

**[Uma imagem com símbolo, Tipo de letra, Gráficos, captura de ecrã

Os conteúdos gerados por IA poderão estar incorretos.](https://creativecommons.org/licenses/by-nc/4.0/)**

**Luís Simões da Cunha (2025)**

# 🧱 Parte 1 – Iniciar com a SDL: Janela e Renderização Básica

## 🎯 Objetivos:

* Instalar a **SDL2** e compilar usando -lSDL2
* Criar uma **janela** (SDL\_CreateWindow)
* Criar um **renderer** (SDL\_CreateRenderer)
* Desenhar formas básicas (usando SDL\_SetRenderDrawColor, SDL\_RenderClear, SDL\_RenderPresent)

## 🚀 1. Instalação da SDL2

Se ainda não tens a SDL2 instalada:

* No Ubuntu/Linux:
* sudo apt install libsdl2-dev
* No Windows:
  + Faz download do [SDL2 Development Libraries](https://www.libsdl.org/download-2.0.php)
  + Extrai a pasta SDL2 e aponta o compilador para:
    - Includes (-I caminho/SDL2/include)
    - Libraries (-L caminho/SDL2/lib) e liga -lSDL2

Compilar um programa SDL típico:

gcc programa.c -o programa -lSDL2

## 🛠️ 2. Código Completo: Abrir uma Janela e Desenhar

Aqui está o **código minimalista** que cumpre o mini-projeto:

// SDL\_Basico.c  
#include <SDL2/SDL.h>  
#include <stdio.h>  
  
int main(int argc, char\* argv[]) {  
 if (SDL\_Init(SDL\_INIT\_VIDEO) != 0) {  
 printf("Erro ao iniciar SDL: %s\n", SDL\_GetError());  
 return 1;  
 }  
  
 SDL\_Window\* janela = SDL\_CreateWindow(  
 "Emulador: Janela Inicial",   
 SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,   
 640, 480,   
 SDL\_WINDOW\_SHOWN  
 );  
 if (!janela) {  
 printf("Erro ao criar janela: %s\n", SDL\_GetError());  
 SDL\_Quit();  
 return 1;  
 }  
  
 SDL\_Renderer\* renderizador = SDL\_CreateRenderer(  
 janela, -1, SDL\_RENDERER\_ACCELERATED  
 );  
 if (!renderizador) {  
 printf("Erro ao criar renderizador: %s\n", SDL\_GetError());  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
 return 1;  
 }  
  
 // Cor de fundo: preto  
 SDL\_SetRenderDrawColor(renderizador, 0, 0, 0, 255);  
 SDL\_RenderClear(renderizador);  
  
 // Cor do quadrado: branco  
 SDL\_SetRenderDrawColor(renderizador, 255, 255, 255, 255);  
 SDL\_Rect quadrado = { 270, 190, 100, 100 }; // x, y, largura, altura  
 SDL\_RenderFillRect(renderizador, &quadrado);  
  
 SDL\_RenderPresent(renderizador);  
  
 // Espera 3 segundos antes de sair  
 SDL\_Delay(3000);  
  
 // Limpeza  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
  
 return 0;  
}

## 🖥️ 3. O que faz cada parte?

| Código | Função |
| --- | --- |
| SDL\_Init(SDL\_INIT\_VIDEO) | Inicializa o sistema de vídeo |
| SDL\_CreateWindow | Cria uma janela 640x480 pixels |
| SDL\_CreateRenderer | Cria um motor de desenho (hardware/software) |
| SDL\_SetRenderDrawColor | Define a cor atual (R, G, B, Alpha) |
| SDL\_RenderClear | Limpa o ecrã com a cor de fundo |
| SDL\_RenderFillRect | Desenha um retângulo preenchido |
| SDL\_RenderPresent | Mostra no ecrã o que foi desenhado |
| SDL\_Delay(3000) | Aguarda 3000 ms (3 segundos) |
| SDL\_Destroy\* | Liberta recursos antes de sair |

## ✨ Resultado Final

* Janela 640x480 preta
* Quadrado branco (100x100) no meio
* Fecha automaticamente ao fim de 3 segundos.

