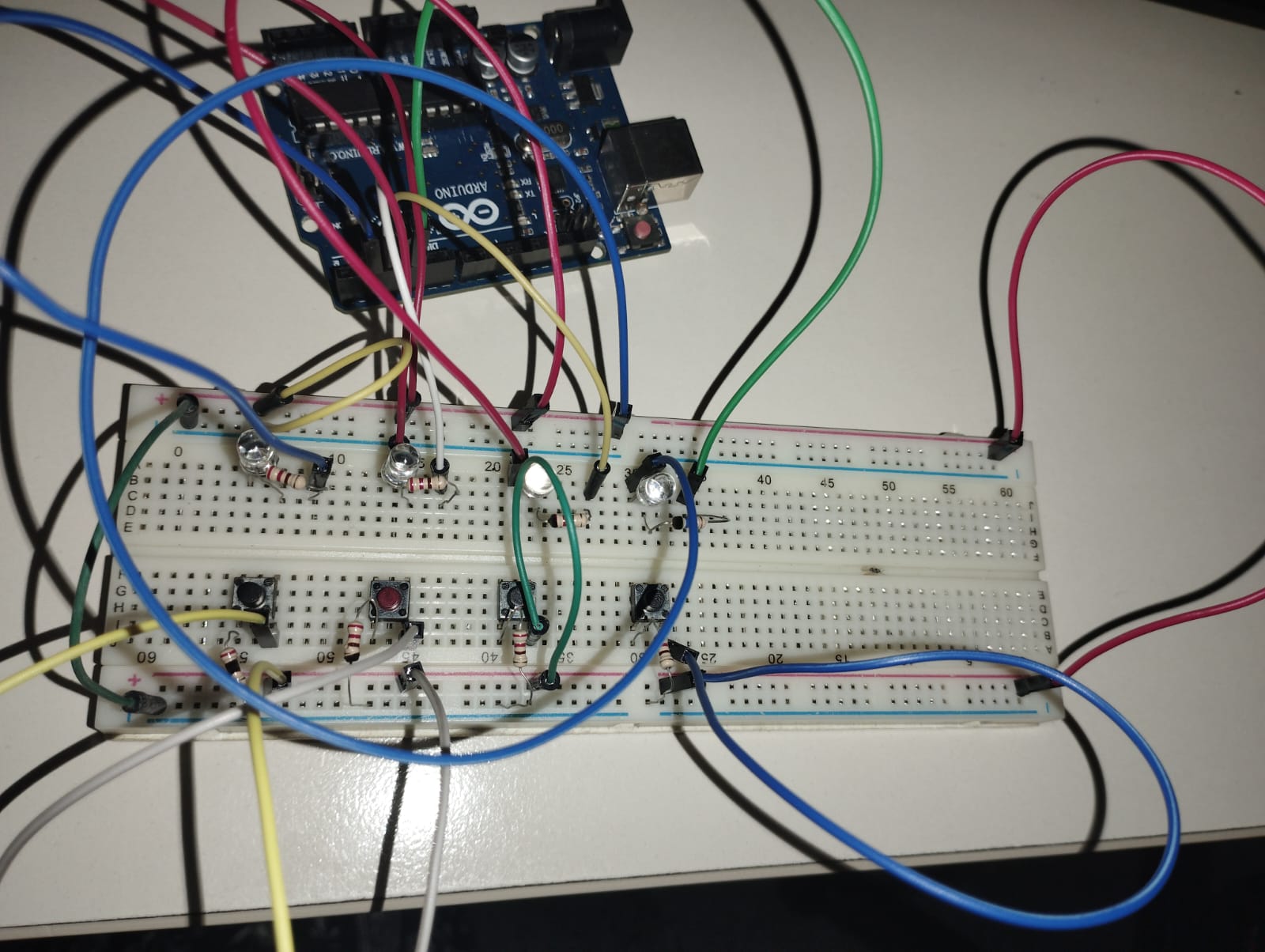
Para solucionar este problema, implementé un enfoque que consistió en dos pasos clave. Primero, declaré una variable global para almacenar el tiempo de inicio.  Utilicé el tipo de dato unsigned long, que es una variable entera capaz de contener solo números positivos y puede almacenar valores grandes. Este tipo es ideal para contar el tiempo en milisegundos o manejar grandes cantidades de datos que siempre son positivos.

En segundo lugar, en la función setup(), inicialicé el tiempo de inicio. Esto asegura que cada vez que comience una nueva partida, la variable registre correctamente el momento en que se inicia el juego. Esta estructura es fundamental para que el temporizador funcione como se espera y se reinicie adecuadamente cuando el jugador comete un error.



