Seminário
Wolfenstein 3D

apresentação sobre as páginas :78 à 90

Alunos: **Teo Giambarba** **Arthur Solar** **Gabriel Henrique** **Arthur Prates**
Rafael Fassina

seção 3.2 programming

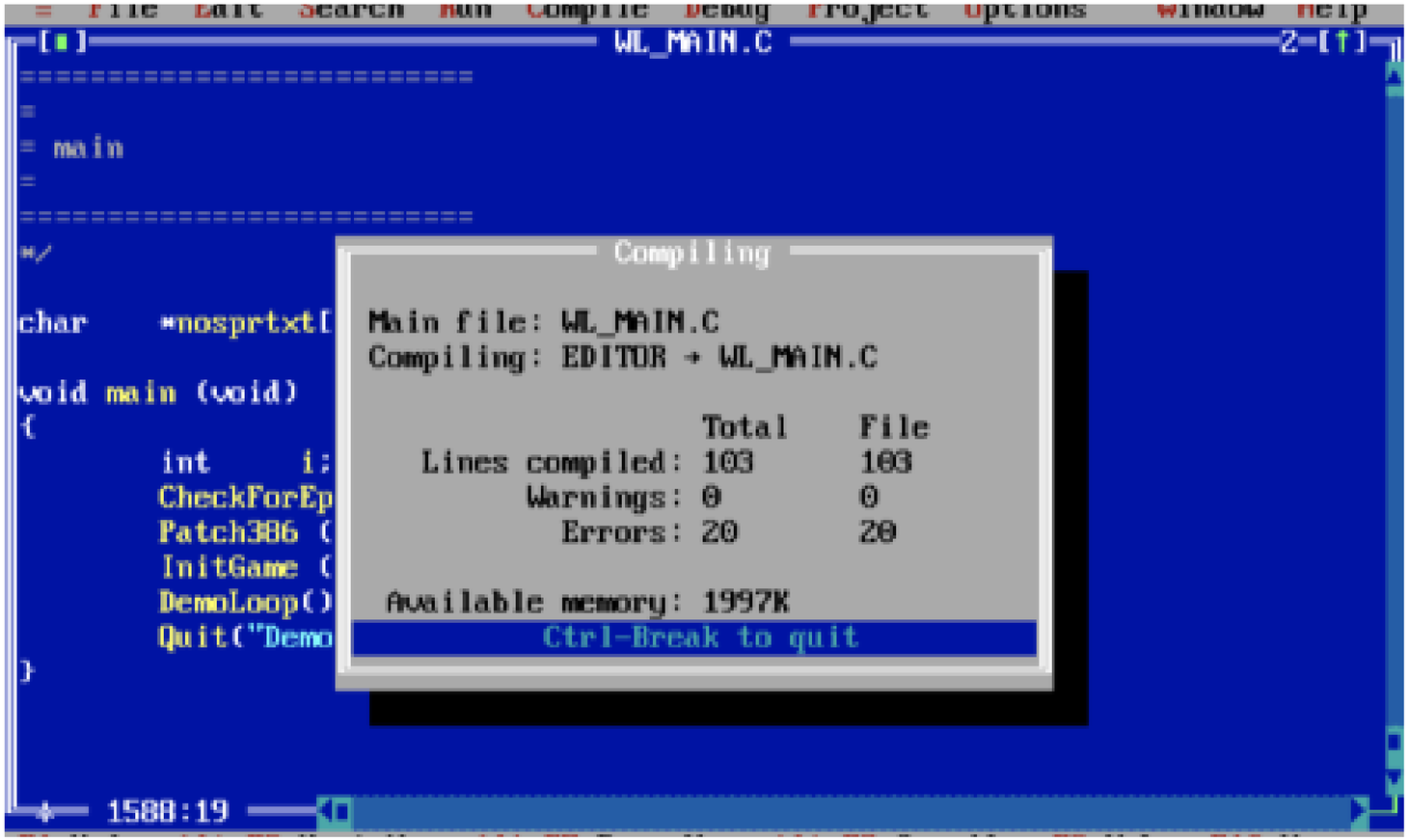
Falando sobre o processo de programação, essa foi feita usando **Borland C++** (IDE e compilador de C/C#) em **C**. Padrão para o modo VGA modo 3 com tela de 80 caracteres de espessura e 25 de altura.