## 📢 Dicas práticas:

* Queres o programa a correr até o utilizador fechar? Em vez de SDL\_Delay(3000), usa um loop de eventos com SDL\_PollEvent (veremos na Parte 2!)
* SDL usa **RGBA** → o último valor (255) é **opacidade** máxima.
* Todos os valores de cor são de **0 a 255**.

## ✅ Tarefa sugerida:

* Tenta mudar a cor do fundo para azul e o quadrado para vermelho!
* Experimenta alterar a posição do quadrado (SDL\_Rect).

# 🎮 Parte 2 – Eventos e Teclado: Controlar com as Teclas

## 🎯 Objetivos:

* Processar eventos com SDL\_PollEvent
* Detetar pressionar e largar de teclas (SDL\_KEYDOWN, SDL\_KEYUP)
* Criar um **array de estados** de teclas (ideal para emuladores)
* Encerrar o programa pressionando a tecla **ESC**

## 🚀 1. Código Completo: Mover um Quadrado com as Setas

// SDL\_Teclado.c  
#include <SDL2/SDL.h>  
#include <stdio.h>  
#include <stdbool.h>  
  
int main(int argc, char\* argv[]) {  
 if (SDL\_Init(SDL\_INIT\_VIDEO) != 0) {  
 printf("Erro ao iniciar SDL: %s\n", SDL\_GetError());  
 return 1;  
 }  
  
 SDL\_Window\* janela = SDL\_CreateWindow(  
 "Mover Quadrado com Setas",   
 SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,   
 640, 480,   
 SDL\_WINDOW\_SHOWN  
 );  
 if (!janela) {  
 printf("Erro ao criar janela: %s\n", SDL\_GetError());  
 SDL\_Quit();  
 return 1;  
 }  
  
 SDL\_Renderer\* renderizador = SDL\_CreateRenderer(  
 janela, -1, SDL\_RENDERER\_ACCELERATED  
 );  
 if (!renderizador) {  
 printf("Erro ao criar renderizador: %s\n", SDL\_GetError());  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
 return 1;  
 }  
  
 bool correr = true;  
 SDL\_Event evento;  
  
 // Estado das teclas  
 bool teclas[SDL\_NUM\_SCANCODES] = {false};  
  
 SDL\_Rect quadrado = { 270, 190, 100, 100 };  
  
 while (correr) {  
 while (SDL\_PollEvent(&evento)) {  
 if (evento.type == SDL\_QUIT) {  
 correr = false;  
 } else if (evento.type == SDL\_KEYDOWN) {  
 teclas[evento.key.keysym.scancode] = true;  
 if (evento.key.keysym.sym == SDLK\_ESCAPE) {  
 correr = false;  
 }  
 } else if (evento.type == SDL\_KEYUP) {  
 teclas[evento.key.keysym.scancode] = false;  
 }  
 }  
  
 // Movimento baseado no estado das teclas  
 if (teclas[SDL\_SCANCODE\_UP]) quadrado.y -= 5;  
 if (teclas[SDL\_SCANCODE\_DOWN]) quadrado.y += 5;  
 if (teclas[SDL\_SCANCODE\_LEFT]) quadrado.x -= 5;  
 if (teclas[SDL\_SCANCODE\_RIGHT]) quadrado.x += 5;  
  
 // Limpar e desenhar  
 SDL\_SetRenderDrawColor(renderizador, 0, 0, 0, 255); // Preto  
 SDL\_RenderClear(renderizador);  
  
 SDL\_SetRenderDrawColor(renderizador, 255, 255, 255, 255); // Branco  
 SDL\_RenderFillRect(renderizador, &quadrado);  
  
 SDL\_RenderPresent(renderizador);  
  
