JUEGO DE LABERINTO

PAOLA ANDREA ORDOÑEZ LOPEZ
ING.ELECTRONICA Y TELECOMUNICACIONES, POPAYAN,COLOMBIA,
PAOLAORDONEZL@UNICAUCA.EDU.CO

UNIVERSIDAD DEL CAUCA
FAC.INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
INGENIERÍA ELECTRÓNICA
POPAYÁN-CAUCA
2025

Tabla de contenido

1.RESUMEN

El presente informe detalla el proceso de implementación de un juego de laberinto en dos dimensiones, utilizando una pantalla TFT LCD ILI9341 para la visualización, una placa Arduino. El objetivo principal del proyecto fue desarrollar un juego funcional e interactivo que incluyera varios niveles de dificultad creciente, elementos recolectables, trampas, vidas y temporización, buscó ofrecer una experiencia básica de juego donde el personaje pudiera realizar acciones simples ; como desplazarse a través del laberinto, recolectar ítems, evitar trampas y alcanzar una meta dentro de un límite de tiempo establecido. La metodología empleada incluyo la configuración del entorno de programación de VSCode, la interfaz con la pantalla TFT mediante la comunicación SPI y el uso de las librerías Adafruit_GFX y Adafruit_ILI9341.La lógica del juego se implementó en el lenguaje C++, definiendo la estructura del laberinto(pared, camino, ítem, trampa y meta) mediante matrices, el movimiento del personaje se gestionó a través de botones externos con interrupciones, y con el manejo de los diferentes estados del juego(JUGANDO,GANADO,GAME_OVER_NIVEL_CUMPLIDO).
Como resultado, se logró implementar de forma correcta un juego con tres niveles distintos y funcionales , visualización de sprites para el personaje, paredes, meta, trampas, recolectables, vidas, puntuación y tiempo restante. Se concluyo que el proyecto permitió adquirir nuevos conocimientos en cuanto a programación, además de poner a prueba la creatividad al ser un proyecto libre y tener algunas limitantes en cuanto a los componentes utilizados.

2.MARCO TEORICO

Pantalla TFT ILI9341

La visualización del juego se realiza mediante una pantalla TFT LCD. El módulo TFT ILI9341 contiene un controlador de pantalla con el mismo nombre: ILI9341. Es una pantalla a color que utiliza el protocolo de interfaz SPI(Serial Peripheral Interface) para comunicarse y requiere 4 0 5 pines de control, es comúnmente utilizada debido a su versatilidad y bajo costo. Ofrece una resolución de 240x340, lo cual significa que tiene 76800 pixeles.

Librerías Adafruit e ILI9341

Se utilizaron librerías de Adafruit. La Adafruit_GFX proporciona un conjunto de funciones graficas, permitiendo dibujar formas primitivas como líneas, puntos, círculos, rectángulos, texto y también mapas de bits.

La Adafruit_ILI9341 es una librería especifica para el controlador ILI9341, que implementa las funciones de Adafruit_GFX y maneja la comunicación y comandos necesarios para controlar la pantalla. Esta librería facilita la programación al proporcionar funciones para inicializar la pantalla y dibujar en ella. Estas librerías abstraen la complejidad del manejo directo de hardware, permitiendo centrarse en la lógica del juego.

Protocolo de Comunicación SPI

La Interfaz Periférica Serial(SPI) es un protocolo de comunicación síncrono utilizado para la comunicación de corta distancia, principalmente en sistemas embebidos. (“Cómo usar la comunicación SPI con Arduino y Raspberry Pi”) Permite la comunicación full-dúplex entre un dispositivo maestro.(en este caso, el Arduino) y uno o mas dispositivos esclavos(la pantalla ILI9341).Permite tener una mayor eficiencia y velocidad al transferir los datos gráficos necesarios para actualizar la pantalla.

