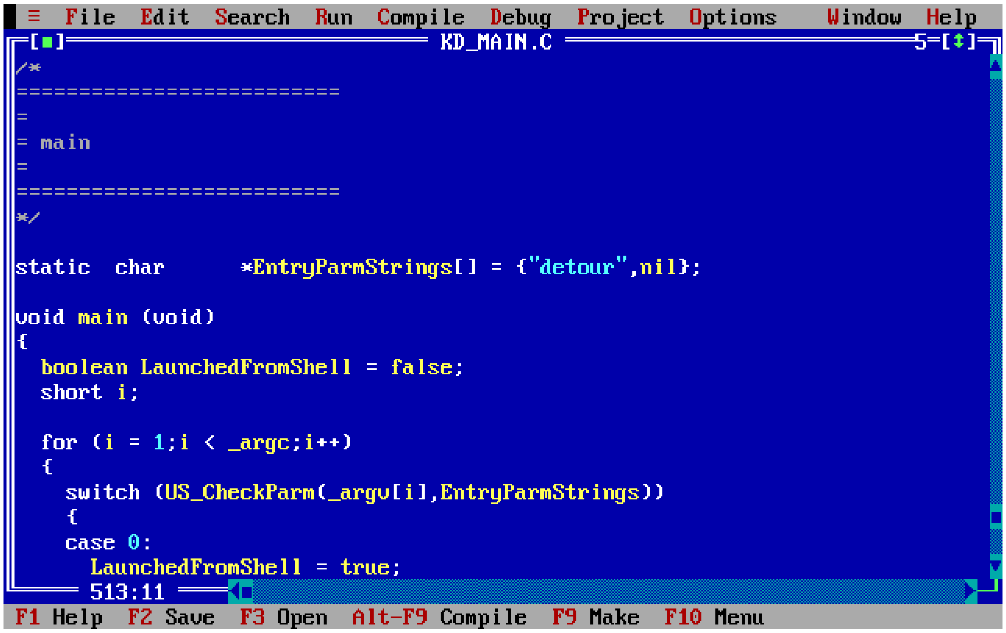
0.1 Programming

Development was done with Borland C++ 3.1 (but the language used was C) which by default ran in EGA mode 3 offering a screen 80 characters wide and 25 characters tall.

John Carmack took care of the runtime code. John Romero programmed many of the tools (TED5 map editor, IGRAB asset packer, MUSE sound packer). Jason Blochowiak wrote important subsystems of the game (Input manager, Sound manager, User manager).


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] KD_MAIN.C 5=[↕]
513:11
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

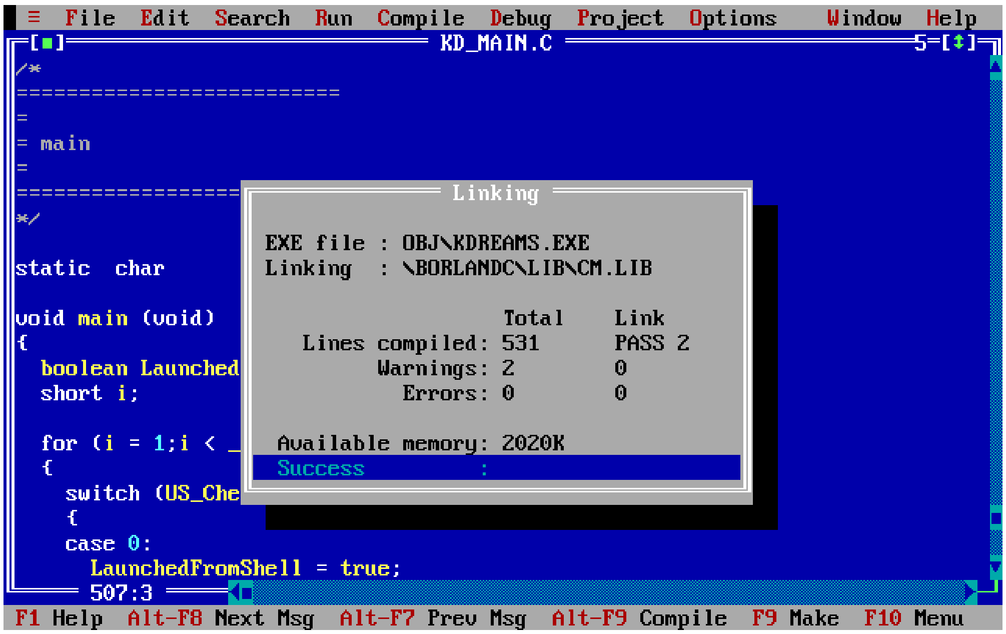
```
/*
==========================
=
= main
=
==========================
*/

static  char      *EntryParmStrings[] = {"detour",nil};

void main (void)
{
  boolean LaunchedFromShell = false;
  short i;

  for (i = 1;i < _argc;i++)
  {
    switch (US_CheckParm(_argv[i],EntryParmStrings))
    {
    case 0:
      LaunchedFromShell = true;
```