Essa IDE era um pacote **tudo em um** contendo:

- BC.EXE (Executavel da IDE): Executável principal que abria a IDE, permitindo edição de código em múltiplas janelas (com instabilidades) também com destaques na sintaxe dos códigos .
- BCC.EXE (Executavel do Compilador): Compilador de linha de comando que pegava o código fonte e traduzia para código de maquina.
- TLINK.EXE(Executavel Turbo Linker) : Unia o arquivo em código de mapa gerado pelo BCC.EXE e as bibliotecas necessárias em um executável final.

Seção 3.2 Programação: IDE

O sistema funcionava em um DOS (Sistema operacional de disco) o que envolve uso de comandos digitados, porem depois de executar a IDE, você não precisava fazer isso.



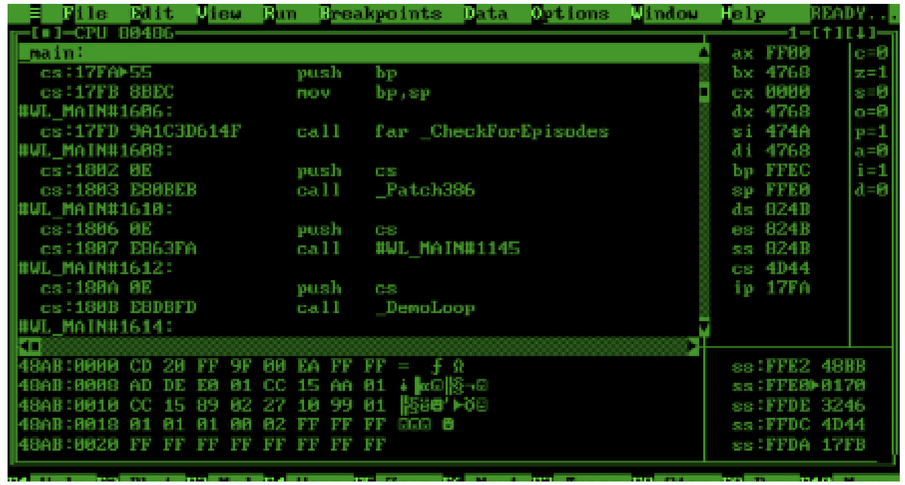
A IDE permitia também:

- project: Criação de projeto com vários arquivos
- build: Compilar tudo
- run: Executar o programa
- Debug

Seção 3.2 Programação: Display

Eles usavam pequenos monitores **CRT**(Monitor de Tubo) , para lidar com esse problema, alguns desenvolvedores usavam 2 monitores ligados em 2 placas de video diferentes no mesmo computador (era possível pois ambas usavam endereços diferentes na memória RAM)