Arduino Mega 2560

"El Arduino Mega 2560 es una placa de desarrollo basada en el microcontrolador ATmega2560." (“Arduino Mega 2560 | Arduino.cl - Compra tu Arduino en Línea”) Se distingue dentro de la familia Arduino por estar orientada a proyectos de mayor envergadura y complejidad, aquellos que demandan una capacidad de procesamiento superior, una memoria extendida y, fundamentalmente, un número considerablemente mayor de pines de entrada y salida (I/O) en comparación con las placas estándar. Esta configuración la hace particularmente idónea para aplicaciones avanzadas como el control de impresoras 3D, sistemas robóticos complejos, implementaciones de domótica y sistemas de adquisición de datos con múltiples sensores.

Las características fundamentales de la placa Arduino Mega 2560, son:

- Microcontrolador principal: Integra el chip ATmega2560.
- Voltaje de Operación Lógica: Establecido en 5V.

- Rango de Voltaje de Alimentación(Recomendado): Se sugiere un rango de 7V a 12V a través del conector de alimentación externo.
- Rango de Voltaje de Alimentación(Limites): La placa puede tolerar voltajes entre 6V y 20V, aunque no se recomienda operar en los extremos de ese rango de forma continua.
- Pines Digitales de Entrada y Salida: Dispone de un total de 54 pines digitales.
- Pines con Modulación por Ancho de Pulso (PWM): De los 54 pines digitales, 15 pueden ser configurados para generar señales PWM.
- Pines de Entrada Analógica: Cuenta con 16 pines dedicados a la lectura de señales analógicas.
- Corriente Máxima por Pin I/O : La corriente máxima que puede suministrar o drenar cada pin I/O es de 20mA.
- Corriente Máxima para el pin de 3.3V: El regulador de 3.3V puede suministrar una corriente máxima de 50 mA.
- Memoria Flash: Posee 256 KB de memoria Flash para el almacenamiento del programa. Es importante notar que 8 KB de esta memoria son reservados para el gestor de arranque(Bootloader).
- Memoria SRAM: Dispone de 8KB de memoria SRAM(Static Random-Access Memory) para almacenamiento de variables durante la ejecución del programa.
- Memoria EEPROM: Incluye 4 KB de memoria EEPROM(Electrically Erasable Programmable Read- Only Memory) para almacenamiento no volátil de datos.
- Frecuencia de Operación(Reloj) : Opera a una velocidad de reloj de 16 MHz.
- Interfaces de Comunicación Serial(UART) : Ofrece 4 puertos seriales(UART) gestionados por hardware, lo que facilita la comunicación con múltiples dispositivos seriales simultáneamente.
- Otras Interfaces: Soporta nativamente los protocolos de comunicación TWI (compatible con I2C) y SPI, además de la comunicación USB gestionada por un microcontrolador secundario (ATmega16U2).

Fundamentos de Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que se fundamenta en el concepto de “objetos”. Estos objetos agrupan tanto datos (en forma de atributos o propiedades) como los procedimientos o funciones que operan sobre esos datos (conocidos como métodos). A diferencia de la programación procedural tradicional, que se centra en la lógica y las funciones, la POO organiza el software en torno a estos objetos, que a menudo modelan entidades del mundo real o conceptos abstractos. Este enfoque promueve un mayor modularidad, facilita la reutilización del código y mejora la mantenibilidad de los sistemas de software complejos. Lenguajes como C++. Java, Python y C# son prominentemente orientados a objetos.

La POO se sustenta en varios principios clave que permiten estructurar el software me manera eficaz:

- Encapsulación: Este principio consiste en agrupar los datos (atributos) y los métodos que los manipulan dentro de una misma unida, el objeto. Además, implica el ocultamiento de la información, restringiendo el acceso directo a los datos internos del objeto desde el exterior

y exponiendo únicamente una interfaz pública (métodos) para interactuar con él. Esto protege la integridad de los datos y reduce las dependencias entre distintas partes del sistema.