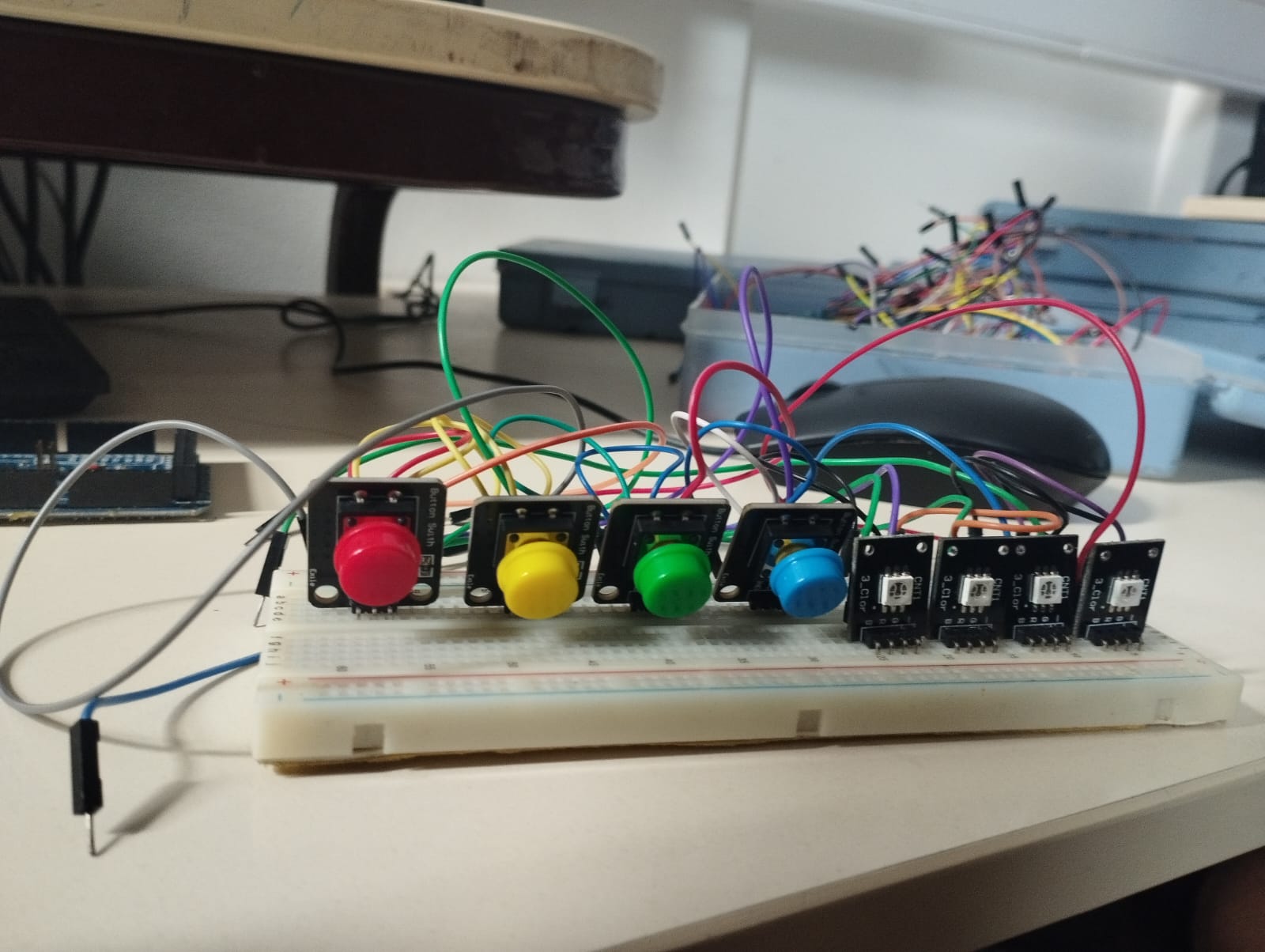
## **29 de octubre de 2024**

Hoy, mientras armábamos físicamente el juego "Simón dice", nos encontramos con varios problemas técnicos que impidieron que el sistema funcionara correctamente. Al ejecutar el código, solo se encendía una luz y los botones no respondían, lo que sugiere que probablemente conectamos algunos componentes incorrectamente. Además, nos dimos cuenta de que no teníamos suficientes pines disponibles para conectar tanto la pantalla LCD como el buzzer. Tuvimos un buen rato intentando encontrar formas de conectar los componentes de manera efectiva. Durante nuestras discusiones, consideramos varias ideas, como reducir la cantidad de luces utilizadas y aumentar en uno la cantidad de luces encendidas en cada secuencia. También contemplamos la posibilidad de utilizar otro tipo de Arduino que tuviera más pines disponibles, entre muchas otras opciones.



A medida que formulábamos estas ideas, no pudimos evitar reírnos con ironía de lo absurdas que parecían algunas de nuestras propuestas. A veces, la frustración nos llevaba a pensar en soluciones poco prácticas, pero eso también nos ayudó a liberar un poco la tensión del momento. Este proceso de brainstorming, aunque lleno de desafíos, fue una parte importante de nuestra colaboración y nos permitió mantener un ambiente positivo mientras buscábamos la mejor manera de avanzar en nuestro proyecto.

Afortunadamente, cuando el profesor Consorti se desocupó, nos vino a ayudar y nos mostró un prototipo ya armado que incluía diferentes botones y luces. Inicialmente, estábamos trabajando con luces LED estándar, pero entendimos que era más apropiado utilizar luces RGB para poder mostrar una gama más amplia de colores. Esto significó que tuvimos que adaptar nuestro código para ajustarnos a esta nueva configuración proporcionada por Consorti. La verdad es que ahora veo esta estructura como más adecuada para un juego interactivo. Al principio nos resultó inquietante como continuar con esto, ya que teníamos todo el proyecto planeado en torno al prototipo en Tinkercad, tuvimos que configurar de nuevo los pines, cambiar el funcionamiento de como se mostraban las secuencias, tuvimos problemas al mostrar los colores que queríamos mostrar, no entendíamos bien que eran los componentes físicos nuevos, como los LEDS RGB, los botones, etc. porque teníamos en mente los componentes como estaban representados en Tinkercad. Pero esto no nos tiró para abajo y logramos solucionar todos estos temas rápidamente.



Sin embargo, aún enfrentamos un problema con la pantalla LCD. Por suerte, Ramiro me explicó que existen dos tipos de pantallas LCD: una con 12 pines, que requeriría conexiones adicionales al Arduino (que era la que estábamos usando) y no sería útil en nuestra situación actual, y otra con 4 pines hembra, conocida como LCD 16x2 (I2C), que es más rentable y, coincidimos en que era más fácil de usar. Este último tipo simplifica las conexiones y facilita la implementación.

# **Explicación del código**

Este código lo hicimos entre Ramiro Averbuj y yo. Estoy bastante feliz del grupo que me tocó, todos trabajan en lo que deben y muy rara vez los encuentro haciendo algo que no sea trabajar en horario de clases.

| #include <LiquidCrystal\_I2C.h> #define BUZZER 7 #define ENTRADA\_A A0 #define ENTRADA\_B A1 #define ENTRADA\_C A2 #define ENTRADA\_D A3 #define SDA A4 #define SCL A5 |
| --- |

* + #include <LiquidCrystal\_I2C.h>: Esta línea incluye la biblioteca necesaria para controlar el LCD a través de la interfaz I2C. Esta biblioteca simplifica la comunicación con el LCD utilizando solo unos pocos pines, lo que es ideal para proyectos donde el número de pines es limitado.
  + #define BUZZER 7: Define el pin 7 como el pin del buzzer.
  + #define ENTRADA\_A A0, #define ENTRADA\_B A1, #define ENTRADA\_C A2, #define ENTRADA\_D A3: Estos pines se definen como entradas para los botones que los jugadores presionarán durante el juego para interactuar con las luces.
  + #define SDA A4 y #define SCL A5: Estos pines se utilizan para la comunicación I2C, donde SDA es la línea de datos y SCL es la línea de reloj.

El código establece las bases para interactuar con los componentes del juego "Simón dice". La biblioteca LiquidCrystal\_I2C permite enviar mensajes y datos al LCD, mientras que los pines definidos permiten gestionar las entradas de los botones y el buzzer. Esto es fundamental para que el juego funcione correctamente, ya que se necesita recibir las entradas del jugador y proporcionar retroalimentación visual y auditiva.

El uso de un LCD 16x2 (I2C) es particularmente ventajoso porque solo requiere cuatro conexiones (VCC, GND, SDA y SCL), lo que reduce la complejidad del cableado. Además, permite mostrar información importante como el nivel actual del juego y el tiempo transcurrido.

Esta parte del código se encarga de definir los pines del display LCD I2C, así como los pines de los botones y del buzzer. A continuación, explicaré la sección que he implementado yo:

| // Pines de los LED RGB int led1R = 3; // Rojo solo int ledB = 2; // Azul solo int led2R = 5; // Rojo del amarillo int led1G = 4; // Verde solo int led2G = 6; // Verde del amarillo  LiquidCrystal\_I2C lcd(0x27, 16, 2); |
| --- |

* int led1R = 3; : Este pin se utiliza para controlar el LED rojo independiente. Al asignarlo al pin 3, podemos encenderlo o apagarlo según las secuencias del juego.
* int ledB = 2; : Este pin está destinado al LED azul. Al igual que el rojo, este LED se activará en función de las secuencias que se generen.
* int led2R = 5; : Este pin controla el LED rojo que forma parte del LED amarillo. Esto permite que el juego utilice colores combinados para enriquecer la experiencia visual.
* int led1G = 4; : Este pin se utiliza para el LED verde independiente. Su activación será parte de las secuencias que los jugadores deben recordar.
* int led2G = 6; : Este pin controla el LED verde que forma parte del LED amarillo, permitiendo la mezcla de colores.
* LiquidCrystal\_I2C lcd(0x27, 16, 2); : Esta línea crea un objeto llamado lcd de la clase LiquidCrystal\_I2C. El constructor toma tres parámetros:
  + Dirección I2C (0x27): Este es el identificador único del LCD en el bus I2C. La dirección puede variar dependiendo de cómo esté configurado el módulo I2C. Para encontrar la dirección correcta, se puede utilizar un escáner I2C.
  + Número de Columnas (16): Este valor indica cuántos caracteres puede mostrar el LCD en una fila. En este caso, se trata de un LCD de 16 columnas.
  + Número de Filas (2): Este valor especifica cuántas filas tiene el LCD. En este caso, se trata de un LCD de 2 filas.