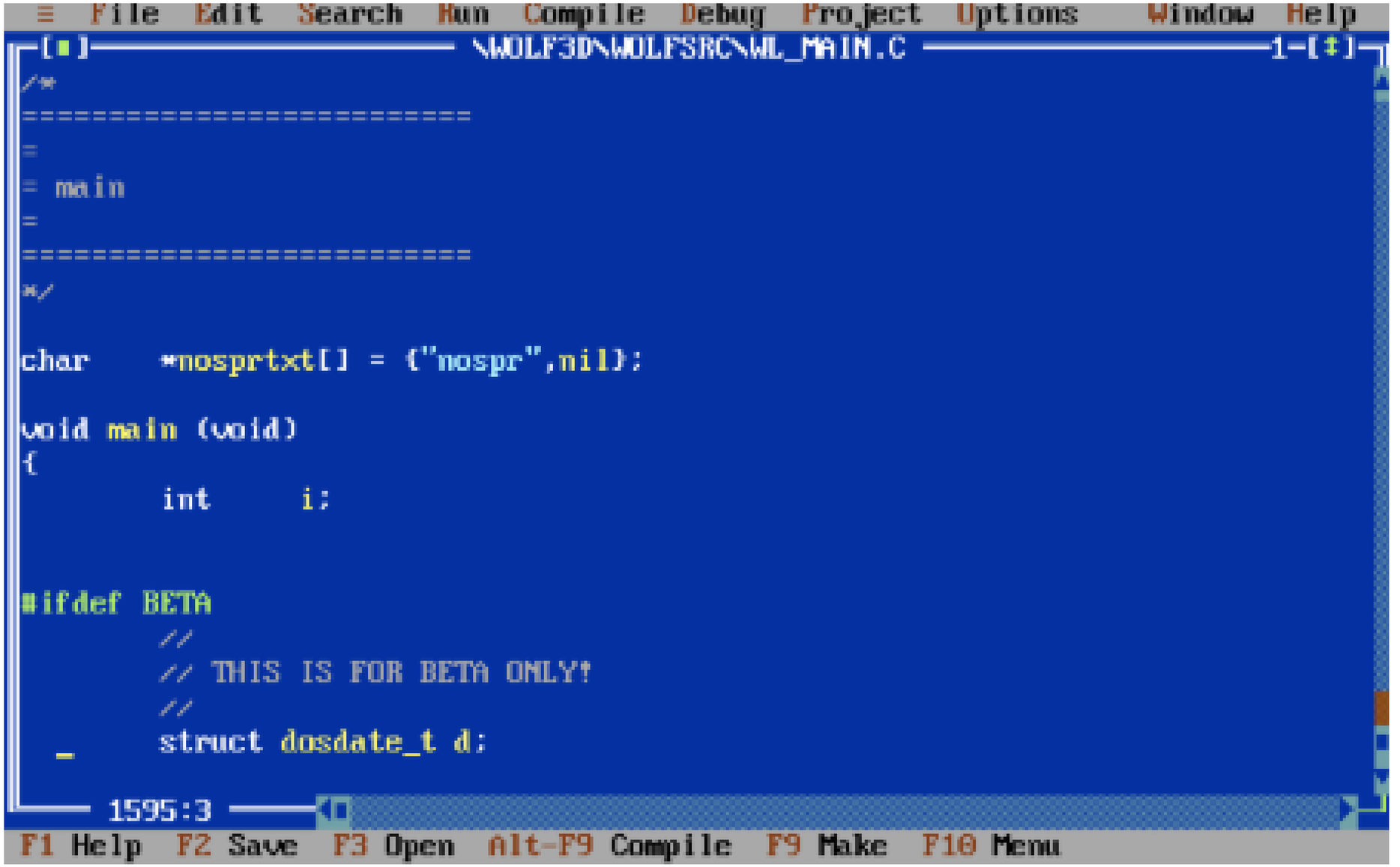
- **Monitor placa VGA:** Rodava o jogo (região 0xA0000)
- **Monitor para placa MDA(suporta apenas texto preto, branco e verde):** Mostrava a IDE/editor e os outros recursos escritos.