- Abstracción: Se refiere a la capacidad de mostrar las características esenciales de un objeto, ocultando los detalles complejos de su implementación interna. Permite a los programadores interactuar con objetos a un nivel conceptual más alto, sin necesidad de conocer todos los pormenores de su funcionamiento, lo que simplifica el diseño y el uso del software.
- Herencia: Es un mecanismo que permite a una clase (plantilla para crear objetos) heredar atributos y métodos de otra clase existente (llamada clase base o superclase). La nueva clase (clase derivada o subclase) puede extender o modificar el comportamiento heredado. La herencia es fundamental para la reutilización del código y para establecer relaciones jerárquicas entre clases.
- Polimorfismo: Significa "muchas formas". En POO, se refiere a la capacidad de los objetos de presentar una interfaz común para diferentes implementaciones. Permite que objetos de distintas clases respondan al mismo mensaje (llamada a un método) de maneras específicas a su clase. Esto se logra comúnmente a través de la sobrecarga de métodos y la sobreescritura de métodos en clases heredadas, lo que aumenta la flexibilidad y extensibilidad del código.

3.DESARROLLO

Especificación del Sistema

El sistema se basa en un microcontrolador que controla a una pantalla TFT ILI9341 de 240x320 pixeles. Las interacciones se realizan por medio de cuatro botones pulsadores(Arriba, Abajo, Derecha e Izquierda) conectados a los pines digitales de una placa Arduino Mega(20,21,18 y 19) respectivamente, los cuales están configurados como entradas y son manejados por interrupciones para una respuesta inmediata. Se implementa un tiempo anti-rebote de 10ms para evitar lecturas múltiples por una sola pulsación.
Se incluye un buzzer, configurado como salida, conectado un pin digital(8), el cual actúa como un dispositivo de salida de audio, proporcionando un apoyo auditivo para eventos dentro del juego como lo son ganar o perder.
Consta de tres niveles con laberintos definidos en matrices de 11 filas por ocho columnas, mostrados en las Figuras 2,3 y 4, además incluye un sistema de puntuación basado en los ítems recogidos, temporización por nivel(inicia en 40 seg y disminuye 5 seg por nivel) y vidas(inicia con tres vidas). Para ganar, se debe llegar a la meta habiendo recogido todos los ítems. Caer en una trampa genera la perdida de una vida y el reinicio de la posicion del personaje.
La pantalla LCD muestra el laberinto, el personaje y una barra superior con el tiempo restante, el puntaje y el número de vidas.

Diseño y Funcionamiento

Se diseñaron tres laberintos(maze1, maze2, maze3) mediante matrices 2D de enteros. Cada número dentro de la matriz representa un elemento: 1 (Pared),2(Camino), 3 (Meta), 4(Ítem Recolectable) y 5 (Trampa). El personaje se representa por coordenadas (x, y) dentro de la matriz y su movimiento se limita a celdas que no son paredes. Para la representación visual de cada elemento del juego se utilizaron sprites (mapas de bits):Pared, Trampa, Meta, Recolectable, Personaje, Vidas y otros elementos de la interfaz.

El movimiento del personaje se gestiona mediante las funciones de moverPlayer( ) y esMovimientoValido( ), actualizando las coordenadas del personaje y redibujando su Sprite en la nueva posicion esto de acuerdo con el boton que se esté presionando y la utilización de la función dibujarJugador( ) y la función limpiarCelda( ), la cual se encarga de borrar la posicion anterior del personaje al realizar un desplazamiento.

Al moverse a una celda, se verifica su contenido. Si es un ítem, se incrementa el puntaje y se elimina el ítem. Si es una trampa, se pierde una vida y se reinicia la posicion del personaje. Si es la meta y se han recogido todos los ítems, se avanza de nivel o se gana en el caso de estar en el ultimo nivel, por el contario si no se han recogido todos los ítems se pierde el juego.

Se emplean contadores para las vidas y los ítems recogidos, cuyos valores se muestran continuamente en la pantalla de la LCD y también para sumar dificultad se utiliza un contador descendente para el manejo del tiempo limite en cada nivel, donde si llega a cero y el número de ítems recogidos es menor al numero total de estos el juego termina.

El juego presenta tres niveles, cada uno con un laberinto y tiempo límite. Además, el juego transita por diferentes estados; JUGANDO,GAME_OVER, GANAR y NIVEL_CUMPLIDO, estos estados ocurren de acuerdo con las acciones que realice el personaje y a su vez según el estado activo se generan diferentes acciones dentro del juego.