La definición de estos pines es fundamental para el funcionamiento de los LEDs RGB en el juego y del display LCD I2C en el proyecto. Cada LED representa un color específico que se iluminará en secuencias determinadas por el juego. La correcta asignación de pines permite que el microcontrolador (como un Arduino) controle cada LED de manera individual, lo que es crucial para crear las secuencias de luces que los jugadores deben seguir.

| // Definición de secuencia const int NIVEL\_MAX = 50; int secuencia[NIVEL\_MAX]; int secuenciaUsuario[NIVEL\_MAX]; unsigned long tiempoInicio; |
| --- |

Estas líneas de código establecen la estructura fundamental de las variables que controlan diversos aspectos del juego "Simón dice", incluyendo el nivel, la secuencia de luces y la entrada del jugador, así como el seguimiento del tiempo transcurrido desde el inicio del juego.

Desglose de Variables

* const int NIVEL\_MAX = 50;: Esta constante define el número máximo de pasos que puede tener la secuencia en el juego. Al limitar la secuencia a 50 niveles, se establece un marco claro que permite a los jugadores avanzar y enfrentarse a nuevos desafíos a medida que progresan. Este límite no solo ayuda a estructurar el juego, sino que también asegura que los jugadores tengan un objetivo alcanzable y motivador.
* int secuencia[NIVEL\_MAX];: Este arreglo se utiliza para almacenar la secuencia de colores generados automáticamente por el sistema. Cada elemento del arreglo representa un paso en la secuencia que los jugadores deben recordar y replicar. La capacidad de almacenar hasta 50 pasos permite crear secuencias más largas y complejas a medida que avanza el juego, lo que aumenta la dificultad y el desafío para los jugadores.
* int secuenciaUsuario[NIVEL\_MAX];: Este segundo arreglo está destinado a almacenar la secuencia de colores ingresada por el jugador. A medida que los jugadores intentan replicar la secuencia presentada por el sistema, sus respuestas se almacenan aquí para ser comparadas con la secuencia original. Esto es crucial para determinar si el jugador ha tenido éxito o ha cometido un error, lo que influye en su progreso dentro del juego.
* unsigned long tiempoInicio;: Esta variable mide el tiempo transcurrido desde que comenzó el juego. Utilizar un tipo de dato unsigned long permite almacenar grandes cantidades de tiempo en milisegundos sin riesgo de desbordamiento. Esta medición es esencial para implementar funciones relacionadas con el temporizador, como mostrar cuánto tiempo ha pasado desde el inicio o medir la rapidez con la que un jugador completa una secuencia.

Impacto en la Dinámica del Juego

La implementación de estas variables no solo proporciona una base sólida para la lógica del juego, sino que también permite que "Simón dice" sea una experiencia dinámica y emocionante. Al aumentar la dificultad con cada paso y al permitir que los jugadores avancen a través de niveles progresivamente más desafiantes, se evita que el juego se vuelva monótono o repetitivo.

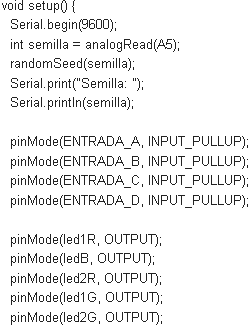
A medida que los jugadores avanzan en el juego, se enfrentan a desafíos cada vez mayores, lo que les brinda una sensación de logro y progreso. Esto no solo mantiene su interés, sino que también fomenta una competencia saludable mientras intentan superar sus propios récords o los de otros jugadores.

Continuando con el código…

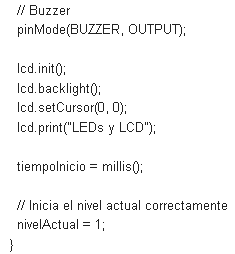
| int melodiaAciert[] = {587, 698, 587, 698, 784, 831, 880, 988}; int duracionNotasAcierto[] = {125, 125, 125, 125, 125, 125, 125, 125}; int melodiaErr[] = {333, 308, 285, 265, 247, 231, 217}; int duracionNotasErr[] = {125, 125, 250, 125, 125, 250, 250}; int nivelActual = 0; int velocidad = 500; |
| --- |
|  |

* int melodia(Aciert/Err)[] : En estas líneas de código, definimos dos arrays llamados melodiaAciert y melodiaErr, que contienen las frecuencias de notas musicales en hertzios (Hz). Las frecuencias dentro de las llaves representan diferentes notas que se utilizarán en el juego para proporcionar retroalimentación auditiva a los jugadores.
* int duracionNotas(Acierto/Err)[] : En estas líneas de código, definimos nuevamente dos arrays llamado duracionNotasAcierto y duracionNotasErr, que especifican la duración de cada nota en la melodía correspondiente a los arrays melodiaAciert y melodiaErr. Cada número dentro de las llaves representa la duración de una nota musical en términos de tiempos musicales.
* int nivelActual = 0; : La variable nivelActual se inicializa en 0 y representa el nivel actual del jugador en el juego. Esta variable es fundamental para la mecánica del juego, ya que determina en qué etapa se encuentra el jugador y, por lo tanto, influye directamente en la dificultad de las secuencias que debe replicar.
* int velocidad = 500; :La variable velocidad se establece en 500 y controla la rapidez con la que se muestran las secuencias de luces en el juego. Este valor se representa en milisegundos, en este caso, serían 500 ms entre cada paso de la secuencia. Esto es fundamental para la dinámica del juego, ya que determina el intervalo de tiempo entre cada paso de la secuencia que el jugador debe seguir. Además, que te permite crear funciones para aumentar la rapidez de las secuencias de forma más sencilla.