 SDL\_Delay(16); // Aprox 60 FPS  
 }  
  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
  
 return 0;  
}

## 🧠 2. O que aprendemos?

| Conceito | Explicação |
| --- | --- |
| SDL\_PollEvent | Captura eventos (teclado, rato, fechar janela) |
| SDL\_KEYDOWN / SDL\_KEYUP | Identifica pressionar e largar teclas |
| SDL\_SCANCODE\_\* | Usa códigos de varrimento (scan codes), independentes de teclado físico |
| bool teclas[] | Array que guarda o estado atual de cada tecla |
| SDL\_Delay(16) | Pausa 16 ms para ~60 atualizações por segundo (suavidade) |

## 📦 Como funciona o controlo por array de teclas?

* Quando uma tecla é pressionada (KEYDOWN), o respetivo scancode é marcado como true.
* Quando a tecla é largada (KEYUP), é marcada como false.
* Em cada frame, o programa lê os estados (true/false) e move o quadrado de acordo.

🧠 Isto é **essencial para emuladores** porque permite:

* **Pressionar várias teclas ao mesmo tempo** (ex: correr + saltar num jogo).
* **Controlo fino** dos timings entre pressão e largada.

## ✨ Mini-projeto concluído!

* Quadrado branco movido pelas setas (↑ ↓ ← →).
* Programa termina se carregares na tecla **ESC** ou fechares a janela.

## 📢 Desafios extra:

* Limitar o quadrado a não sair fora da janela!
* Aumentar a velocidade ao carregar na tecla SHIFT?
* Adicionar um fundo colorido diferente conforme a direção (ex: verde para cima, vermelho para baixo).

# 🖼️ Parte 3 – Renderizar Pixéis: Framebuffer e Emulação de Vídeo

## 🎯 Objetivos:

* Criar uma **textura** (SDL\_CreateTexture)
* Atualizar pixéis diretamente com SDL\_UpdateTexture
* Apresentar a textura com SDL\_RenderCopy
* Compreender **formatos de pixéis** (SDL\_PIXELFORMAT\_\*)

## 🚀 1. Mini-projeto: Buffer 256×192 estilo ZX81/Spectrum

Vamos criar um pequeno **framebuffer** 256×192 e pintar manualmente alguns pixéis, como nos primeiros computadores!

## 🛠️ Código Completo:

// SDL\_Framebuffer.c  
#include <SDL2/SDL.h>  
#include <stdio.h>  
#include <stdbool.h>  
  
#define SCREEN\_WIDTH 640  
#define SCREEN\_HEIGHT 480  
#define FB\_WIDTH 256  
#define FB\_HEIGHT 192  
  
int main(int argc, char\* argv[]) {  
 if (SDL\_Init(SDL\_INIT\_VIDEO) != 0) {  
 printf("Erro ao iniciar SDL: %s\n", SDL\_GetError());  
 return 1;  
 }  
  
 SDL\_Window\* janela = SDL\_CreateWindow(  
 "Framebuffer Estilo Spectrum",  
 SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,  
 SCREEN\_WIDTH, SCREEN\_HEIGHT,  
 SDL\_WINDOW\_SHOWN  
 );  
 if (!janela) {  
 printf("Erro ao criar janela: %s\n", SDL\_GetError());  
 SDL\_Quit();  
 return 1;  
 }  
  
 SDL\_Renderer\* renderizador = SDL\_CreateRenderer(janela, -1, SDL\_RENDERER\_ACCELERATED);  
 if (!renderizador) {  
 printf("Erro ao criar renderizador: %s\n", SDL\_GetError());  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
 return 1;  
 }  
  
 SDL\_Texture\* textura = SDL\_CreateTexture(  
 renderizador,  
 SDL\_PIXELFORMAT\_RGBA8888,  
 SDL\_TEXTUREACCESS\_STREAMING,  
 FB\_WIDTH, FB\_HEIGHT  
 );  
 if (!textura) {  
 printf("Erro ao criar textura: %s\n", SDL\_GetError());  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
 return 1;  
 }  
  
 bool correr = true;  
 SDL\_Event evento;  
  