Como apoyo sonoro y visual , se emplean funciones para reproducir una secuencia de notas y mostrar un mensaje especifico de acuerdo con el estado en el cual este el juego, si el estado es GANAR se reproduce un sonido de victoria y se muestra un mensaje en la pantalla , por el contario si es GAME_OVER se reproduce un sonido de derrota e igualmente se muestra un mensaje.

Implementación

La implementación se realizo en el entorno de desarrollo VSCode, utilizando lenguaje C++ y para el montaje del circuito requerido se utilizó el simulador Wokwi, el cual se visualiza en la **Figura. 1.**

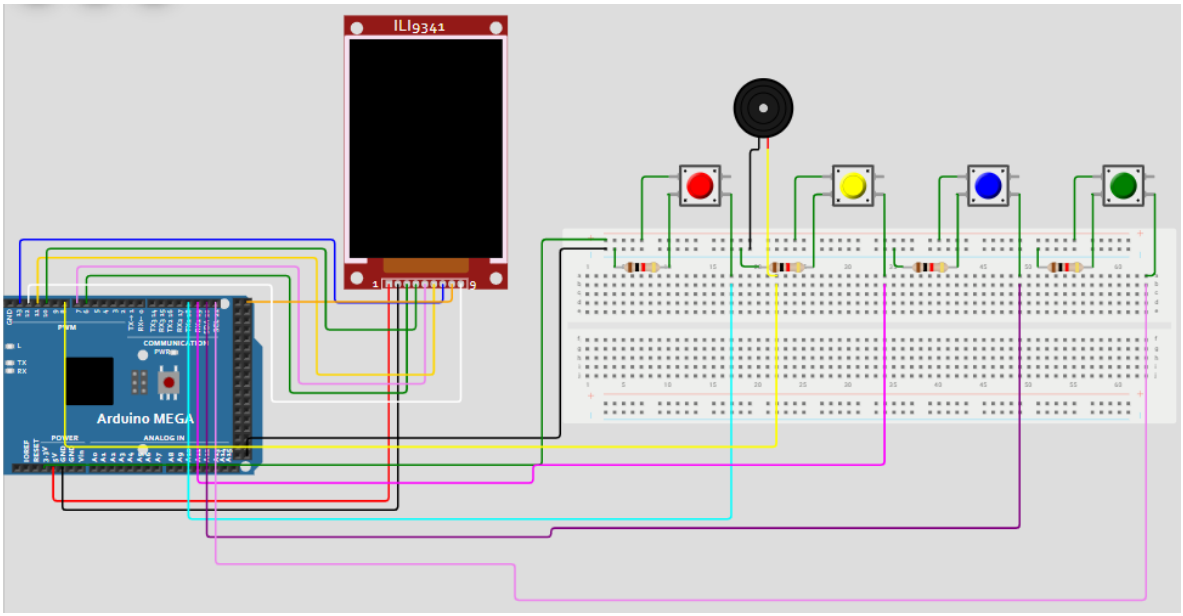

Figura 1. Simulacion en Wokwi.