Ahora, explicaré el setup, que es crucial para inicializar los componentes y preparar el entorno antes de que el juego comience a ejecutarse:



* Serial.begin(9600);: Esta línea inicia la comunicación serial a una velocidad de 9600 baudios. Esto es esencial para enviar y recibir datos entre el Arduino y la computadora, o cualquier dispositivo conectado. Permite la depuración del código y la visualización de información en el monitor serial.
* int semilla = analogRead(A5);: Aquí se lee el valor analógico del pin A5. Este valor se utilizará como semilla para la generación de números aleatorios. Al leer un valor analógico, se obtiene un número entre 0 y 1023, lo que proporciona una base variable para la función random(), asegurando que las secuencias generadas sean diferentes en cada ejecución del programa.
* randomSeed(semilla);: Esta línea establece la semilla para la función de números aleatorios utilizando el valor leído del pin A5. Esto garantiza que las secuencias de luces generadas por el juego sean impredecibles y variadas en cada partida.
* Serial.print("Semilla: "); y Serial.println(semilla);: Estas dos líneas envían el valor de la semilla al monitor serial para su visualización. Esto es útil para depurar y verificar que se está generando una semilla diferente en cada inicio del juego.
* pinMode(ENTRADA\_A, INPUT\_PULLUP);: Acá se configuran los pines de entrada (A0, A1, A2 y A3) como entradas digitales con resistencias pull-up internas. Esto significa que los pines estarán en un estado alto (HIGH) por defecto y solo cambiarán a bajo (LOW) cuando se presionen los botones correspondientes. Esta configuración simplifica el manejo de entradas y evita la necesidad de resistencias externas.
* pinMode(led1R, OUTPUT); , pinMode(ledB, OUTPUT); , pinMode(led2R, OUTPUT); , pinMode(led1G, OUTPUT); , pinMode(led2G, OUTPUT);: Con estas, se configuran los pines asignados a los LEDs RGB como salidas. Esto permite controlar el encendido y apagado de los LEDs durante el juego, lo cual es fundamental para mostrar las secuencias que deben ser replicadas por los jugadores.

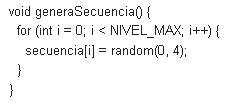
Esa fue la primera parte del setup, ahora vamos con lo que queda de él. Aquí se configuran el buzzer, la pantalla LCD y se establecen otros parámetros iniciales.  


* pinMode(BUZZER, OUTPUT);: Configura el pin asignado al buzzer como una salida. Esto permite emitir sonidos durante el juego. Al definirlo como salida, el programa podrá encender y apagar el buzzer según sea necesario para proporcionar retroalimentación auditiva a los jugadores.
* lcd.init();: Esta línea inicializa la pantalla LCD. Es un paso necesario para preparar el display para recibir y mostrar información. Sin esta inicialización, el LCD no podrá funcionar correctamente.
* lcd.backlight();: Esta instrucción activa la retroiluminación del LCD, lo que mejora la visibilidad del texto mostrado en la pantalla. Esto es especialmente importante en entornos con poca luz, ya que permite a los jugadores leer fácilmente la información presentada.
* lcd.setCursor(0, 0);: Aquí se establece la posición del cursor en la pantalla LCD. En este caso, se coloca en la primera columna (0) y en la primera fila (0). Esto significa que cualquier texto que se imprima a continuación aparecerá en esta ubicación.
* lcd.print("LEDs y LCD");: Con esto se imprime el mensaje "LEDs y LCD" en la pantalla. Este mensaje puede servir como una introducción visual al juego o simplemente como una forma de verificar que el LCD está funcionando correctamente.
* tiempoInicio = millis();: Almacena el tiempo actual en milisegundos desde que comenzó el programa en la variable tiempoInicio. Esto es útil para medir cuánto tiempo ha transcurrido desde que se inició el juego y puede ser utilizado más adelante para implementar características relacionadas con el tiempo.
* nivelActual = 1;: Establece el nivel actual del jugador en 1 al iniciar el juego. Esto indica que el jugador comenzará desde el primer nivel, lo cual es fundamental para la progresión del juego. A medida que avance y complete secuencias correctamente, esta variable se incrementará para reflejar su progreso. A su vez, se utiliza como valor para cuando el usuario pierda, así comienza desde el primer nivel.

Antes de continuar con el loop, me parece que la mejor decisión sería explicar las funciones que creamos. Esta parte del proyecto fue lo que nos llevó más tiempo, pero a su vez, lo que me resultó más divertido/entretenido. Las funciones son:

1. generaSecuencia()
2. muestraSecuencia()
3. leeSecuencia()
4. secuenciaCorrecta()
5. muestraTiempo()
6. secuenciaError()
7. melodiaError()
8. melodiaAcierto()
9. enciendeLED()
10. apagaLEDs()

A continuación, mostraré y explicaré la función “generaSecuencia()”.



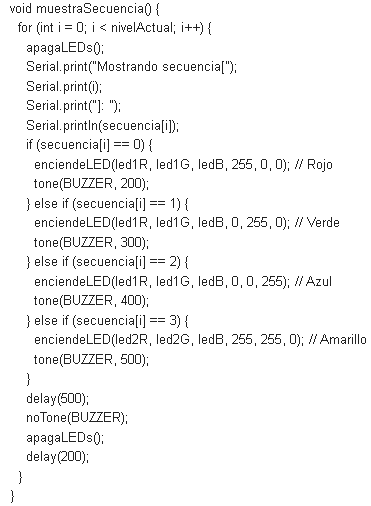
* for (int i = 0; i < NIVEL\_MAX; i++): Este bucle se ejecuta desde 0 hasta NIVEL\_MAX - 1. En cada iteración, se asigna un valor aleatorio a un elemento del arreglo secuencia. Este ciclo asegura que se generen tantos pasos en la secuencia como el número máximo permitido por el juego.
* secuencia[i] = random(0, 4);: En cada iteración del bucle, se utiliza la función random(0, 4) para asignar un número aleatorio entre 0 y 3 a la posición i del arreglo secuencia. Este rango corresponde a los diferentes colores o LEDs que se pueden utilizar en el juego. Por ejemplo:
  + 0 podría representar el LED rojo.
  + 1 podría representar el LED azul.
  + 2 podría representar el LED verde.
  + 3 podría representar el LED amarillo.

Considero que, aunque es una de las funciones más cortas, también es una de las más importantes del juego. Sin una secuencia generada, no habría base para el juego en sí. La secuencia es lo que proporciona el desafío principal: los jugadores deben recordar y replicar los patrones de luces presentados por el sistema.

La generación aleatoria de la secuencia asegura que cada partida sea única, lo que aumenta la rejugabilidad y mantiene el interés del jugador. Cada vez que se inicia un nuevo nivel, los jugadores se enfrentan a un nuevo conjunto de desafíos, lo que les permite mejorar sus habilidades y mantener la emoción del juego.

Siguiendo con la lista, ahora explicaré la función “muestraSecuencia()”. A diferencia de la anterior, esta es bastante más extensa, y fue una de las que más errores (u horrores, en algunos casos) presentó. Algunos fueron:

* No se prendían los LEDs.
* No se mostraban los colores que queríamos.
* Se prendían los LEDs, pero el buzzer no reproducía sonido.

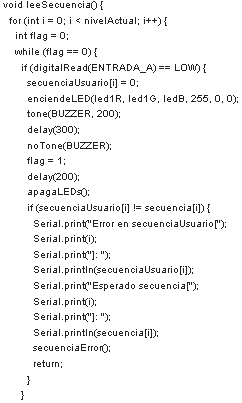


* for (int i = 0; i < nivelActual; i++): Este bucle itera desde 0 hasta el nivel actual del jugador. En cada iteración, se mostrará un paso de la secuencia almacenada en el arreglo secuencia.
* apagaLEDs();: Antes de mostrar cada nuevo paso de la secuencia, se apagan todos los LEDs. Esto asegura que solo el LED correspondiente al paso actual esté encendido.
* Las líneas que utilizan Serial.print() y Serial.println() envían información al monitor serial sobre qué parte de la secuencia se está mostrando. Esto es útil para la depuración y para entender cómo avanza la secuencia.
* Dependiendo del valor en secuencia[i], se ejecutan diferentes acciones:
  + Rojo (0): Se enciende el LED rojo y se reproduce un tono de frecuencia 200 Hz.
  + Verde (1): Se enciende el LED verde y se reproduce un tono de frecuencia 300 Hz.
  + Azul (2): Se enciende el LED azul y se reproduce un tono de frecuencia 400 Hz.
  + Amarillo (3): Se encienden los LEDs amarillo y se reproduce un tono de frecuencia 500 Hz.

