

# COMO FAZER EMULADORES COM SDL & C



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**
  - 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
<code>SDL_Init(SDL_INIT_VIDEO)</code>	Inicializa o sistema de vídeo
<code>SDL_CreateWindow</code>	Cria uma janela 640x480 pixels
<code>SDL_CreateRenderer</code>	Cria um motor de desenho (hardware/software)
<code>SDL_SetRenderDrawColor</code>	Define a cor atual (R, G, B, Alpha)
<code>SDL_RenderClear</code>	Limpa o ecrã com a cor de fundo
<code>SDL_RenderFillRect</code>	Desenha um retângulo preenchido
<code>SDL_RenderPresent</code>	Mostra no ecrã o que foi desenhado
<code>SDL_Delay(3000)</code>	Aguarda 3000 ms (3 segundos)
<code>SDL_Destroy*</code>	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 ( $\uparrow \downarrow \leftarrow \rightarrow$ ).
- 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
<code>SDL_Init(SDL_INIT_AUDIO)</code>	Inicializa o subsistema de áudio
<code>SDL_AudioSpec</code>	Estrutura que define o formato do áudio
<code>SDL_OpenAudioDevice</code>	Abre um dispositivo de áudio com o formato pedido
<code>SDL_QueueAudio</code>	Envia um buffer com samples para serem reproduzidos
<code>sin()</code>	Função matemática para gerar uma onda senoidal
<code>SDL_PauseAudioDevice</code>	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
<code>main.c</code>	Ciclo principal ( <code>main_loop</code> ), coordena tudo
<code>cpu.c</code> / <code>cpu.h</code>	Simular a CPU (por agora só inicializar e resetar)
<code>video.c</code> / <code>video.h</code>	Desenhar e atualizar o ecrã
<code>input.c</code> / <code>input.h</code> ( <i>opcional</i> )	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 ( <code>fopen</code> , <code>fread</code> )
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)!