Apostila prática e didática

Como fazer jogos para Game Boy que cabem em 512 KB

Uma introdução clara a ROMs, tiles, bancos de memória, GB Studio, GBDK, assembly, gráficos, música e fluxo de trabalho.

Objetivo: entender como jogos homebrew de Game Boy são feitos hoje, por que eles cabem em arquivos pequenos e quais ferramentas usar para começar.

Nível: iniciante curioso com alguma familiaridade com jogos, lógica ou programação.

Material gerado em abril de 2026. Links e fontes no final.

```
160 x 144 pixels
tiles 8x8
ROM em bancos
som por canais
disciplina > força bruta
```

Sumário

1 O que você realmente comprou no itch.io

2 O Game Boy em cinco ideias simples

3 Por que 512 KB é suficiente?

4 ROM, RAM, VRAM e bancos de memória

5 Gráficos: tiles, mapas, sprites e metasprites

6 Cores e limitações visuais

7 Som e música sem MP3

8 Três jeitos de criar jogos: GB Studio, C e assembly

9 IDE, emuladores e depuração

10 Como organizar um projeto

11 Mini projeto 1: jogo simples no GB Studio

12 Mini projeto 2: primeiro programa em C com GBDK

13 Como desenhar assets que não explodem a ROM

14 Checklist para caber em 512 KB

15 Plano de estudo de 30 dias

16 Glossário e fontes

1. O que você realmente comprou no itch.io

Quando você compra jogos modernos feitos para Game Boy, normalmente recebe uma ROM, quase sempre com extensão `.gb` ou `.gbc`. Essa ROM é um arquivo que tenta representar o conteúdo de um cartucho: código, gráficos, música, mapas, textos e dados.

Essa ROM pode ser executada em três lugares:

Lugar	Como roda	Observação
Emulador	Você abre o arquivo `.gb`/`.gbc` num programa como SameBoy, BGB ou Emulicious.	Mais fácil para jogar e testar.
Hardware real	Você coloca a ROM em um flashcart compatível.	Mais fiel e mais caro.
Analogue Pocket/outros aparelhos	Alguns aceitam ROMs ou builds específicos.	Depende do aparelho e do formato.

O ponto que engana

Um arquivo de 512 KB parece minúsculo porque estamos acostumados a imagens de celular com vários megabytes. Mas jogo de Game Boy não guarda imagens grandes, vídeos ou áudio gravado. Ele guarda dados compactos e reutilizáveis.

É por isso que a pergunta certa não é “como coube tudo?”. A pergunta certa é: o que eles evitaram guardar?

2. O Game Boy em cinco ideias simples

Antes de falar de ferramentas, você precisa entender o alvo. O Game Boy é um computador simples, especializado em desenhar telas pequenas e tocar som simples.

Parte	O que faz	Tradução humana
CPU	Executa instruções do jogo.	O cérebro que roda a lógica.
ROM	Guarda o jogo no cartucho.	O livro de receitas.
RAM	Guarda dados temporários.	O caderno de rascunho.
VRAM	Guarda dados gráficos para a tela.	A mesa onde ficam os blocos visuais.
APU	Gera som por canais eletrônicos.	Um mini sintetizador, não um tocador de MP3.

A tela pequena ajuda muito

A tela do Game Boy tem 160 x 144 pixels. Isso é menor que muitos ícones de aplicativo atuais. Menos tela significa menos gráficos, menos detalhes e menos dados para armazenar.

Tamanho da tela

```
Game Boy: 160 x 144 = 23.040 pixels
Imagem 1080p: 1920 x 1080 = 2.073.600 pixels

Uma tela 1080p tem cerca de 90 vezes mais pixels.
```

3. Por que 512 KB é suficiente?

Primeiro, uma precisão chata mas importante: quando falamos “512 KB” nesse contexto, normalmente o correto técnico é 512 KiB, ou seja, 524.288 bytes. Na prática, todo mundo fala KB.

512 KiB é pouco para padrões modernos, mas é bastante quando o jogo:

- não usa vídeo;
- não usa áudio gravado;
- não guarda imagens grandes;
- reutiliza tiles;
- tem animações curtas;
- usa texto simples;
- divide a ROM em bancos.