 Uint32\* pixels = malloc(FB\_WIDTH \* FB\_HEIGHT \* sizeof(Uint32));  
 if (!pixels) {  
 printf("Erro ao alocar memória para framebuffer.\n");  
 SDL\_DestroyTexture(textura);  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
 return 1;  
 }  
  
 // Inicializar framebuffer a preto  
 memset(pixels, 0, FB\_WIDTH \* FB\_HEIGHT \* sizeof(Uint32));  
  
 // Vamos desenhar uma linha branca horizontal no meio  
 for (int x = 0; x < FB\_WIDTH; x++) {  
 pixels[(FB\_HEIGHT/2) \* FB\_WIDTH + x] = 0xFFFFFFFF; // Branco RGBA  
 }  
  
 while (correr) {  
 while (SDL\_PollEvent(&evento)) {  
 if (evento.type == SDL\_QUIT) {  
 correr = false;  
 }  
 }  
  
 SDL\_UpdateTexture(textura, NULL, pixels, FB\_WIDTH \* sizeof(Uint32));  
  
 SDL\_SetRenderDrawColor(renderizador, 0, 0, 0, 255); // Preto  
 SDL\_RenderClear(renderizador);  
  
 SDL\_RenderCopy(renderizador, textura, NULL, NULL);  
  
 SDL\_RenderPresent(renderizador);  
  
 SDL\_Delay(16); // ~60 FPS  
 }  
  
 free(pixels);  
 SDL\_DestroyTexture(textura);  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
  
 return 0;  
}

## 🧠 2. O que aprendemos aqui?

| Conceito | Explicação |
| --- | --- |
| SDL\_CreateTexture | Cria uma textura que podemos escrever diretamente (tipo RAM de ecrã) |
| SDL\_PIXELFORMAT\_RGBA8888 | Cada píxel é 4 bytes: Red, Green, Blue, Alpha |
| SDL\_UpdateTexture | Atualiza a memória da textura com os nossos dados |
| SDL\_RenderCopy | Copia a textura para o ecrã |
| pixels[] | Array de memória onde manipulamos os pixéis individualmente |
| Uint32 | Cada píxel é um inteiro de 32 bits (cor completa) |

## 🔥 Dicas práticas:

* **Branco total** em RGBA é 0xFFFFFFFF.
* **Preto** é 0x000000FF.
* Cada linha ocupa FB\_WIDTH \* sizeof(Uint32) bytes.
* Trabalhar diretamente com pixels[] é ideal para **emular ecrãs antigos**!
* Se quiseres cores diferentes, basta alterar o valor dos pixéis:  
  Exemplo: Vermelho puro 0xFF0000FF.

## ✨ Mini-projeto concluído!

* Janela com resolução 640×480
* Framebuffer interno 256×192
* Linha horizontal branca no meio

## 📢 Desafios extra:

* Desenhar um quadrado branco no meio do ecrã.
* Criar um gradiente de cores de cima para baixo.
* Simular "intermitência" piscando a linha branca!

# 🔊 Parte 4 – Áudio com SDL: Beeps, Samples e Som de Emulador

## 🎯 Objetivos:

* Inicializar o sistema de som com SDL\_OpenAudioDevice
* Criar e preencher **buffers de áudio**
* Gerar **ondas simples** (seno, quadrada)
* Reproduzir **beeps** (ex: ao pressionar uma tecla)

## 🚀 1. Introdução rápida ao Áudio na SDL

🎧 No SDL2, o áudio funciona assim:

* Define-se um formato de áudio: frequência (ex: 44100 Hz), amostra (ex: 16 bits), mono ou estéreo.
* Cria-se um **callback** ou envia-se manualmente dados de áudio para a placa de som.
* Para controlo mais fácil, vamos usar **modo push**: enviamos manualmente os dados!