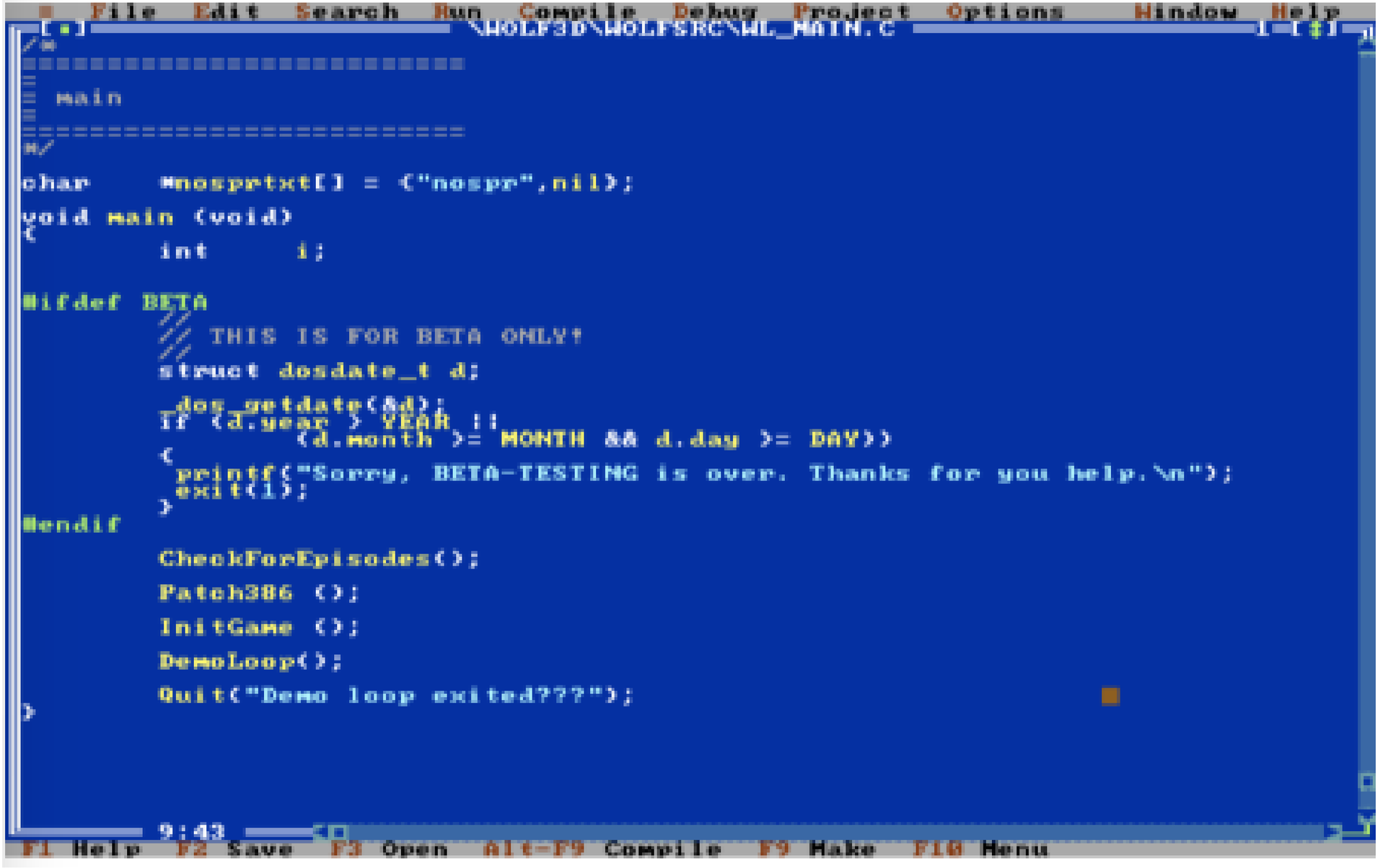
Seção 3.2 Programação: Display

A tela era organizada como uma grade, onde normalmente cabem 80 colunas (cada coluna é um caractere) e 25 linhas.
Alta resolução (50x80) modo texto: Era outra forma de lidar com o baixo tamanho do display. Dobrando a quantidade de linhas .
Essa escolha resulta numa troca: Mais informações na tela porem menos legibilidade e vice versa.