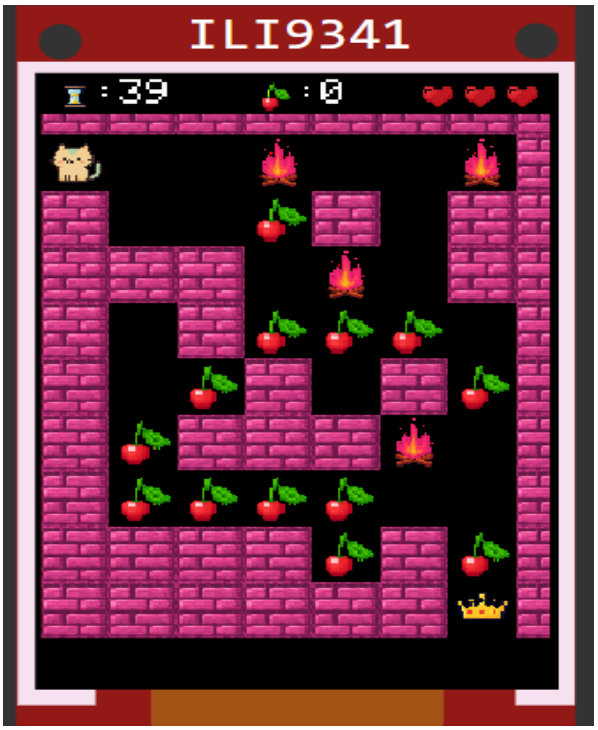

Figura 2.Nivel 1 del juego.

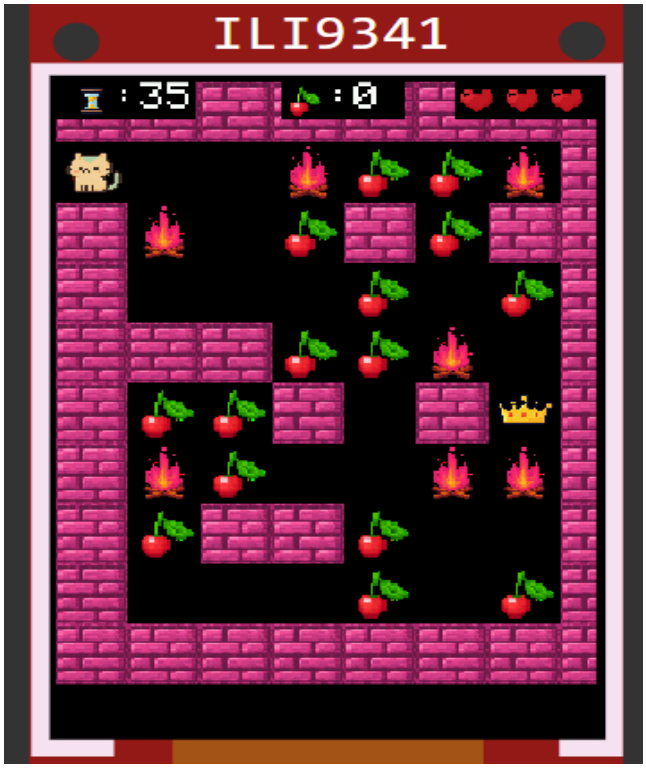

Figura 3.Nivel 2 del juego.

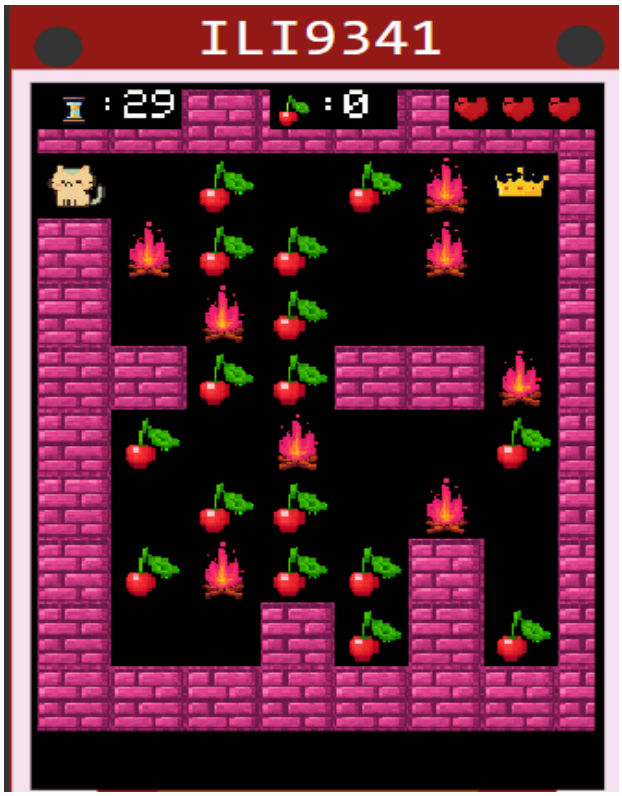

Figura 4.Nivel 3 del juego.