## 🛠️ 2. Código Completo: Tocar um Beep ao pressionar tecla

// SDL\_Beep.c  
#include <SDL2/SDL.h>  
#include <stdio.h>  
#include <stdbool.h>  
#include <math.h> // Para gerar onda seno  
  
#define SAMPLE\_RATE 44100 // 44.1 kHz - padrão CD  
#define BEEP\_DURATION 1 // 1 segundo  
#define AMPLITUDE 28000 // Volume do som  
#define FREQUENCY 440 // Freqüência do beep (440Hz = nota A)  
  
int main(int argc, char\* argv[]) {  
 if (SDL\_Init(SDL\_INIT\_AUDIO | SDL\_INIT\_VIDEO) != 0) {  
 printf("Erro ao iniciar SDL: %s\n", SDL\_GetError());  
 return 1;  
 }  
  
 // Criação de janela para capturar eventos de teclado  
 SDL\_Window\* janela = SDL\_CreateWindow(  
 "Beep ao carregar tecla",  
 SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,  
 640, 480,  
 SDL\_WINDOW\_SHOWN  
 );  
 if (!janela) {  
 printf("Erro ao criar janela: %s\n", SDL\_GetError());  
 SDL\_Quit();  
 return 1;  
 }  
 SDL\_Renderer\* renderizador = SDL\_CreateRenderer(janela, -1, SDL\_RENDERER\_ACCELERATED);  
  
 // Descrição do formato de áudio  
 SDL\_AudioSpec wanted\_spec;  
 SDL\_zero(wanted\_spec);  
 wanted\_spec.freq = SAMPLE\_RATE;  
 wanted\_spec.format = AUDIO\_S16SYS; // 16-bit signed  
 wanted\_spec.channels = 1; // Mono  
 wanted\_spec.samples = 4096; // Tamanho do buffer  
 wanted\_spec.callback = NULL; // Sem callback (modo push)  
  
 SDL\_AudioDeviceID audio\_device = SDL\_OpenAudioDevice(  
 NULL, 0, &wanted\_spec, NULL, 0  
 );  
 if (audio\_device == 0) {  
 printf("Erro ao abrir áudio: %s\n", SDL\_GetError());  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
 return 1;  
 }  
  
 bool correr = true;  
 SDL\_Event evento;  
  
 while (correr) {  
 while (SDL\_PollEvent(&evento)) {  
 if (evento.type == SDL\_QUIT) {  
 correr = false;  
 } else if (evento.type == SDL\_KEYDOWN) {  
 if (evento.key.keysym.sym == SDLK\_ESCAPE) {  
 correr = false;  
 } else {  
 // Gerar beep ao carregar em qualquer tecla  
 int samples = SAMPLE\_RATE \* BEEP\_DURATION;  
 Sint16\* buffer = (Sint16\*)malloc(samples \* sizeof(Sint16));  
  
 for (int i = 0; i < samples; ++i) {  
 double time = (double)i / SAMPLE\_RATE;  
 buffer[i] = (Sint16)(AMPLITUDE \* sin(2.0 \* M\_PI \* FREQUENCY \* time));  
 }  
  
 SDL\_QueueAudio(audio\_device, buffer, samples \* sizeof(Sint16));  
 SDL\_PauseAudioDevice(audio\_device, 0); // Começar reprodução  
  
 free(buffer);  
 }  
 }  
 }  
  
 // Atualizar janela (apenas cor preta)  
 SDL\_SetRenderDrawColor(renderizador, 0, 0, 0, 255);  
 SDL\_RenderClear(renderizador);  
 SDL\_RenderPresent(renderizador);  
  
 SDL\_Delay(16); // ~60 FPS  
 }  
  
 SDL\_CloseAudioDevice(audio\_device);  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
  
 return 0;  
}

## 🧠 3. Entender cada peça

| Comando | Explicação |
| --- | --- |
| SDL\_Init(SDL\_INIT\_AUDIO) | Inicializa o subsistema de áudio |
| SDL\_AudioSpec | Estrutura que define o formato do áudio |
| SDL\_OpenAudioDevice | Abre um dispositivo de áudio com o formato pedido |
| SDL\_QueueAudio | Envia um buffer com samples para serem reproduzidos |
| sin() | Função matemática para gerar uma onda senoidal |
| SDL\_PauseAudioDevice | Retoma a reprodução do som |