O segredo não é mágica; é representação

Um cenário moderno poderia guardar uma imagem completa. Um jogo de Game Boy guarda:

- um conjunto pequeno de tiles;
- um mapa dizendo onde cada tile aparece;
- algumas informações de colisão;
- scripts simples de evento.

Imagem grande vs mapa de tiles

```
Jeito moderno burro para o Game Boy:
"guardar uma imagem inteira da fase"

Jeito Game Boy:
tile 01 = chão
tile 02 = parede
tile 03 = grama
tile 04 = pedra

mapa:
01 01 01 02 02
01 03 03 03 02
01 03 04 03 02
```

Estimativa didática

Um tile 8x8 em 2 bits por pixel ocupa 16 bytes. Isso acontece porque 8 x 8 = 64 pixels, cada pixel usa 2 bits, totalizando 128 bits, ou 16 bytes.

```
1 tile = 8 x 8 pixels
1 pixel = 2 bits
64 x 2 = 128 bits
128 / 8 = 16 bytes
```

Agora compare: 100 tiles únicos ocupam cerca de 1.600 bytes. Isso é ridiculamente pequeno perto de uma imagem PNG moderna.

4. ROM, RAM, VRAM e bancos de memória

O Game Boy tem um mapa de memória. Para um iniciante, isso assusta. Mas a ideia é simples: cada faixa de endereço tem uma função.

Mapa de memória simplificado

O problema

A CPU não enxerga 512 KiB inteiros de uma vez. Ela enxerga uma janela de ROM. Para acessar jogos maiores que 32 KiB, o cartucho usa um chip chamado MBC, ou Memory Bank Controller.

O que é banco de ROM?

Um banco é um pedaço de ROM. No Game Boy, cada banco de ROM tem 16 KiB. Em uma ROM de 512 KiB, você pode pensar assim:

```
512 KiB / 16 KiB = 32 bancos
```

O banco 0 normalmente fica fixo. Os outros bancos podem ser colocados na janela `4000-7FFF` quando o jogo precisa.

O que fica no banco fixo?

- rotinas essenciais;
- código que troca bancos;
- interrupções;
- funções que precisam existir sempre;
- parte da engine.

O que fica nos bancos trocáveis?

- dados de fases;
- gráficos específicos de uma área;
- músicas;
- textos grandes;
- código específico de minigames ou chefes.

5. Gráficos: tiles, mapas, sprites e metasprites

Essa é a parte mais importante para entender como os jogos cabem em pouco espaço.

Tile

Um tile é um bloquinho de 8x8 pixels. Ele é a peça básica do cenário.

> **Um tile 8x8**
>
> Visual meramente didático. Na ROM, isso vira bytes, não quadradinhos bonitos.

Tileset

Um tileset é o conjunto de tiles disponíveis. Por exemplo: chão, parede, canto, porta, água, sombra, pedra.

Tilemap

Um tilemap é uma matriz de números. Cada número aponta para um tile do tileset.

```
tileset:
00 = vazio
01 = chão
02 = parede
03 = porta
04 = grama

mapa:
02 02 02 02 02
02 01 01 03 02
02 01 04 01 02
02 02 02 02 02
```

O Game Boy tem mapas de tiles na VRAM. Cada entrada do mapa é basicamente um índice de tile. Isso é barato em memória.

Sprite

Sprite é um objeto móvel: personagem, inimigo, item, projétil. O hardware trata sprites separadamente do fundo.

Em muitas ferramentas, você desenha personagens em 16x16 pixels, mas internamente isso costuma ser montado com tiles menores.

Metasprite

Um sprite de hardware é pequeno. Um personagem maior pode ser composto por vários sprites pequenos. Esse conjunto é chamado de metasprite.

Metasprite

```
[sprite 8x8] [sprite 8x8]
[sprite 8x8] [sprite 8x8]

Juntos parecem um personagem 16x16.
```

6. Cores e limitações visuais

No Game Boy original, você trabalha com uma estética de poucas cores. No Game Boy Color, há mais possibilidades, mas ainda com limites fortes por tile e por paleta.

Por que isso é bom?

Porque limitações visuais forçam clareza. Um sprite de 16x16 não perdoa excesso. Se o jogador não entende a silhueta, o desenho falhou.