Seção 3.2 Programação: Display



Seção 3.2 Programação: Display



Seção 3.3 Recursos Gráficos

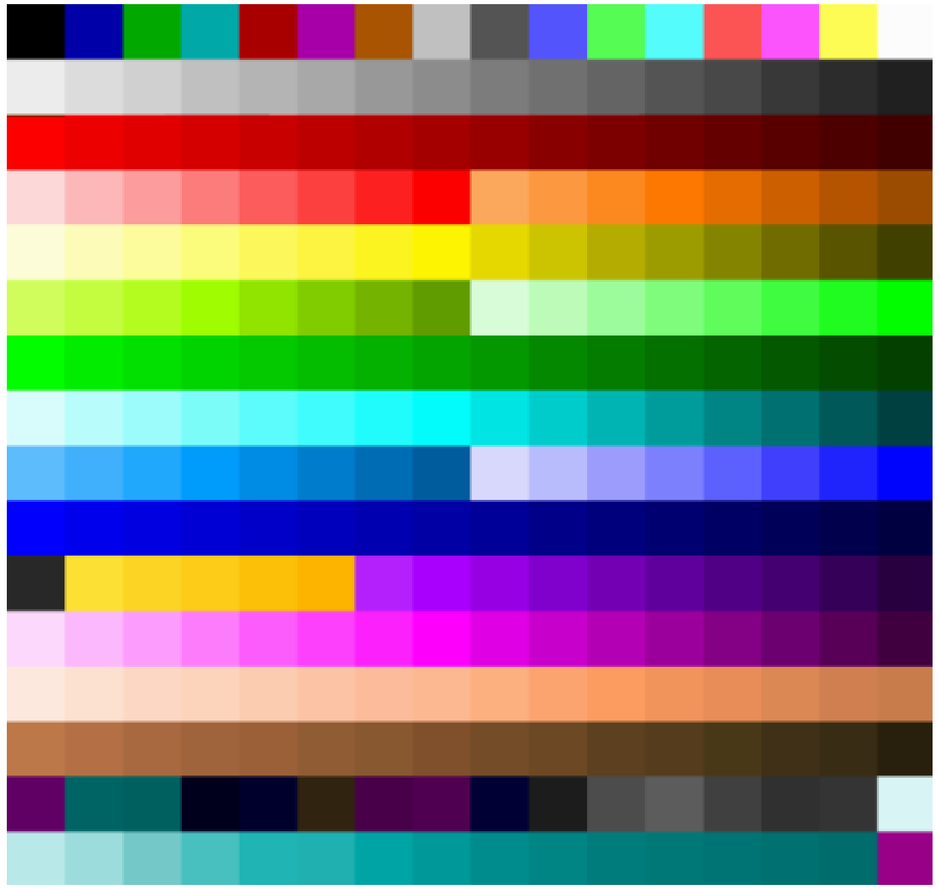
Os recursos gráficos foram feitos usando Deluxe Paint e salvos no formato ILBM
A VGA (sistema usado) era baseado em paletas, assim cada cor em um pixel era um índice que apontava para uma cor na paleta escolhida pelo artista com 256 cores para o jogo inteiro .
Ou seja cada cor na paleta podia ser escolhida via RGB, depois disso, somente aquelas cores poderiam ser usadas para o jogo.

Seção 3.3 Recursos Gráficos

Todos os recursos gráficos (sprites, texturas, menus, etc) foram desenhados a mão com mouse (Eles não tinham ferramentas de digitalização de imagens) . Um feito muito difícil considerando que ele tinha que desenhar pixel a pixel na resolução exata que o jogo usava.
Assim quando o Framebuffer (imagem bruta na memória antes de aparecer na tela) fossem ampliada (devido a diferença de resolução entre o jogo e o monitor), nada ficasse estranho.