Juego Implementado Con Programación Orientación a Objetos

El sistema se descompuso en varias clases, cada una con responsabilidades bien definidas, que interactúan entre sí para conformar el funcionamiento completo del juego.

Diseño y Responsabilidad de las clases

Clase Game: Esta es la clase central que orquesta el flujo del juego. Tiene como responsabilidades gestionar el estado general del juego(EstadoJuego), inicializar los componentes(setup), ejecutar el bucle principal(update), controlar el dibujado(draw), cargar los niveles(cargarNivel),manejar la lógica de colisiones(VerificarColisiones) y actualizar la interfaz de usuario(HUD).

Contiene instancias de Adafruit_ILI9341 para la pantalla, Maze para el laberinto, Player para el jugador y SoundManager para el audio.Tambien almacena variables de estado como nivel, vidas, puntaje y tiempolimite.

Posee instancias de Maze, Player y SoundManager y mantiene un puntero a Entrada para recibir las acciones del usuario.

Clase Player: Esa clase representa al personaje controlado por el usuario dentro del laberinto. Sus responsabilidades son gestionar la posicion actual y anterior del personaje, se dibuja a sí mismo en

la pantalla y gestiona su movimiento(mover).Para facilitar el redibujado almacena las coordenadas actuales (x, y) y las anteriores(prevX, prevY). Además, colabora con Adafruit_ILI9341 para el dibujado y con Maze para validar movimientos y limpiar celdas.

Clase Maze: Esta clase encapsula toda la lógica de los laberintos del juego. Tiene como responsabilidades almacenar las estructuras de los diferentes niveles, cargar el nivel actual(loadLevel), dibujar el laberinto en la pantalla(draw), validar si un movimiento es permitido(isMoveValid), permitir obtener y modificar el contenido de una celda(getTile, setTile) , y contar los ítems. Contiene los datos de los laberintos predefinidos y el laberinto actual y colabora con Adafruit_ILI9341 para el dibujado.

Clase Entrada: Esta clase se dedica exclusivamente a gestionar las entradas del usuario en este caso botones. Tiene como responsabilidades la configuración de los pines de los botones como entradas y la implementación lógica para leer el estado de estos. Cuando detecta una pulsación, invoca el método moverJugador de la clase Game , esto gracias al puntero a dicha instancia.

Clase SoundManager: Esta clase se encarga de gestionar y reproducir sonidos del juego, que son sonidos de victoria y derrota.

Clase main: Esta clase actúa como el punto de entrada, donde se crean las instancias principales(Game y Entrada) y se inicializa el bucle del juego, delegando las tareas de configuración(setup) y actualización(update) a la instancia de Game, y la lectura de entradas a la instancia de Entrada.