Elemento	Regra mental	Erro comum
Personagem	Silhueta legível antes de detalhes.	Enfiar rosto, roupa, arma e textura em 16x16.
Cenário	Repetição elegante de poucos tiles.	Criar tile único para cada pedaço da tela.
Paleta	Contraste antes de beleza.	Cores parecidas que viram lama no hardware.
Animação	Poucos frames fortes.	Frames demais para movimento pouco importante.

GB Studio e regras de assets

No GB Studio, sprites e backgrounds são importados como PNGs, mas precisam seguir regras de tamanho, paleta e quantidade de tiles. Backgrounds, por exemplo, são divididos em tiles 8x8, e a documentação recomenda usar editor de tilemap como Tiled para manter o grid correto.

- Desenhe no tamanho final. Não desenhe grande para reduzir depois.
- Use uma grade de 8x8.
- Repita tiles sempre que possível.
- Teste o sprite sobre fundos claros e escuros.
- Faça o personagem reconhecível em preto e branco antes de colorir.

Ferramentas comuns

Ferramenta	Uso	Comentário honesto
Aseprite	Pixel art, animação, spritesheets.	Excelente para sprites e animação. Pago, mas muito usado.
Tiled	Mapas baseados em tiles.	Ótimo para pensar em grid e repetição.
Piskel	Pixel art online.	Bom para começar sem comprar nada.
Photoshop/GIMP/Krita	Edição geral de imagem.	Funcionam, mas exigem disciplina de grid/paleta.

7. Som e música sem MP3

O som do Game Boy não é áudio gravado. Ele vem de canais sonoros simples controlados por registradores. Pense em um sintetizador pequeno.

Quatro canais

O Game Boy tem quatro unidades de geração sonora. Simplificando:

Canal	Serve para	Exemplo de uso
Canal 1	Onda de pulso com sweep.	Melodia principal, efeitos ascendentes.
Canal 2	Onda de pulso.	Melodia ou harmonia.
Canal 3	Wave channel.	Baixo, timbres customizados simples.
Canal 4	Ruído.	Bateria, explosões, passos, vento.

Como se faz música?

Há dois caminhos comuns no ambiente homebrew:

- Tracker: você compõe padrões de notas, parecido com uma planilha musical vertical.
- Editor integrado: no GB Studio moderno, arquivos UGE podem ser editados em um editor próprio.

Ferramentas citadas frequentemente incluem hUGETracker, OpenMPT, MilkyTracker e o editor de música do GB Studio.

Regra prática

Faça música pensando em função, não em luxo. Uma tela de título precisa de identidade. Uma fase precisa de ritmo. Um menu precisa de feedback. Nem tudo precisa de música longa.

8. Três jeitos de criar jogos: GB Studio, C e assembly

Hoje há três caminhos principais. Nenhum é “o certo” para todo mundo. A escolha depende do seu objetivo.

Caminho	Linguagem	Vantagem	Desvantagem	Para quem
GB Studio	Visual scripting + eventos	Começa rápido e exporta ROM.	Menos controle profundo.	Iniciantes, narrativos, puzzles, aventuras.
GBDK-2020	C + opcional assembly	Controle bom sem escrever tudo em assembly.	Você precisa programar e entender limites.	Programadores que querem fazer engine/jogo próprio.
RGBDS	Assembly SM83	Controle máximo.	Curva de aprendizado cruel.	Hardcore, demos, otimização, estudo profundo.

GB Studio

É um criador visual de jogos de Game Boy. Você cria cenas, atores, triggers, diálogos, eventos e exporta uma ROM. É o caminho mais curto para fazer algo jogável.

GBDK-2020

É um kit de desenvolvimento que permite escrever jogos em C. Ele inclui compilador, bibliotecas e ferramentas. É a ponte entre “quero fazer jogo” e “quero controlar a máquina”.

```
#include <gb/gb.h>

void main(void) {
    while(1) {
        if (joypad() & J_A) {
            // apertou A
        }
        wait_vbl_done();
    }
}
```

O código acima não é um jogo. É o esqueleto mental: loop infinito, leitura de controle e espera pelo VBlank.