Seção 3.3 Recursos Gráficos
Tela do Deluxe Paint

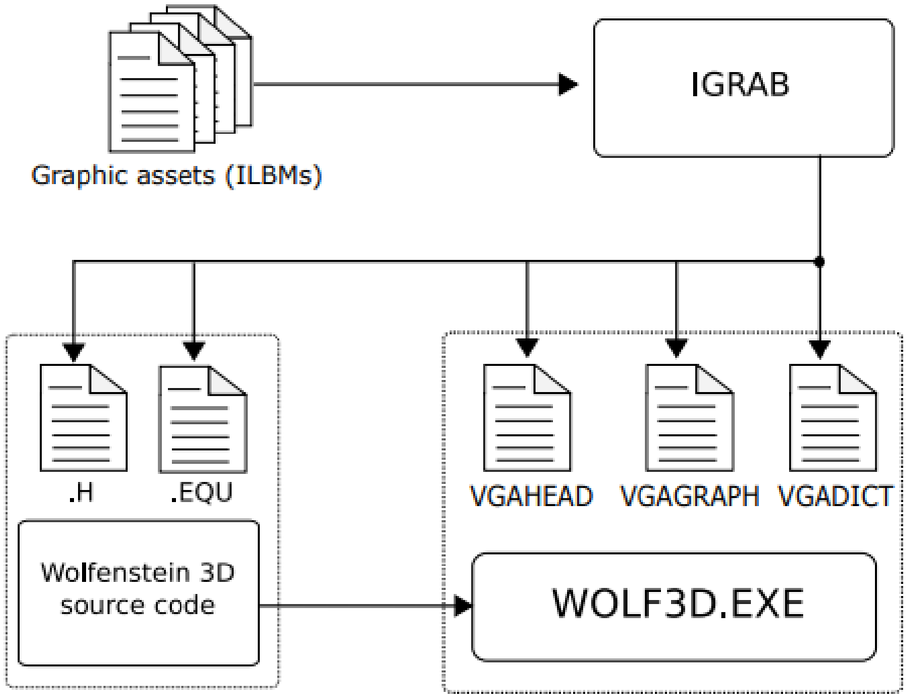
Seção 3.3 Recursos Gráficos



As coordenadas eram dadas pelos números hexadecimais:

- Horizontal (colunas): 0x00 a 0x0F
- Vertical(linhas): 0x00 a 0xF0 (pula de 16 em 16)
- 0xFF (ultimo da ultima linha): escolhida como transparentnne, assim ela não é renderizada (desenhada na tela) permitindo sprites de “fundo invisivel”

Seção 3.4 Fluxo dos Recursos Gráficos



A Ferramenta IGRAB e o Empacotamento

- As artes originais eram desenhadas no Deluxe Paint e salvas no formato proprietário ILBM.
- Para converter isso para o jogo, a id Software utilizou uma ferramenta chamada IGRAB.
- O IGRAB "empacotava" todas as imagens em arquivos de arquivo (ex: VGAGRAPH para menus e VSWAP para o 3D).
- O arquivo VGADICT armazenava a "árvore de Huffman", essencial para descomprimir as imagens.