Diagrama De Clases

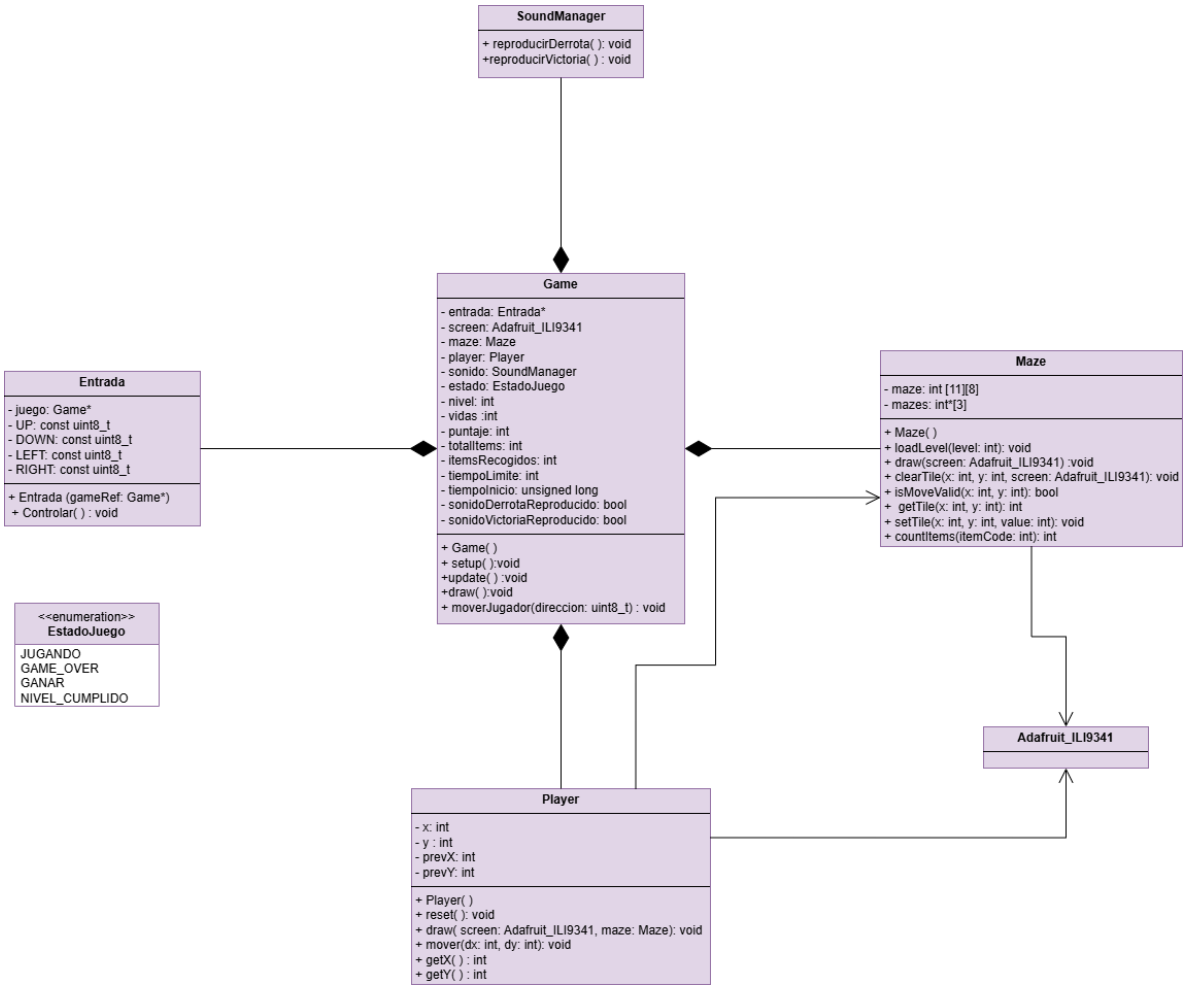

Figura 5.Diagrama de clases.

En el diseño del juego, las clases interactúan entre si mediante distintas relaciones, como se muestra en la Figura 5.

La composición implica una relación en la que el objeto compuesto es responsable del ciclo de vida de sus componentes. Si el objeto contenedor es destruido, también lo son los objetos que contiene. En este caso Game contiene una instancia directa de Maze como atributo privado. El objeto Maze no existe sin el objeto Game, lo que constituye una composición.
Similar al caso anterior, el personaje es parte integral del luego y su instancia es gestionada internamente. SoundManager también es parte del sistema interno de Game, siendo construido y destruido junto a él.

También, Game instancia y tiene un puntero al objeto Entrada, lo cual refleja una relación igualmente de composición, ya que Game es responsable de crear y destruir el objeto de tipo Entrada, el cual no existe sin el juego.
Por último, SoundManager también es instanciada como miembro interno de Game. El juego es responsable de su ciclo de vida, por lo tanto, la relación también es de composición.

La asociación representa una relación más general entre clases, donde una clase usa otra pero no necesariamente la contiene ni depende de su existencia constante.
La clase Player recibe una referencia a Maze para comprobar y limpiar celdas del laberinto. No mantiene una instancia interna de Maze, por lo que se trata de una asociación temporal durante el dibujo.
La clase requiere una pantalla TFT para representar visualmente al personaje, por lo tanto, usa una instancia de la clase Adafruit_ILI9341, lo cual refleja de igual forma una relación asociación, la clase no posee la pantalla ni gestiona su ciclo de vida.
De la misma forma Maze usa temporalmente una referencia a la pantalla para dibujar. No posee la pantalla ni gestiona su ciclo de vida, por lo tanto también es una asociación.