Figure 1: Borland C++ 3.1 editor

Borland's solution was an all-in-one package. The IDE, `BC.EXE`, despite some instabilities allowed crude multi-windows code editing with pleasant syntax highlights. The compiler and linker were also part of the package under `BCC.EXE` and `TLINK.EXE`[1].

[1]Source: Borland C++ 3.1 User Guide.

There was no need to enter command-line mode however. The IDE allowed to create a project, build, run and debug.

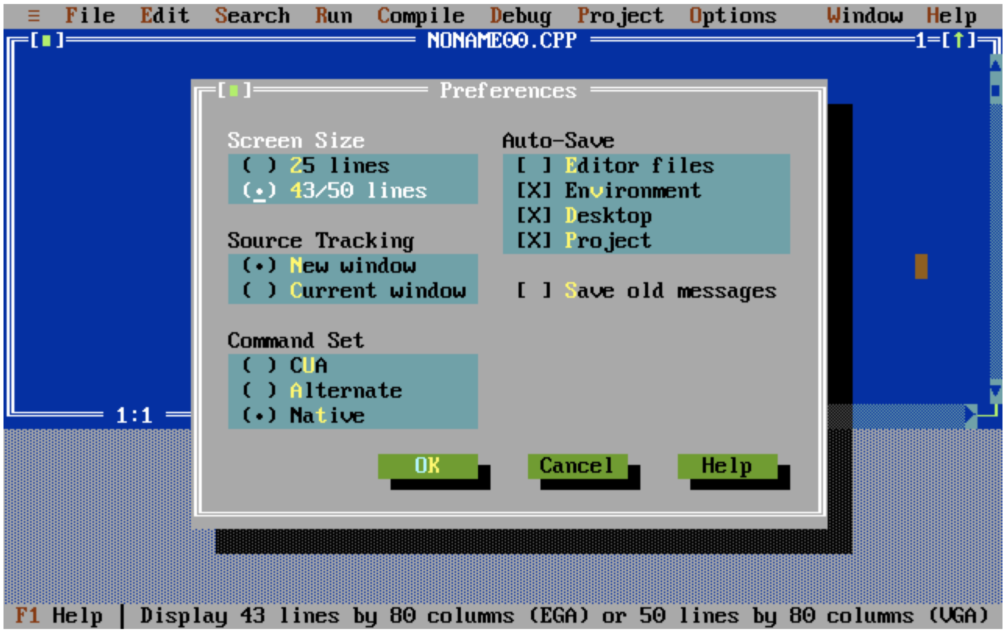

≡ File Edit Search Run Compile Debug Project Options Window Help
KD_MAIN.C
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
/*
==========================
=
= main
=
=================
*/

static  char

void main (void)
{
  boolean Launched
  short i;

  for (i = 1;i < _
  {
    switch (US_Che
    {
    case 0:
      LaunchedFromShell = true;
507:3
```

```
                    Linking

EXE file : OBJ\KDREAMS.EXE
Linking  : \BORLANDC\LIB\CM.LIB

                   Total     Link
   Lines compiled: 531       PASS 2
         Warnings: 2         0
           Errors: 0         0