Cada llamada a enciendeLED() activa el LED correspondiente con los colores adecuados.

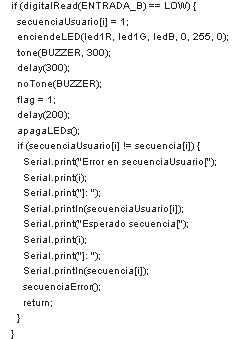
* delay(500);: Después de encender el LED y reproducir el sonido, se introduce una pausa de medio segundo para que los jugadores puedan ver y escuchar claramente cada paso.
* noTone(BUZZER);: Esta línea detiene el sonido del buzzer después del delay.
* apagaLEDs();: Se apagan nuevamente todos los LEDs antes de pasar al siguiente paso.
* delay(200);: Se introduce un segundo delay corto antes de mostrar el siguiente paso para dar tiempo a los jugadores a procesar lo que acaban de ver.

La siguiente función a ser mostrada y explicada, es “leeSecuencia()”. Definitivamente, la más larga y tediosa de hacer, posiblemente me lleve al menos 3 páginas completas de la carpeta en solo esta función. Para respetar la condición de no más de 50% de código o multimedia en la hoja, lo dividiré en partes



* for (int i = 0; i < nivelActual; i++): Este bucle itera desde 0 hasta el nivel actual del jugador. En cada iteración, se procesará un paso de la secuencia que el usuario debe ingresar, permitiendo así comparar las entradas del usuario con la secuencia predefinida.
* int flag = 0: Se inicializa una variable llamada `flag`, que se utiliza para controlar el flujo del bucle interno. Este flag indica si se ha recibido correctamente una entrada del usuario.
* while (flag == 0): Este bucle `while` se ejecuta continuamente hasta que `flag` se establece en 1, lo que significa que se ha procesado una entrada válida. Esto permite que el programa espere hasta que el usuario realice la acción requerida.
* if (digitalRead(ENTRADA\_A) == LOW): Esta línea verifica si la entrada digital correspondiente (definida por `ENTRADA\_A`) está en estado bajo (LOW). Si es así, significa que el usuario ha activado el botón o sensor, indicando su respuesta.
* secuenciaUsuario[i] = 0: Cuando se detecta la entrada del usuario, se registra un valor en el arreglo `secuenciaUsuario[i]`, asignándole un valor de `0`. Esto puede ser parte de un mecanismo para indicar que se ha recibido una entrada.
* enciendeLED(led1R, led1G, ledB, 255, 0, 0):Dependiendo de la entrada del usuario, esta función enciende el LED correspondiente. En este caso, enciende el LED rojo y proporciona una señal visual clara sobre la acción realizada.
* tone(BUZZER, 200): Se emite un tono a través del buzzer con una frecuencia de 200 Hz. Esto proporciona retroalimentación sonora al usuario al momento de registrar su respuesta.
* delay(300): Después de encender el LED y reproducir el sonido, se introduce una pausa de 300 milisegundos. Esto permite que los jugadores vean y escuchen claramente la señal antes de continuar.
* noTone(BUZZER): Esta línea detiene el sonido del buzzer después del delay. Es importante para evitar que el sonido continúe sonando innecesariamente después de que se ha proporcionado la retroalimentación.
* flag = 1: Una vez que se ha procesado la entrada y proporcionado la retroalimentación, se establece `flag` en 1. Esto indica que se ha completado la lectura de la entrada y permite salir del bucle `while`.
* delay(200): Se introduce un segundo delay corto antes de continuar con la siguiente iteración del bucle principal. Esto da tiempo a los jugadores para procesar lo que acaban de ver y escuchar.
* apagaLEDs(): Antes de mostrar cada nuevo paso de la secuencia, se apagan todos los LEDs. Esto asegura que solo el LED correspondiente al paso actual esté encendido y evita confusiones visuales.
* if (secuenciaUsuario[i] != secuencia[i]): Esta condición compara la entrada registrada por el usuario con la secuencia esperada almacenada en `secuencia[i]`. Si hay una discrepancia, significa que el usuario ha cometido un error.
* Serial.print("Error en secuenciaUsuario["): Si hay un error en la secuencia ingresada por el usuario, esta línea envía información al monitor serial indicando el índice donde ocurrió el error. Esto es útil para depuración y para comprender cómo avanza la secuencia.
* Serial.print(i): Imprime en el monitor serial el índice actual donde ocurrió el error en `secuenciaUsuario`.
* Serial.print("]: "): Imprime un separador para facilitar la lectura del mensaje en el monitor serial.
* Serial.println(secuenciaUsuario[i]): Imprime en el monitor serial el valor ingresado por el usuario en esa posición específica. Esto ayuda a identificar qué entrada fue incorrecta.
* Serial.print("Esperado secuencia["): Similar a las líneas anteriores, esta línea imprime en el monitor serial qué valor era esperado en esa posición específica de la secuencia.
* Serial.print(i): Imprime nuevamente el índice actual para indicar dónde debería estar el valor esperado.
* Serial.print("]: "): Imprime un separador para facilitar la lectura del mensaje en el monitor serial.
* Serial.println(secuencia[i]): Imprime en el monitor serial el valor esperado según la secuencia predefinida. Esto proporciona claridad sobre cuál fue la discrepancia.
* secuenciaError(): En caso de un error en la secuencia, se llama a esta función para manejar adecuadamente la situación

Esa fue la primera parte del código, la cual se ejecuta cuando ENTRADA\_A, es decir, cuando el botón correspondiente al led rojo, es presionado. A continuación, explicaré una segunda parte del código.