## ✨ Resultado:

* Ao carregar numa tecla (qualquer tecla), é gerado um **beep de 1 segundo** (440 Hz).
* Pressionar **ESC** termina o programa.
* Todo o controlo é feito manualmente — **ideal para emuladores** onde queres simular beeps, LOAD "", ou sons básicos.

## 📢 Dicas práticas:

* Para criar uma onda quadrada, basta alternar entre AMPLITUDE e -AMPLITUDE em vez de usar sin().
* Queres gerar sons diferentes dependendo da tecla? Basta mudar FREQUENCY!
* O som é gerado **em memória** antes de ser enviado — isto simula como as máquinas antigas carregavam som diretamente para DACs.

# 🧠 Parte 5 – Temporização e Ciclos da CPU

## 🎯 Objetivos:

* Usar SDL\_GetTicks ou SDL\_GetPerformanceCounter
* Criar **loops de emulação** com tempo fixo por frame (ex: 60 FPS)
* Fazer **throttling** para simular **clock do processador**
* Sincronizar **ecrã + som** com o **tempo real**

## 🚀 1. Conceito-base

🎯 Num **emulador real**, **não basta correr o mais depressa possível** — temos de:

* **Simular o "relógio" da máquina original** (ex: 3,5 MHz no ZX81).
* **Atualizar o ecrã e o som** de forma sincronizada, como acontecia no hardware original.

**Exemplo:** Um ZX81 atualiza o ecrã a cerca de **50 Hz** (PAL) ou **60 Hz** (NTSC).

## 🛠️ 2. Código Base: Ciclo de 60Hz com SDL

// SDL\_Temporizacao.c  
#include <SDL2/SDL.h>  
#include <stdio.h>  
#include <stdbool.h>  
  
#define TARGET\_FPS 60  
#define FRAME\_DURATION\_MS (1000 / TARGET\_FPS)  
  
int main(int argc, char\* argv[]) {  
 if (SDL\_Init(SDL\_INIT\_VIDEO | SDL\_INIT\_AUDIO) != 0) {  
 printf("Erro ao iniciar SDL: %s\n", SDL\_GetError());  
 return 1;  
 }  
  
 SDL\_Window\* janela = SDL\_CreateWindow(  
 "Ciclo 60Hz - Emulador",  
 SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,  
 640, 480,  
 SDL\_WINDOW\_SHOWN  
 );  
 if (!janela) {  
 printf("Erro ao criar janela: %s\n", SDL\_GetError());  
 SDL\_Quit();  
 return 1;  
 }  
  
 SDL\_Renderer\* renderizador = SDL\_CreateRenderer(janela, -1, SDL\_RENDERER\_ACCELERATED);  
  
 bool correr = true;  
 SDL\_Event evento;  
  
 Uint32 tempo\_inicial, tempo\_final, tempo\_passado, tempo\_a\_esperar;  
  
 while (correr) {  
 tempo\_inicial = SDL\_GetTicks();  
  
 while (SDL\_PollEvent(&evento)) {  
 if (evento.type == SDL\_QUIT) {  
 correr = false;  
 }  
 }  
  
 // --- Atualizar estado do emulador aqui (CPU, ecrã, som) ---  
 SDL\_SetRenderDrawColor(renderizador, 0, 0, 0, 255);  
 SDL\_RenderClear(renderizador);  
  
 SDL\_SetRenderDrawColor(renderizador, 255, 255, 255, 255);  
 SDL\_RenderDrawLine(renderizador, 100, 100, 200, 200);  
  
 SDL\_RenderPresent(renderizador);  
 // ----------------------------------------------------------  
  