Seção 3.4 Fluxo dos Recursos Gráficos

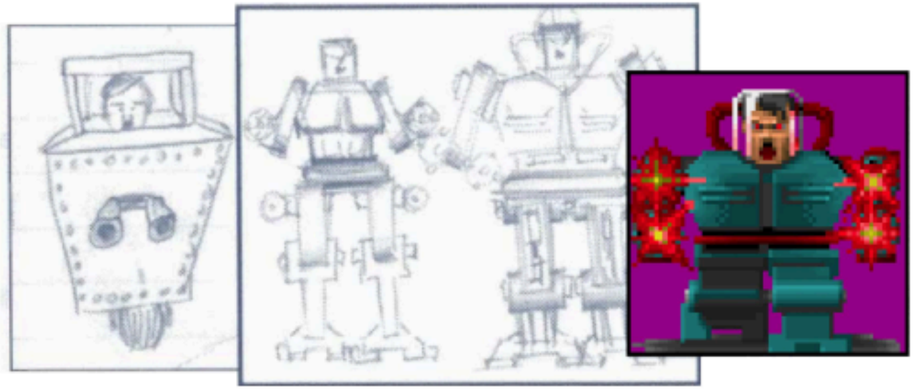
Integrando Arte e Código

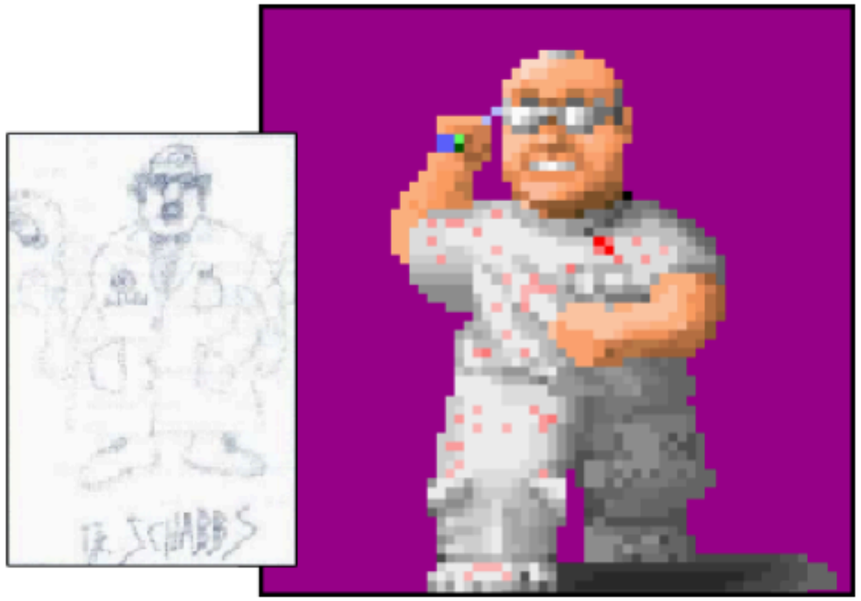
```
//////////////////////////////////////
//
// Graphics .H file for .WL1
// IGRAB-ed on Sun May 03 01:19:32 1992
//
//////////////////////////////////////

typedef enum {
    // Lump Start
    H_BJPIC=3,
    H_CASTLEPIC,                // 4
    H_KEYBOARDPIC,              // 5
    H_JOYPIC,                   // 6
    H_HEALPIC,                  // 7
    H_TREASUREPIC,              // 8
    H_GUNPIC,                   // 9
    H_KEYPIC,                   // 10
    H_BLAZEPIC,                 // 11
    H_WEAPON1234PIC,            // 12
    H_WOLFLOGOPIC,              // 13
    ...
    PAUSEDPIC,                  // 140
    GETPSYCHEDPIC,              // 141
```

- O IGRAB não apenas empacotava imagens, mas gerava arquivos de cabeçalho (.H) para a linguagem C.
- Cada imagem recebia um identificador único através de um enum (ex: GETPSYCHEDPIC ou H_KEYPIC).
- Grande Vantagem: Isso criava uma camada de indireção. Os artistas podiam adicionar ou reordenar as imagens, e o IGRAB atualizava os IDs automaticamente, sem que os programadores precisassem alterar o código-fonte do motor.
- O motor de jogo usava comandos como LatchDrawPic() passando esses IDs para desenhar na tela.

Seção 3.4 Fluxo dos Recursos Gráficos





O Processo Criativo: Rascunhos e Pixel Art

- O processo começava de forma muito analógica.
- Tom Hall (Diretor Criativo) tinha as ideias para as telas e fornecia rascunhos feitos a lápis no papel.
- Adrian Carmack pegava esses rascunhos e os transformava na pixel art final usando o mouse.

Seção 3.4 Fluxo dos Recursos Gráficos

Curiosidade: O Problema do Lançamento do Código

- Anos mais tarde, a id Software liberou o código-fonte do jogo ao público.
- No entanto, ocorreu um erro: os arquivos de cabeçalho (.h) liberados pertenciam à sequência do jogo, Spear of Destiny, e não à versão original Shareware.
- Resultado: Os IDs das imagens não batiam. Quem tentava compilar o código na época se deparava com uma "bagunça gráfica" (sopa de pixels) porque o jogo tentava puxar as imagens erradas.