RGBDS

RGBDS é um pacote de assembler/linker para Game Boy e Game Boy Color. Você escreve instruções de baixo nível. É poderoso, mas exige entender CPU, memória, registradores, timing e formato da ROM.

```
SECTION "Header", ROM0[$100]
    jp EntryPoint

SECTION "Game code", ROM0[$150]
EntryPoint:
    ; inicialização aqui
.loop
    jr .loop
```

9. IDE, emuladores e depuração

Não existe uma IDE oficial única. O fluxo moderno parece mais com desenvolvimento web/embedded do que com Unity.

Possíveis setups

Caminho	Editor/IDE	Build	Teste
GB Studio	GB Studio	Botão Play/Export ROM	Emulador interno ou externo.
GBDK	VS Code, Vim, Sublime etc.	Makefile/terminal	SameBoy, BGB, Emulicious.
RGBDS	Editor + terminal	rgbasm/rgbgfx/rgblink/rgbfix	Emulador com debugger.

Emuladores úteis

Ferramenta	Por que usar	Melhor para
SameBoy	Emulador moderno, preciso, multiplataforma.	Jogar e testar compatibilidade.
BGB	Emulador/debugger clássico no Windows/Wine.	Depurar ROMs e timing.
Emulicious	Debugger com viewer de tile, sprite, memória, profiler.	Desenvolvimento sério e análise visual.

O que é depurar no Game Boy?

Depurar é olhar dentro da máquina enquanto o jogo roda. Você pode ver:

- memória;
- registradores da CPU;
- tiles carregados na VRAM;
- sprites em OAM;
- paletas;
- execução linha por linha;
- uso de bancos de ROM.

10. Como organizar um projeto

Projeto em GB Studio

```
meu-jogo-gb/
  assets/
    backgrounds/
    sprites/
    music/
    ui/
  build/
    rom/
      game.gb
    web/
  project.gbsproj
```

Você coloca arquivos nos diretórios certos e o GB Studio detecta. A lógica do jogo fica em cenas, atores e scripts visuais.

Projeto em GBDK

```
meu-jogo-c/
  src/
    main.c
    player.c
    enemy.c
    collision.c
  res/
    player.png
    tileset.png
    map.tmx
  obj/
  build/
    meu_jogo.gb
  Makefile
```

Você escreve código em C, converte assets para dados do Game Boy e compila tudo para uma ROM.

Projeto em RGBDS

```
meu-jogo-asm/
  src/
    main.asm
    init.asm
    input.asm
    graphics.asm
  gfx/
    tiles.png
  maps/
  build/
    jogo.gb
  Makefile
```

Aqui você tem controle total. Também tem responsabilidade total. Esqueceu de configurar o header direito? A ROM pode nem iniciar.

- Separe código de assets.
- Separe assets por fase/tela quando possível.
- Nomeie tudo de forma óbvia.
- Tenha uma pasta de builds.
- Teste a ROM frequentemente.
- Use controle de versão, nem que seja Git básico.

11. Mini projeto 1: jogo simples no GB Studio

Objetivo: criar uma ROM mínima com uma sala, um personagem, um NPC e um diálogo.

Passo 1 - Crie o projeto

1. Abra o GB Studio.
2. Crie um novo projeto.
3. Escolha um template simples.
4. Dê um nome curto ao jogo.

Passo 2 - Faça uma cena pequena

Crie um background de 160x144 ou use um asset simples. A tela inicial do Game Boy é pequena; não comece com mapa gigante.

Passo 3 - Adicione o player

Use um sprite 16x16. Comece com boneco feio mesmo. O objetivo inicial não é arte bonita; é entender o fluxo.

Passo 4 - Adicione um NPC

Coloque um ator na cena. Adicione evento de interação:

```
Quando o jogador apertar A perto do NPC:
  mostrar diálogo: "Bem-vindo ao meu primeiro jogo de Game Boy!"
```

Passo 5 - Exporte a ROM

No GB Studio, use Export ROM. O resultado deve aparecer como um arquivo `game.gb`. Abra no emulador.

12. Mini projeto 2: primeiro programa em C com GBDK

Objetivo: entender o fluxo de C para ROM.