4.Links

- **Simulación Wokwi:** https://wokwi.com/projects/430298448914947073
- **Repositorio GitHub:** paolaordonez/PROYECTOFINAL: Juego De Laberinto, Utilizando una Pantalla TFT ILI9341

4.CONCLUSIONES

En el desarrollo de este proyecto se enfrentaron bastantes desafíos, desde la elección del juego que se realizaría hasta problemas con el desarrollo del código, problemas con algunas funciones y problemas con el hardware utilizado para la simulación el cual tenía algunas limitaciones, específicamente la pantalla TFT. A pesar de estas dificultades, se obtuvieron bastantes conocimientos, al poder darles solución a cada uno de los problemas presentados y al final obtener un proyecto funcional a pesar de cada una de las limitaciones.

En cuanto a funcionamiento, la implemetacion de los estados fue un punto clave dentro del manejo lógico del juego, contribuyo a un funcionamiento más fluido y eficaz de cada una de las funciones , menos redundancia dentro del código .

La implementación del juego en POO, permitió tener un código más claro, estructurado y organizado, facilitando la detección de errores , la comprensión del flujo y la lógica del sistema.
Además, favoreció a una separación de responsabilidades entre las clases, lo que facilito la realización de modificaciones de manera más eficiente y ordenada.

Este proyecto permitió obtener diversos conocimientos en programación, específicamente en el uso del lenguaje C++. A lo largo del proceso se aprendió la lógica de un juego, como gestionar múltiples elementos gráficos de una forma eficiente. La experiencia de programar un juego desde cero permitió comprender la importación de la optimización de un código, el manejo de las redundancias y el correcto manejo de eventos y estados.

5.REFERENCIAS

ILI9341 LCD Controller. (s. f.). ECE353: Introduction To Microprocessor Systems. Recuperado el 27 de mayo de 2025, de: https://ece353-engr-wisc-edu.translate.goog/external-devices/ili9341/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sge

Projects, S. (2024, 4 Junio). Interfacing Arduino with ILI9341 color TFT display. Simple Circuit. Recuperado el 27 de mayo de 2025, de: https://simple--circuit-com.translate.goog/interfacing-arduino-ili9341-tft-display/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

***Adafruit GFX Graphics Library*. (2012, 29 Julio). Adafruit Learning System. Recuperado el 27 de mayo de 2025, de: https://learn.adafruit.com/adafruit-gfx-graphics-library/overview**

Campbell, S. (2024, 11 Noviembre). Basics of the SPI Communication Protocol. Circuit Basics. Recuperado el 27 de mayo de 2025, de :https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/

Arduino. (s.f.-a). *Arduino Mega 2560 Rev3*. Arduino Docs. Recuperado el 27 de mayo de 2025, de https://docs.arduino.cc/hardware/mega-2560-rev3
Arduino. (s.f.-b). *Arduino Mega 2560 Rev3*. Arduino Store. Recuperado el 27 de mayo de 2025, de https://store.arduino.cc/products/arduino-mega-2560-rev3

Microchip Technology Inc. (s.f.). *ATmega640/V-1280/V-1281/V-2560/V-2561/V*. Recuperado el 27 de mayo de 2025, de https://www.microchip.com/en-us/product/ATmega2560

TechTarget. (s.f.). *What is Object-Oriented Programming (OOP)?*. Recuperado el 27 de mayo de 2025, de https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

GeeksforGeeks. (2025b, mayo 15). *Object Oriented Programming in C++*. GeeksforGeeks. Recuperado el 28 de mayo de 2025, de: https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/

TylerMSFT. (s. f.). *Welcome back to C++ - Modern C++*. Microsoft Learn. Recuperado el 28 de mayo de 2025,de: https://learn.microsoft.com/en-us/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-170