  Available memory: 2020K
  Success          :
```

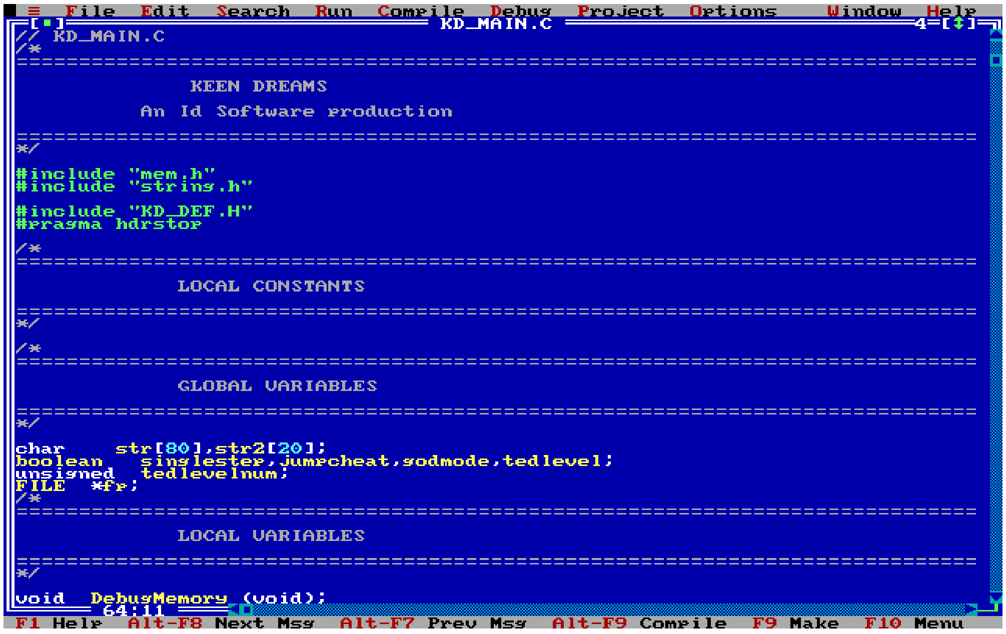
Figure 2: Compiling Keen Dreams with Borland C++ 3.1

Another way to improve screen real estate was to use "high resolution" 50x80 text mode.


≡ File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP
1
1:1
OK
Cancel
Help

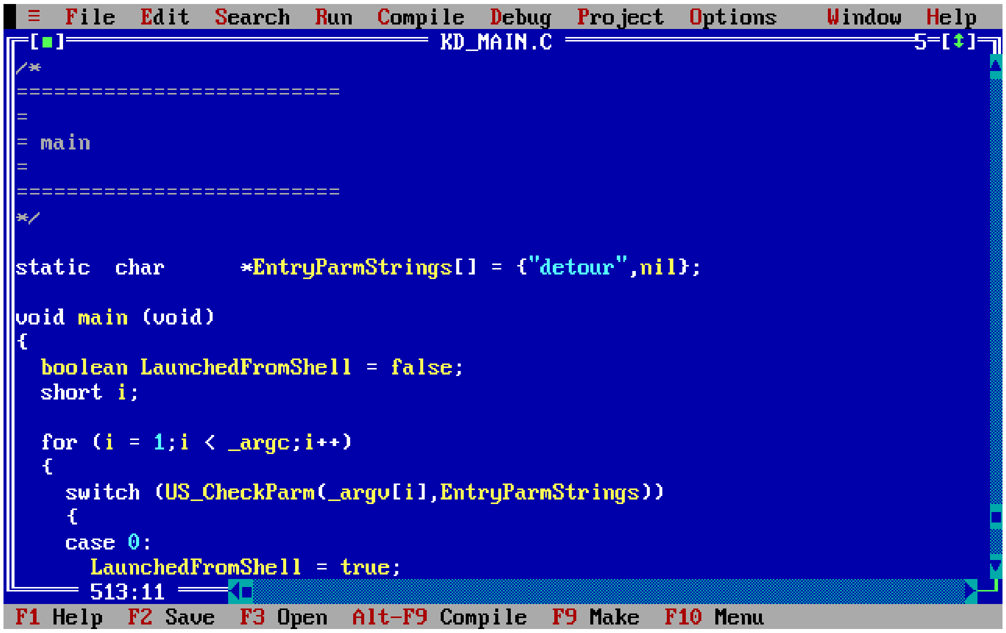
```
Preferences

Screen Size                 Auto-Save
 ( ) 25 lines                [ ] Editor files
 (•) 43/50 lines             [X] Environment
                             [X] Desktop
Source Tracking              [X] Project
 (•) New window
 ( ) Current window          [ ] Save old messages