Passo 1 - Instalar GBDK-2020

Baixe o GBDK-2020, escolha um exemplo ou template e compile com `make`. A documentação recomenda começar por templates incluídos nos exemplos.

Passo 2 - Entender o loop

Um jogo roda em loop. Ele lê entrada, atualiza estado, desenha ou agenda desenho, espera o momento certo e repete.

```
#include <gb/gb.h>
#include <stdint.h>

uint8_t x = 80;
uint8_t y = 72;

void main(void) {
    SHOW_SPRITES;

    while(1) {
        uint8_t keys = joypad();

        if(keys & J_LEFT)  x--;
        if(keys & J_RIGHT) x++;
        if(keys & J_UP)    y--;
        if(keys & J_DOWN)  y++;

        move_sprite(0, x, y);
        wait_vbl_done();
    }
}
```

O que esse código ensina?

- `joypad()` lê botões.
- x e y guardam posição.
- `move_sprite()` move um sprite.
- `wait_vbl_done()` espera o VBlank, evitando bagunça visual.

Passo 3 - Carregar gráficos

Em GBDK, gráficos normalmente são convertidos para dados C/assembly por ferramentas como `png2asset`. Você não usa PNG diretamente na ROM final. O PNG vira arrays de bytes que o Game Boy entende.

Passo 4 - Testar e medir

Depois de compilar, abra a ROM em SameBoy/BGB/Emulicious. Se usar debugger, veja se os tiles aparecem na VRAM.

13. Como desenhar assets que não explodem a ROM

Aqui está o ponto onde muita gente iniciante se sabota: tenta desenhar bonito demais, cedo demais, com tiles demais.

Comece por silhueta

Um sprite pequeno precisa ser reconhecido de longe. Antes de sombrear, pergunte:

- dá para saber onde é a cabeça?
- dá para saber para onde ele está olhando?
- dá para diferenciar player de NPC?
- dá para ver o inimigo no fundo?

Use uma folha de tiles limitada

Para uma floresta, por exemplo, você pode começar com:

Tile	Função
Grama plana	Preenche áreas grandes.
Grama com textura	Variedade visual.
Canto de árvore	Forma obstáculos.
Tronco	Define bordas.
Sombra	Dá profundidade.
Pedra	Detalhe reutilizável.

Com poucos tiles bem desenhados, você monta várias telas.

Faça primeiro em feio jogável

Pipeline brutalmente eficiente:

1. Tiles provisórios.
2. Mapa funcional.
3. Colisão funcionando.
4. Interação funcionando.
5. Aí sim melhorar arte.

Truques para economizar

- Espelhe sprites quando possível.
- Use animação de 2 frames para movimento simples.
- Reutilize tiles de UI em menus diferentes.
- Tenha tiles “coringa” para sombra, borda e textura.

- Use texto para sugerir mundo em vez de desenhar tudo.

14. Checklist para caber em 512 KB

- O jogo tem escopo pequeno?
- Cada fase reaproveita tiles?
- Cada inimigo realmente precisa existir?
- A história pode ser contada com menos texto?
- As animações têm poucos frames?

- Backgrounds usam tiles repetidos?
- Você está respeitando grade de 8x8?
- Sprites são legíveis em 16x16?
- Paletas têm contraste suficiente?
- Você removeu tiles quase iguais?

- Rotinas comuns estão centralizadas?
- Dados grandes estão fora do banco fixo?
- Você sabe quais bancos contêm quais assets?
- Você testou em emulador preciso?
- Você mediu tamanho da ROM após cada etapa grande?

- Músicas são curtas e reaproveitáveis?
- Efeitos sonoros não brigam com a música?
- Você evita tentar simular áudio moderno?

Orçamento mental de memória

Não use estes números como regra exata; use como forma de pensar:

Categoria	Como economizar
Código	Evite duplicação, use funções comuns.
Gráficos	Tiles reutilizados, mapas por índices.
Música	Sequências de notas, temas curtos.
Texto	Frases curtas e fonte compacta.
Mapas	Tilemaps e compressão quando necessário.

15. Plano de estudo de 30 dias

Este plano é para sair da curiosidade e chegar numa ROM pequena funcionando.