 // Cálculo de quanto tempo passou  
 tempo\_final = SDL\_GetTicks();  
 tempo\_passado = tempo\_final - tempo\_inicial;  
  
 if (tempo\_passado < FRAME\_DURATION\_MS) {  
 tempo\_a\_esperar = FRAME\_DURATION\_MS - tempo\_passado;  
 SDL\_Delay(tempo\_a\_esperar); // Esperar para manter 60 FPS fixos  
 }  
 }  
  
 SDL\_DestroyRenderer(renderizador);  
 SDL\_DestroyWindow(janela);  
 SDL\_Quit();  
  
 return 0;  
}

## 🧠 3. O que este código faz?

| Parte | Explicação |
| --- | --- |
| SDL\_GetTicks() | Dá o número de milissegundos desde que o programa arrancou |
| tempo\_inicial / tempo\_final | Medem o tempo no início e no fim do ciclo |
| SDL\_Delay() | Espera o tempo necessário para que cada ciclo demore exatamente 16,67ms (60Hz) |

## 🎯 Isto é essencial porque:

* Sem **temporização**, o emulador correria "o mais depressa que o processador atual permite" — o que seria **muito mais rápido** que o hardware original.
* Com **temporização**, controlamos o "batimento do coração" da máquina que estamos a emular.

## ⚡ Alternativa mais precisa: SDL\_GetPerformanceCounter

Se precisares de temporização **de altíssima precisão** (ex: simular clocks de 3,5 MHz com ciclos de CPU individuais!), podes usar:

Uint64 inicio = SDL\_GetPerformanceCounter();  
Uint64 freq = SDL\_GetPerformanceFrequency(); // ticks por segundo  
double elapsed\_ms = (SDL\_GetPerformanceCounter() - inicio) \* 1000.0 / freq;

Esta abordagem evita erros cumulativos em ciclos muito rápidos.

## ✨ Resultado:

* Um **ciclo fixo de 60Hz** controlado ao milissegundo.
* Atualizações suaves do ecrã e futuras atualizações do som também.

## 📢 Desafios Extra:

* Em vez de forçar 60Hz, tenta permitir **trocar entre 50Hz/60Hz** consoante a região (PAL/NTSC).
* Contar quantos frames realmente se desenham por segundo (FPS real).
* Tocar um "beep" sincronizado a cada frame múltiplo de 10.

# 🧩 Parte 6 – Modularizar um Emulador: Arquitetura Simples e Testável

## 🎯 Objetivos:

* Separar o projeto em ficheiros:
  + cpu.c (e cpu.h)
  + video.c (e video.h)
  + main.c
* Definir interfaces entre **CPU → Vídeo → Teclado**
* Preparar estrutura para **carregar ROMs** (futuramente)
* Deixar a base pronta para adicionar uma CPU (ex: **Z80**!)

## 🛠️ 1. Arquitetura Sugerida:

| Módulo | Função Principal |
| --- | --- |
| main.c | Ciclo principal (main\_loop), coordena tudo |
| cpu.c / cpu.h | Simular a CPU (por agora só inicializar e resetar) |
| video.c / video.h | Desenhar e atualizar o ecrã |
| input.c / input.h *(opcional)* | Lidar com teclado e eventos SDL |

📦 **Separar módulos** ajuda a:

* Fazer debug mais fácil.
* Atualizar o código sem quebrar o projeto inteiro.
* Melhorar o desempenho (podes, por exemplo, otimizar só video.c mais tarde).