* if (digitalRead(ENTRADA\_B) == LOW): Esta línea verifica si la entrada digital correspondiente a `ENTRADA\_B` está en estado bajo (LOW). Si es así, significa que el usuario ha activado el botón o sensor asociado a esta entrada, indicando su respuesta.
* secuenciaUsuario[i] = 1: Cuando se detecta la entrada del usuario en `ENTRADA\_B`, se registra un valor en el arreglo `secuenciaUsuario[i]`, asignándole un valor de `1`. Esto indica que el usuario ha seleccionado la opción correspondiente a esta entrada.
* enciendeLED(led1R, led1G, ledB, 0, 255, 0): Esta función enciende el LED correspondiente al valor ingresado por el usuario. En este caso, se enciende el LED verde (RGB: 0, 255, 0), proporcionando una señal visual clara sobre la acción realizada.
* tone(BUZZER, 300): Se emite un tono a través del buzzer con una frecuencia de 300 Hz. Esto proporciona retroalimentación sonora al usuario al momento de registrar su respuesta.
* delay(300): Después de encender el LED y reproducir el sonido, se introduce una pausa de 300 milisegundos. Esto permite que los jugadores vean y escuchen claramente la señal antes de continuar.
* noTone(BUZZER): Esta línea detiene el sonido del buzzer después del delay. Es importante para evitar que el sonido continúe sonando innecesariamente después de que se ha proporcionado la retroalimentación.
* flag = 1: Una vez que se ha procesado la entrada y proporcionado la retroalimentación, se establece `flag` en 1. Esto indica que se ha completado la lectura de la entrada y permite salir del bucle `while`.
* delay(200): Se introduce un segundo delay corto antes de continuar con la siguiente iteración del bucle principal. Esto da tiempo a los jugadores para procesar lo que acaban de ver y escuchar.
* apagaLEDs():Antes de mostrar cada nuevo paso de la secuencia o procesar una nueva entrada, se apagan todos los LEDs. Esto asegura que solo el LED correspondiente al paso actual esté encendido y evita confusiones visuales.
* if (secuenciaUsuario[i] != secuencia[i]): Esta condición compara la entrada registrada por el usuario con la secuencia esperada almacenada en `secuencia[i]`. Si hay una discrepancia, significa que el usuario ha cometido un error.
* Serial.print("Error en secuenciaUsuario["): Si hay un error en la secuencia ingresada por el usuario, esta línea envía información al monitor serial indicando el índice donde ocurrió el error. Esto es útil para depuración y para comprender cómo avanza la secuencia.
* Serial.print(i): Imprime en el monitor serial el índice actual donde ocurrió el error en `secuenciaUsuario`.
* Serial.print("]: "): Imprime un separador para facilitar la lectura del mensaje en el monitor serial.
* Serial.println(secuenciaUsuario[i]): Imprime en el monitor serial el valor ingresado por el usuario en esa posición específica. Esto ayuda a identificar qué entrada fue incorrecta.
* Serial.print("Esperado secuencia["): Similar a las líneas anteriores, esta línea imprime en el monitor serial qué valor era esperado en esa posición específica de la secuencia.
* Serial.print(i): Imprime nuevamente el índice actual para indicar dónde debería estar el valor esperado.
* Serial.print("]: "): Imprime un separador para facilitar la lectura del mensaje en el monitor serial.
* Serial.println(secuencia[i]): Imprime en el monitor serial el valor esperado según la secuencia predefinida. Esto proporciona claridad sobre cuál fue la discrepancia.
* secuenciaError(): En caso de un error en la secuencia, se llama a esta función para manejar adecuadamente la situación. Puede incluir acciones como reiniciar el juego o proporcionar retroalimentación adicional al usuario.

Esta segunda parte, al igual que la primera, es para que se ejecute cuando el ENTRADA\_B, es decir, el botón vinculado al led verde, sea presionado.

A medida que avanzamos en la función leeSecuencia(), es importante destacar que el siguiente segmento se centra en la configuración que permite que las pulsaciones de los botones se relacionen de manera efectiva con el juego, asegurando así que la experiencia sea interactiva y jugable. Este proceso es fundamental para que el usuario pueda participar activamente en la secuencia, y se basa en una lógica similar a las partes anteriores del código.

En esencia, lo que cambia en esta sección es la entrada específica que se utiliza como condición dentro de la instrucción if (digitalRead() == LOW). Cada botón tiene su propia entrada digital, y al presionar uno de ellos, se desencadenan acciones específicas, como encender un LED de un color determinado y emitir un sonido correspondiente a través del buzzer.

| if (digitalRead(ENTRADA\_C) == LOW) {  secuenciaUsuario[i] = 2;  enciendeLED(led1R, led1G, ledB, 0, 0, 255);  tone(BUZZER, 400);  delay(300);  noTone(BUZZER);  flag = 1;  delay(200);  apagaLEDs(); |
| --- |

Lo que queda por explicar en esta función es un manejo similar para el caso en el que se presione el botón vinculado a ENTRADA\_D. En esta sección final del código:

* Se verificará si el botón correspondiente a ENTRADA\_D ha sido activado.
* Se registrará un valor específico en secuenciaUsuario[i].
* Se encenderá un LED con un color particular y se emitirá un sonido a través del buzzer para proporcionar retroalimentación al usuario.

Esta estructura garantiza que cada botón tenga su propia respuesta visual y sonora, creando una experiencia interactiva y dinámica para los jugadores.

Para concluir la implementación de la función leeSecuencia(), es necesario cerrar adecuadamente la llave del bucle while. Este paso asegura que todas las iteraciones y condiciones dentro del bucle se completen antes de proceder a la siguiente parte del código.

Una vez que hemos salido del bucle while, el siguiente paso es llamar la función secuenciaCorrecta(). Al llamarla, se permite que el programa realice las acciones necesarias en caso de que la secuencia sea válida, como avanzar al siguiente nivel del juego o proporcionar retroalimentación positiva al usuario.

Finalmente, cerramos la llave que delimita el final de toda la función leeSecuencia(). Con esto, hemos completado el proceso de lectura y verificación de las entradas del usuario, permitiendo que el juego funcione de manera fluida y eficiente.

Una vez que hemos terminado de explicar la función leeSecuencia(), es momento de adentrarnos en la siguiente función del programa: secuenciaCorrecta(). Esta función se encarga de gestionar el avance del jugador a través de los niveles y ajustar la velocidad del juego en función del progreso.

| **void secuenciaCorrecta() {  if (nivelActual < NIVEL\_MAX)  nivelActual++;  velocidad -= 50 / (nivelActual / 50 + 1);  lcd.setCursor(0, 1);  lcd.print("Nivel: ");  lcd.print(nivelActual);    melodiaAcierto();    delay(200); }** |
| --- |

* if (nivelActual < NIVEL\_MAX) nivelActual++; :Esta línea verifica si el nivel actual en el que se encuentra el jugador es menor que el nivel máximo permitido (NIVEL\_MAX). Si esta condición se cumple, se incrementa el valor de nivelActual en uno. Este incremento permite al jugador avanzar al siguiente nivel, lo que es esencial para mantener el desafío y la emoción del juego.
* velocidad -= 50 / (nivelActual / 50 + 1); :Se ajusta la variable velocidad, que controla la rapidez con la que se presentan los desafíos al jugador. La fórmula utilizada reduce la velocidad en función del nivel actual. A medida que el jugador avanza a niveles más altos, la velocidad disminuye, lo que puede hacer que el juego sea más dinámico y emocionante. Este ajuste es crucial para mantener un equilibrio entre dificultad y jugabilidad.
* lcd.setCursor(0, 1); lcd.print("Nivel: "); lcd.print(nivelActual); : Se actualiza la pantalla LCD para mostrar el nivel actual al jugador. La función setCursor(0, 1) posiciona el cursor en la primera columna de la segunda fila de la pantalla. A continuación, se imprime el texto "Nivel: " seguido del número del nivel actual. Esta retroalimentación visual es importante para mantener al jugador informado sobre su progreso en el juego.
* melodiaAcierto(); :Al final de la ejecución, se invoca la función melodiaAcierto() para reproducir una melodía animada que sirve como una forma de felicitar al jugador por su éxito. Esta melodía proporciona una retroalimentación positiva, celebrando el avance del jugador al siguiente nivel y mejorando la experiencia general del juego.
* delay(200); :Finalmente, se introduce una pausa de 200 milisegundos antes de continuar con cualquier otra operación. Este retraso permite que el jugador tenga tiempo para procesar la actualización en pantalla antes de que se presente el siguiente desafío o acción.

