Universidad del Valle de Guatemala
Facultad de Ingeniería

Videojuego tipo Bad Ice Cream en NUCLEO-F446RE con ESP32

Proyecto final — Electrónica Digital 2 (IE3054)

Integrantes:	Willy Cuellar (23182), Mario Cano (23589)
Correos:	cue23182@uvg.edu.gt, can23589@uvg.edu.gt
Fecha:	21 de octubre de 2025

Enlace al video

Video del Proyecto

Enlace al repositorio

Repositorio oficial del proyecto en GitHub: Repositorio Proyecto2_Digt2

1. Resumen

Se desarrolló un videojuego inspirado en Bad Ice Cream utilizando una NUCLEO-F446RE como plataforma principal y tres módulos ESP32 para la comunicación inalámbrica y los controles. El sistema incluye una pantalla LCD ILI9341 en modo paralelo, una microSD para almacenamiento de puntajes, un buzzer pasivo controlado por PWM y un protocolo UART que conecta la NUCLEO con un ESP-CAM actuando como puente WiFi-UART. Este trabajo integra conocimientos de control embebido, comunicación serial, manejo de periféricos, almacenamiento, gráficos en tiempo real y sincronización entre dispositivos.

2. Arquitectura general

2.1. Hardware

El sistema se compone de los siguientes elementos:

- **Plataforma principal:** NUCLEO-F446RE, encargada de ejecutar la lógica del juego, renderizar los sprites, controlar el audio y leer datos de los jugadores.
- **Pantalla ILI9341:** Conectada en bus paralelo de 8 bits para alcanzar una velocidad de actualización suficiente para gráficos en movimiento.
- **microSD:** Utiliza SPI1 y el sistema FATFS. Almacena los puntajes de los jugadores en un archivo `scores.txt`.
- **Buzzer pasivo:** Controlado por PWM mediante el temporizador TIM11. Reproduce melodías tipo piano con frecuencias generadas por software.
- **Módulos ESP32:** Dos de ellos actúan como controles con joysticks y botones físicos; el tercero (ESP-CAM) crea la red WiFi y transmite las lecturas a la NUCLEO mediante UART.

2.2. Comunicación

El ESP-CAM crea la red WiFi "`BAD_ICE_CREAM`". Los ESP-J1 y ESP-J2 se conectan y envían paquetes UDP con el formato:

`M1:X123Y245SW0A1B0`

Los datos se combinan y se envían por UART3 a la NUCLEO a 115200 bps, con una frecuencia de actualización de 100 Hz. Cada paquete se interpreta como las coordenadas y acciones del jugador correspondientes.

3. Código del STM32 y explicación

3.1. Estructura general

El programa del STM32 se organiza con una máquina de estados principal (`app_state`) que controla las etapas: menú, juego, pausa y pantalla final. La ejecución sigue un bucle infinito con tareas periódicas.

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART3_UART_Init();
  LCD_Init();
  SD_Init();

  while (1)
  {
    switch(app_state)
    {
      case ST_MENU: Menu_Handle(); break;
      case ST_PLAY: Game_Handle(); break;
      case ST_OVER: GameOver_Handle(); break;
    }
  }
}
```

El código principal configura los periféricos, inicializa la pantalla y gestiona el flujo del juego mediante la máquina de estados. La función `Game_Handle()` coordina los movimientos, colisiones, actualización de pantalla y detección de fin de juego.

3.2. Control de interrupciones UART

Los datos llegan por interrupciones UART desde el ESP-CAM. Cada mensaje con formato `M1:X...;M2:X...` se procesa por una ISR que separa las lecturas en estructuras de entrada.

```
void USART3_IRQHandler(void)
{
  uint8_t byte;
  HAL_UART_Receive(&huart3, &byte, 1, 0);
  UART_Buffer[UART_Index++] = byte;

  if (byte == '\n') {
    ParseInputs(UART_Buffer);
    UART_Index = 0;
  }
}
```

Esta rutina permite capturar información en tiempo real de ambos jugadores y actualizar su estado sin bloquear el flujo del programa principal.

3.3. Colisiones y movimiento

```
void TryMovePlayer(int* x, int* y, int w, int h,
                   int dx, int dy,
                   int ox, int oy, int ow, int oh)
{
  int newX = *x + dx;
  int newY = *y + dy;
  if (!Collides(newX,newY,w,h,ox,oy,ow,oh)
      && Map_IsFree(newX,newY))
  {
    *x = newX;
    *y = newY;
  }
}
```

El algoritmo de colisión verifica la intersección entre los límites del jugador y los obstáculos mediante comparación rectangular (bounding boxes). También valida el mapa con `Map_IsFree()` para evitar moverse sobre celdas bloqueadas. Esto garantiza que los jugadores no atraviesen muros o el contorno de hielo.

3.4. Sistema de almacenamiento en microSD

Cada vez que se termina una partida, se escribe una línea en `scores.txt`:

```
f_open(&file, "scores.txt", FA_WRITE | FA_OPEN_APPEND);
sprintf(buffer, "P1=%d, P2=%d\n", p1_score, p2_score);
f_write(&file, buffer, strlen(buffer), &bw);
```

```
f_close(&file);
```

Esto permite mantener un registro histórico de los puntajes. Si la microSD no está presente, se notifica con "SD ERR" en pantalla, evitando bloqueos.

3.5. Control de tiempos y sincronización

El STM utiliza `HAL_GetTick()` como referencia temporal de 1 ms. Las funciones de animación y música usan comparaciones diferenciales de tiempo para ejecutar acciones periódicas sin bloquear la CPU.

4. Código y funcionamiento de los ESP32

4.1. ESP-CAM

El ESP-CAM actúa como servidor central: crea la red WiFi, recibe los mensajes UDP de los dos controles y reenvía las cadenas por UART a la NUCLEO.

```
while (Udp.parsePacket()) {
  char incoming[128];
  int len = Udp.read(incoming, sizeof(incoming) - 1);
  incoming[len] = '\0';
  String msg = String(incoming);
  if (msg.startsWith("M1:")) dataM1 = msg;
  else if (msg.startsWith("M2:")) dataM2 = msg;
}
Serial.println(dataM1 + ";" + dataM2);
```

El código lee continuamente paquetes UDP, actualiza los mensajes de cada jugador y envía ambos concatenados a la NUCLEO. La transmisión es no bloqueante y con tasa fija de 100 Hz.

4.2. ESP-J1 y ESP-J2

Cada ESP32 lee un joystick (VRX, VRY) y tres botones (SW, A, B). Los valores analógicos se obtienen mediante ADC1, mientras que los digitales se leen con resistencias pull-up internas. El envío se realiza con un retardo de 50 ms para lograr 20 Hz de actualización.

```
int yValue = analogRead(VRY);
int xValue = analogRead(VRX);
int swState = !digitalRead(SW);
int aState  = !digitalRead(BTN_A);
int bState  = !digitalRead(BTN_B);
String paquete = "M1:X" + String(xValue) +
```

```
                    "Y" + String(yValue) +
                    "SW" + String(swState) +
                    "A" + String(aState) +
                    "B" + String(bState);
Udp.beginPacket(SERVER_IP, UDP_PORT);
Udp.print(paquete);
Udp.endPacket();
```

El ESP-J2 envía el mismo tipo de paquete con prefijo `M2`. Ambos se conectan automáticamente al punto de acceso `BAD_ICE_CREAM`.

4.3. Sincronización y estabilidad

Para mantener estabilidad, los ESP-J1/J2 almacenan el último valor del eje X en una variable `cachedX`. Este valor se actualiza cada 800 ms apagando brevemente el WiFi para leer el ADC2 (restringido en uso simultáneo). El ESP-CAM aplica un watchdog de 1.5 s: si no se recibe señal, conserva el último valor válido.

5. Módulo de audio

El buzzer pasivo, conectado a un pin PWM, genera melodías definidas en un arreglo de notas. Cada nota contiene frecuencia y duración. El temporizador TIM11 genera la onda cuadrada con el periodo apropiado, y las melodías se cambian dependiendo del estado del juego (inicio, juego o fin).

6. Resultados y conclusiones

Durante las pruebas se logró mantener una comunicación estable entre todos los módulos, con una latencia promedio de 12-15 ms. Los joysticks respondieron correctamente, los botones se registraron sin rebote perceptible, y las animaciones del juego fueron suaves con una tasa promedio de 30 fps.

La detección de colisiones funcionó de manera confiable usando comparaciones rectangulares. El sistema fue capaz de manejar colisiones múltiples sin pérdida de cuadros ni bloqueos. El almacenamiento FATFS permitió registrar los puntajes de forma persistente. El audio por PWM proporcionó una ambientación funcional y sincronizada con los eventos del juego.

El sistema demostró una integración efectiva entre microcontroladores heterogéneos, utilizando UART, WiFi, SPI y PWM en un solo ecosistema. Como proyección futura se planea:

- Implementar niveles generados desde archivos en microSD.

- Agregar efectos de sonido independientes del canal principal.
- Utilizar Bluetooth Low Energy para los controles.
- Mejorar la IA de los enemigos mediante seguimiento adaptativo.

Anexos

Video del proyecto

Video del Proyecto

Repositorio del proyecto

Repositorio Proyecto2_Digt2