## 🏗️ 2. Estrutura de Pastas

/emulador/  
 ├── cpu.c  
 ├── cpu.h  
 ├── video.c  
 ├── video.h  
 ├── input.c (opcional)  
 ├── input.h (opcional)  
 ├── main.c  
 └── Makefile (ou CMakeLists.txt)

## 📜 3. Código Base: Esqueleto de um Emulador

### 🔹 cpu.h

#ifndef CPU\_H  
#define CPU\_H  
  
void cpu\_reset();  
void cpu\_step();  
  
#endif

### 🔹 cpu.c

#include "cpu.h"  
#include <stdio.h>  
  
void cpu\_reset() {  
 printf("CPU reset.\n");  
}  
  
void cpu\_step() {  
 // Emular um passo de CPU  
}

### 🔹 video.h

#ifndef VIDEO\_H  
#define VIDEO\_H  
  
#include <SDL2/SDL.h>  
  
bool video\_init();  
void video\_update();  
void video\_destroy();  
  
#endif

### 🔹 video.c

#include "video.h"  
  
static SDL\_Window\* window = NULL;  
static SDL\_Renderer\* renderer = NULL;  
  
bool video\_init() {  
 if (SDL\_Init(SDL\_INIT\_VIDEO) != 0) {  
 return false;  
 }  
  
 window = SDL\_CreateWindow("Emulador Base",  
 SDL\_WINDOWPOS\_CENTERED, SDL\_WINDOWPOS\_CENTERED,  
 640, 480, SDL\_WINDOW\_SHOWN);  
 if (!window) return false;  
  
 renderer = SDL\_CreateRenderer(window, -1, SDL\_RENDERER\_ACCELERATED);  
 if (!renderer) return false;  
  
 return true;  
}  
  
void video\_update() {  
 SDL\_SetRenderDrawColor(renderer, 0, 0, 0, 255);  
 SDL\_RenderClear(renderer);  
  
 SDL\_SetRenderDrawColor(renderer, 255, 255, 255, 255);  
 SDL\_RenderDrawLine(renderer, 100, 100, 200, 200);  
  
 SDL\_RenderPresent(renderer);  
}  
  
void video\_destroy() {  
 if (renderer) SDL\_DestroyRenderer(renderer);  
 if (window) SDL\_DestroyWindow(window);  
 SDL\_Quit();  
}

### 🔹 main.c

#include <SDL2/SDL.h>  
#include "cpu.h"  
#include "video.h"  
  
int main(int argc, char\* argv[]) {  
 if (!video\_init()) {  
 printf("Erro ao iniciar vídeo.\n");  
 return 1;  
 }  
  
 cpu\_reset();  
  
 bool correr = true;  
 SDL\_Event evento;  
  
 Uint32 frameStart, frameTime;  
 const int FRAME\_DELAY = 1000 / 60; // 60 FPS  
  
 while (correr) {  
 frameStart = SDL\_GetTicks();  
  
 while (SDL\_PollEvent(&evento)) {  
 if (evento.type == SDL\_QUIT) {  
 correr = false;  
 }  
 }  
  
 cpu\_step(); // Futuramente emular CPU  
 video\_update(); // Atualizar o ecrã  
  
 frameTime = SDL\_GetTicks() - frameStart;  
 if (FRAME\_DELAY > frameTime) {  
 SDL\_Delay(FRAME\_DELAY - frameTime);  
 }  
 }  
  
 video\_destroy();  
 return 0;  
}

## 🎯 4. Mini-Projeto Concluído:

✅ Estrutura modular  
✅ Janela aberta com SDL2  
✅ Ciclo fixo de 60Hz  
✅ CPU a ser "*reset*-ada" (futuro: emular a CPU real)  
✅ Ecrã atualizado constantemente

# 📚 Próximos Passos:

| O que fazer a seguir? | Descrição |
| --- | --- |
| 1. | Adicionar carregamento de ROM (fopen, fread) |
| 2. | Emular uma CPU simples (instruções básicas Z80) |
| 3. | Ligar input (teclado) à memória |
| 4. | Melhorar som para LOAD "" ou BEEP emulada |
| 5. | Começar a desenhar o framebuffer a partir da memória de vídeo |

# 🔥 Em Resumo:

Dividimos o emulador em blocos claros e comunicantes. Agora estás prontíssimo para adicionar **cérebro** (CPU) e **memória** (ROMs)!