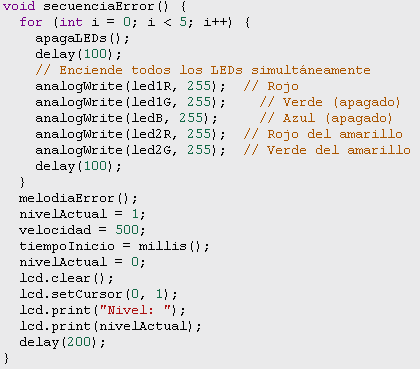
La función secuenciaCorrecta() sirve para gestionar el avance del jugador dentro del juego. Al incrementar el nivel actual y ajustar la velocidad, se asegura que cada sesión de juego sea desafiante y entretenida. Además, al proporcionar retroalimentación visual a través de la pantalla LCD, se mejora la experiencia general del usuario.

Con la explicación de la función secuenciaCorrecta() completa, ahora es momento de abordar la siguiente función en nuestro programa: muestraTiempo(). Esta función tiene como propósito mostrar el tiempo transcurrido desde el inicio del juego en la pantalla LCD. Mantener un registro del tiempo no solo es útil para los jugadores casuales, sino también para aquellos que busquen mejorar su récord personal constantemente.

| void muestraTiempo() {  lcd.setCursor(0, 0);  lcd.print("Time: ");  lcd.print((millis() - tiempoInicio) / 1000); // Muestra el tiempo en segundos  lcd.print(" segs"); } |
| --- |

Comienza posicionando el cursor en la primera fila y columna de la pantalla con lcd.setCursor(0, 0), y luego imprime el texto "Time: " para contextualizar la información. A continuación, calcula el tiempo transcurrido utilizando millis() para obtener el tiempo en milisegundos desde el inicio del programa, restando tiempoInicio y dividiendo el resultado por 1000 para convertirlo a segundos. Finalmente, se añade " segs" para indicar la unidad de medida, proporcionando así una visualización clara y concisa del tiempo que ha pasado, lo que contribuye a mejorar la experiencia del usuario durante el juego.

Luego de explicar la función anterior, es momento de abordar la siguiente parte del código: la función secuenciaError().



A continuación, a partir de la siguiente página del documento, desglosaré esta función en partes para explicarlo de una mejor manera.

| for (int i = 0; i < 5; i++) {  apagaLEDs();  delay(100);  // Enciende todos los LEDs simultáneamente  analogWrite(led1R, 255); // Rojo  analogWrite(led1G, 255); // Verde (apagado)  analogWrite(ledB, 255); // Azul (apagado)  analogWrite(led2R, 255); // Rojo del amarillo  analogWrite(led2G, 255); // Verde del amarillo  delay(100); } |
| --- |

En esta primera parte, se utiliza un bucle que se ejecuta cinco veces para crear un efecto visual llamativo. En cada iteración, primero se apagan todos los LEDs y se introduce un retraso de 100 milisegundos. Luego, todos los LEDs se encienden simultáneamente, incluyendo el LED rojo y los LEDs que forman el color amarillo. Este parpadeo repetido sirve como una clara señal de error para el jugador.

| melodiaError(); nivelActual = 1; velocidad = 500; tiempoInicio = millis(); nivelActual = 0; |
| --- |

Después del bucle visual, se llama a la función melodiaError(). Esta función reproduce un sonido o melodía asociada al error, proporcionando así una retroalimentación auditiva que complementa la señal visual.

A continuación, varias variables se restablecen para reiniciar el estado del juego. nivelActual se establece en 1 para indicar que el jugador debe comenzar desde el primer nivel nuevamente. La variable velocidad se ajusta a 500 milisegundos, lo que es el valor predeterminado para la velocidad del juego. Además, tiempoInicio se actualiza con el tiempo actual utilizando millis(), lo que asegura que el temporizador comience desde cero en este nuevo ciclo.

| lcd.clear(); lcd.setCursor(0, 1); lcd.print("Nivel: "); lcd.print(nivelActual); delay(200); |
| --- |

Finalmente, se limpia la pantalla LCD y se posiciona el cursor en la primera columna de la segunda fila. Se imprime "Nivel: " seguido del valor actual de nivelActual, que ahora es 0. Este paso proporciona al jugador una clara indicación de su estado en el juego tras haber cometido un error.

Esta función se centra en el manejo de errores dentro del juego, proporcionando retroalimentación visual y sonora al usuario cuando se produce un error en la secuencia ingresada. A través de una serie de efectos luminosos y sonoros, secuenciaError() no solo informa al jugador sobre el fallo, sino que también reinicia el estado del juego para permitirle comenzar de nuevo.

Con la explicación de la función secuenciaError() completada, solo me quedan tres funciones por detallar. A continuación, procederé a explicar la función melodiaError().

| void melodiaError() {  for (int i = 0; i < sizeof(melodiaErr) / sizeof(melodiaErr[0]); i++) {  tone(BUZZER, melodiaErr[i], duracionNotasErr[i]);  delay(duracionNotasErr[i] + 62);  } } |
| --- |

* La función utiliza un bucle for que recorre cada nota en el array melodiaErr. Este array contiene las frecuencias de las notas que se van a reproducir.
* El bucle se ejecuta hasta que alcanza el tamaño total del array melodiaErr.
* Dentro del bucle, se llama a la función tone() para activar el buzzer y reproducir una nota con una frecuencia específica. La frecuencia de cada nota está determinada por los valores almacenados en el array melodiaErr.
* La duración de cada nota está almacenada en el array duracionNotasErr.
* Después de reproducir cada nota, la función introduce una pausa usando delay(). La duración de la pausa es igual a la duración de la nota más un valor adicional (en este caso, 62 milisegundos) para garantizar una separación clara entre las notas.

Esta función es crucial para proporcionar una señal auditiva que indica la ocurrencia de un error, mejorando así la interacción del usuario con el dispositivo al hacer evidente cuándo se produce un fallo.

Con la explicación de esta función, solo restan dos más, que son relativamente sencillas. Ya casi hemos terminado con la explicación completa del código.

Ya que hemos explicado la función melodiaError(), es momento de abordar su contraparte, la función melodiaAcierto(). Ambas funciones tienen un propósito similar: reproducir una secuencia de tonos a través del buzzer, pero se utilizan en contextos diferentes. Mientras que melodiaError() indica un error, melodiaAcierto() se usa para señalar un éxito o acierto. Veamos el código de la función melodiaAcierto():

| void melodiaAcierto(){  for (int i = 0; i < sizeof(melodiaAciert) / sizeof(melodiaAciert[0]); i++) {  tone(BUZZER, melodiaAciert[i], duracionNotasAcierto[i]);  delay(duracionNotasAcierto[i] + 65);  } } |
| --- |

* La función melodiaAcierto() utiliza un bucle for para recorrer cada nota en el array melodiaAciert, similar a como lo hace melodiaError(). Este array contiene las frecuencias de las notas que se reproducirán cuando ocurra un acierto.
* El bucle se ejecuta hasta que alcanza el tamaño total del array melodiaAciert.
* Dentro del bucle, se llama a la función tone() para activar el buzzer y reproducir cada nota con una frecuencia específica. Estas frecuencias están almacenadas en el array melodiaAciert.
* La duración de cada nota está determinada por los valores del array duracionNotasAcierto.
* Después de reproducir cada nota, la función introduce una pausa usando delay(). La duración de la pausa es igual a la duración de la nota más un pequeño valor adicional (en este caso, 65 milisegundos) para asegurar una separación clara entre las notas, similar a la función melodiaError().