Command Set
 ( ) CUA
 ( ) Alternate
 (•) Native

F1 Help | Display 43 lines by 80 columns (EGA) or 50 lines by 80 columns (VGA)
```

The comments still fit perfectly on screen since only the vertical resolution is doubled.

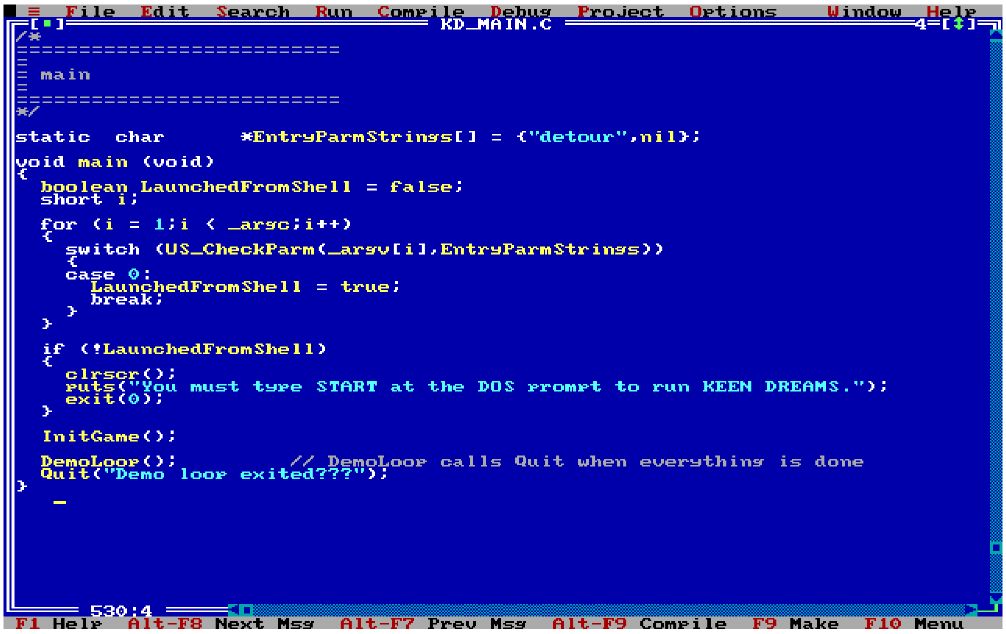

≡ File Edit Search Run Compile Debug Project Options Window Help
KD_MAIN.C
4
64:11
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
// KD_MAIN.C
/*
=============================================================================

                          KEEN DREAMS

                  An Id Software production

=============================================================================
*/

#include "mem.h"
#include "string.h"

#include "KD_DEF.H"
#pragma hdrstop

/*
=============================================================================

                         LOCAL CONSTANTS

=============================================================================
*/

/*
=============================================================================

                         GLOBAL VARIABLES

=============================================================================
*/

char       str[80],str2[20];
boolean     singlestep,jumpcheat,godmode,tedlevel;
unsigned    tedlevelnum;
FILE    *fp;
/*
=============================================================================

                         LOCAL VARIABLES

=============================================================================
*/

void    DebugMemory (void);
```

The file KD_MAIN.C opened in both modes demonstrates the readability/visibility trade-off.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] KD_MAIN.C 5 [↕]
513:11
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
/*
==========================
=
= main
=
==========================
*/

static  char      *EntryParmStrings[] = {"detour",nil};

void main (void)
{
  boolean LaunchedFromShell = false;
  short i;