Dias	Meta	Entregável
1-3	Entender ROM, emulador, tela, tiles.	Abrir ROMs em emulador e observar.
4-7	Aprender GB Studio básico.	Uma sala com player e NPC.
8-10	Aprender backgrounds e sprites.	Sprite próprio e sala própria.
11-14	Eventos, portas, itens.	Mini aventura com 3 salas.
15-17	Música e efeitos.	Uma trilha curta e 2 efeitos.
18-20	Polimento e build.	ROM jogável do começo ao fim.
21-24	Introdução a GBDK.	Compilar um exemplo em C.
25-27	Movimento e sprites em C.	Quadrado/personagem movendo na tela.
28-30	Comparar caminhos.	Decidir: continuar em GB Studio ou migrar para C.

Critério de sucesso

Ao fim de 30 dias, sucesso não é “ter um jogo comercial”. Sucesso é ter:

- uma ROM que abre no emulador;
- um loop jogável;
- pelo menos uma interação;
- um asset próprio;
- uma noção real do que é difícil.

16. Glossário e fontes

Glossário rápido

Termo	Significado simples
ROM	Arquivo/conteúdo do cartucho com código e dados do jogo.
RAM	Memória temporária para estado do jogo.
VRAM	Memória de vídeo usada para tiles e mapas.
Tile	Bloco gráfico de 8x8 pixels.
Tilemap	Matriz de índices que monta o cenário com tiles.
Sprite	Objeto gráfico móvel, como personagem/inimigo.
Metasprite	Conjunto de sprites menores formando um personagem maior.
MBC	Chip/controlador que permite trocar bancos de ROM/RAM no cartucho.
Bank switching	Troca de banco de ROM visível em uma região da memória.
VBlank	Intervalo seguro entre frames para atualizar certos dados gráficos.
Tracker	Ferramenta de composição musical baseada em padrões de notas.
GBDK	Kit para desenvolver jogos/programas de Game Boy em C/assembly.
RGBDS	Toolchain para assembly de Game Boy/Game Boy Color.

Fontes principais e links úteis

Pan Docs - documentação técnica do Game Boy
https://gbdev.io/pandocs/

Usado como referência para mapa de memória, gráficos, tiles, tilemaps, áudio, header de cartucho e MBCs.

Pan Docs - Memory Map
https://gbdev.io/pandocs/Memory_Map.html

Pan Docs - Graphics
https://gbdev.io/pandocs/Graphics.html

Pan Docs - Tile Data
https://gbdev.io/pandocs/Tile_Data.html

Pan Docs - Tile Maps
https://gbdev.io/pandocs/Tile_Maps

Pan Docs - Audio
https://gbdev.io/pandocs/Audio.html

Pan Docs - Cartridge Header
https://gbdev.io/pandocs/The_Cartridge_Header.html

Pan Docs - MBC1
https://gbdev.io/pandocs/MBC1

GB Studio - documentação oficial
https://www.gbstudio.dev/docs/

Referências específicas: introdução, build/export ROM, assets, sprites, backgrounds e música.

- Introdução
- Build / Export ROM
- Assets
- Sprites
- Backgrounds
- Music

GBDK-2020 - documentação oficial
https://gbdk.org/docs/api/index.html

- Getting Started
- ROM/SRAM Banking and MBCs
- Metasprites / png2asset

RGBDS - assembler/linker para Game Boy
https://rgbds.gbdev.io/

Emuladores e debug

- SameBoy: https://sameboy.github.io/
- BGB: https://bgb.bircd.org/
- Emulicious: https://emulicious.net/

Ferramentas de arte/mapa

- Aseprite: https://www.aseprite.com/
- Tiled Map Editor: https://www.mapeditor.org/

Comunidade e lista de recursos
https://gbdev.io/resources.html

Fechamento

Se você entendeu uma coisa, guarde isto: um jogo de Game Boy pequeno funciona porque quase tudo é representação eficiente. O cenário é mapa de tiles. A música é comando de sintetizador. A ROM é dividida em bancos. O design aceita limites em vez de brigar com eles.

O caminho mais inteligente é começar simples, exportar uma ROM cedo, testar em emulador e só então aumentar ambição. O Game Boy recompensa disciplina. E pune fantasia de grandeza.