Una vez explicado “melodiaError()” y “melodiaAcierto()”, llegó la última función del proyecto y la más sencilla de todas: “enciendeLED()”.

| void enciendeLED(int pinR, int pinG, int pinB, int valorR, int valorG, int valorB) {  // Configura los valores para los colores del LED  analogWrite(pinR, valorR);  analogWrite(pinG, valorG);  analogWrite(pinB, valorB); } |
| --- |

Esta función recibe como parámetros los pines correspondientes a los colores rojo, verde y azul del LED, así como los valores que determinan la intensidad de cada color (0 a 255). Al utilizar analogWrite(), la función configura los valores para cada componente del color, mejorando así la experiencia del jugador.

Para finalizar la explicación del código, lo único que falta es mostrar y explicar la función loop(). La función loop() es el corazón del programa, encargándose de gestionar el flujo del juego en cada iteración. Su estructura es simple, pero efectiva, permitiendo que el juego responda a las acciones del jugador. En el primer nivel, se generan y muestran las secuencias que el usuario debe seguir. Esto se logra mediante las funciones generaSecuencia(), muestraSecuencia() y leeSecuencia(), que se invocan secuencialmente para preparar y validar la interacción del jugador con el juego.

En niveles posteriores, el flujo cambia ligeramente: se omite la generación de una nueva secuencia y solo se muestran las secuencias existentes junto con la lectura de las respuestas del usuario. Esto permite que el jugador continúe desafiándose sin tener que reiniciar el proceso de generación de secuencias en cada nivel.

Finalmente, la función muestraTiempo() se llama en cada ciclo para actualizar y mostrar el tiempo transcurrido desde el inicio del juego, proporcionando así una referencia temporal al jugador. Esta estructura asegura que el juego funcione de manera continua y fluida, manteniendo al jugador comprometido y consciente de su progreso.

| void loop() {  if (nivelActual == 1) {  generaSecuencia();  muestraSecuencia();  leeSecuencia();  } else {  muestraSecuencia();  leeSecuencia();  }  muestraTiempo();  } |
| --- |

# **Conclusión del proyecto**

Una vez finalizado el trabajo de explicar el funcionamiento del código, es momento de dar cierre a esta carpeta de campo. Ha sido una experiencia muy interesante y entretenida, sobre todo. El grupo con el que trabajé fue excelente; no podría haber pedido más. Como ya mencioné anteriormente, todos colaboraron de manera increíble desde su respectivo rol, y nos ayudamos mutuamente en todo momento.

Ramiro fue con quien más interactué, ya que ambos nos encargamos de la misma faceta del proyecto: el código. Siempre estaba presentándome ideas, problemáticas y soluciones a estas, lo que hizo que el trabajo fuera mucho más ligero de lo que esperaba. Mis experiencias pasadas en trabajos grupales me habían llevado a tener una aversión hacia este tipo de proyectos en conjunto, pero esta vez fue realmente la excepción. La dinámica de trabajo en equipo fue fluida y colaborativa, lo que facilitó la resolución de problemas y la implementación de nuevas ideas.

Daiana se encargó de monitorear el proyecto de manera efectiva; siempre preguntaba a cada integrante si había algún problema o si podía ayudar en algo. Durante la presentación, tomó la iniciativa sin dudarlo y no nos decepcionó. Desde hace años sé que le apasiona todo lo relacionado con el diseño, ya sea en presentaciones, páginas web o carátulas. Fue muy gratificante contar con su presencia en nuestro grupo, ya que su creatividad y habilidades organizativas aportaron un gran valor al proyecto.

Adrián, por otro lado, fue a quien menos vi en clase debido a que nos sentábamos bastante distanciados y dándonos la espalda. No era común que alguno de los dos se acercara para revisar lo que estaba haciendo el otro. Sin embargo, él fue quien dio forma al primer prototipo que tuvimos, aunque finalmente fue reemplazado por el que nos proporcionó el profesor Gonzalo Consorti. Su contribución inicial fue fundamental para sentar las bases del proyecto.

A Rodrigo lo vi con frecuencia, ya que siempre se acercaba a Ramiro y a mí para preguntarnos cosas y mostrarnos cómo avanzaba con el diseño 3D. Aunque no logramos llevar ese diseño a la práctica, me gustó cómo quedó. Su entusiasmo por el diseño 3D y su disposición para colaborar enriquecieron nuestra experiencia grupal.

Finalmente, estoy yo. Considero que hice un buen trabajo; además de asegurarme del funcionamiento del código, aporté ideas para el diseño 3D, ayudé con las carpetas de campo de mis compañeros, les di recomendaciones y recordatorios. Intenté explicarles si no sabían para qué servía algo y también aprendí bastante de ellos. Por ejemplo, Ramiro me enseñó sobre los diferentes tipos de pantallas LCD disponibles; Daiana me mostró cómo utilizar Canva; Rodrigo me explicó un poco sobre el diseño 3D y cómo se realizaba; y Adrián me mostró distintas formas de hacer lo que tenía en mente.

Una parte muy importante del proyecto fue la ayuda del profesor Gonzalo Consorti. Muchas de las problemáticas que enfrentamos fueron solucionadas gracias a sus recomendaciones. Además, él fue quien nos proporcionó los componentes físicos necesarios para llevar a cabo el proyecto. Su apoyo constante y su experiencia fueron clave para guiarnos a través del proceso.

Este proyecto ha sido muy disfrutable y es algo que siempre tuve interés en realizar: llevar un código creado por mí a un objeto físico utilizable. La satisfacción de ver cómo nuestras ideas cobraban vida a través del trabajo en equipo y la colaboración ha sido una experiencia invaluable que atesoraré en mi formación académica y profesional futura. Estoy agradecido por haber tenido la oportunidad de trabajar con un grupo tan comprometido, lo cual ha hecho que este proceso sea memorable y enriquecedor en todos los sentidos.

# **Referencias**

**Perplexity.** (2024). *Perplexity AI: Tu asistente de búsqueda inteligente*. Recuperado de <https://www.perplexity.ai/>

**Microsoft Copilot.** (2024). *Conversación con Copilot*. Microsoft. Recuperado de <https://copilot.microsoft.com/>

**Bitwise Art.** (2018) *Arduino desde cero en Español - Capítulo 35 - LCD I2C adaptador e instalación de librería específica.* [*https://www.youtube.com/watch?v=kuLgPLrg-cY*](https://www.youtube.com/watch?v=kuLgPLrg-cY)

**Robot UNO.** (2020) *MINIJUEGO con ARDUINO para MEJORAR LA MEMORIA || SIMON DICE || [proyecto explicado paso a paso].* [*https://www.youtube.com/watch?v=Kk6Hax4D6YI*](https://www.youtube.com/watch?v=Kk6Hax4D6YI)