  for (i = 1;i < _argc;i++)
  {
    switch (US_CheckParm(_argv[i],EntryParmStrings))
    {
    case 0:
      LaunchedFromShell = true;
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] KD_MAIN.C 4 [↕]
530:4
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
/*
==========================
= main
=
==========================
*/
static  char      *EntryParmStrings[] = {"detour",nil};
void main (void)
{
  boolean LaunchedFromShell = false;
  short i;
  for (i = 1;i < _argc;i++)
  {
    switch (US_CheckParm(_argv[i],EntryParmStrings))
    {
    case 0:
      LaunchedFromShell = true;
      break;
    }
  }
  if (!LaunchedFromShell)
  {
    clrscr();
    puts("You must type START at the DOS prompt to run KEEN DREAMS.");
    exit(0);
  }
  InitGame();
  DemoLoop();             // DemoLoop calls Quit when everything is done
  Quit("Demo loop exited???");
}
   _
```

0.2 Graphic Assets

All graphic assets were produced by Adrian Carmack. All of the work was done with Deluxe Paint (by Brent Iverson, Electronic Arts) and saved in ILBM[2] files (Deluxe Paint proprietary format). All assets were hand drawn with a mouse.

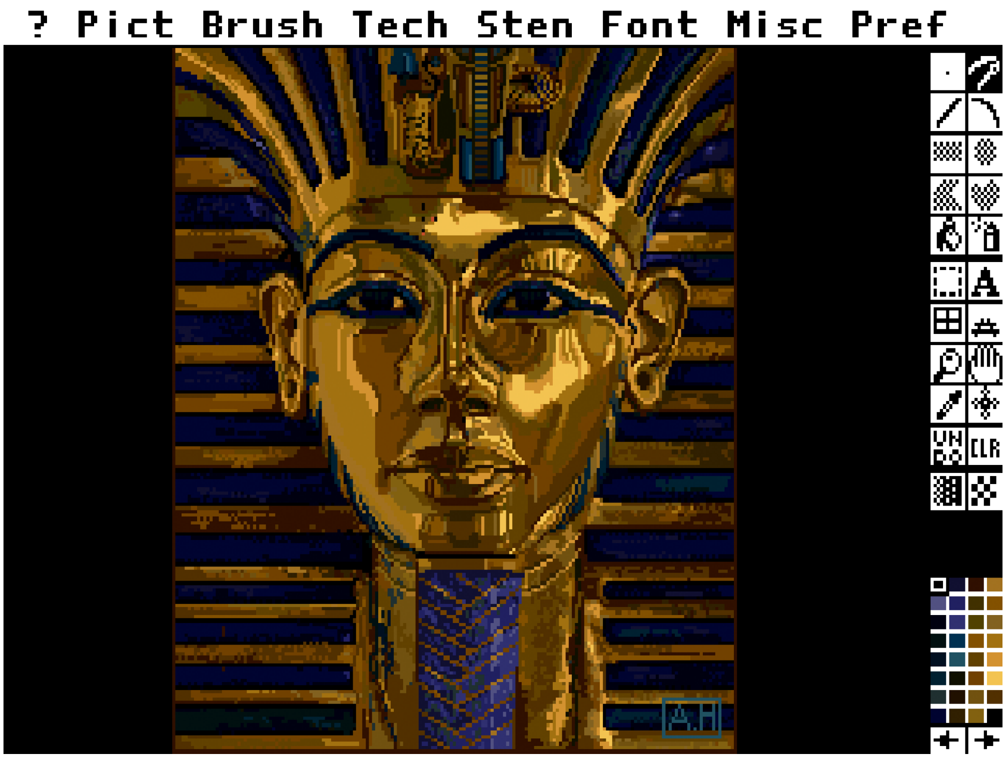


Figure 3: Deluxe Paint was used to draw all assets in the game.

0.2.1 Assets Workflow

After the graphic assets were generated, a tool (IGRAB) packed all ILBMs together in an archive and generated a header table file (`KDR`-format) and C header file with asset IDs. The engine references an asset directly by using these IDs.

[2]InterLeaved BitMap.

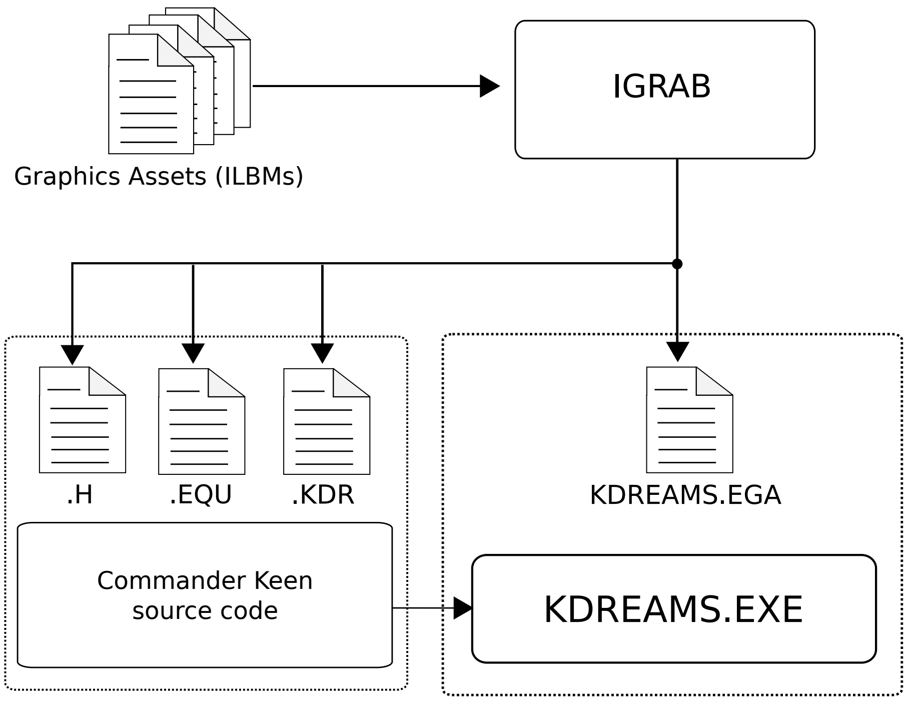


Figure 4: Asset creation pipeline for graphics items

```
//////////////////////////////////////
//
// Graphics .H file for .KDR
// IGRAB-ed on Fri Sep 10 11:18:07 1993
//
//////////////////////////////////////

typedef enum {
    #define CTL_STARTUPPIC        4
    #define CTL_HELPUPPIC         5
    #define CTL_DISKUPPIC         6
    #define CTL_CONTROLSUPPIC     7
    #define CTL_SOUNDUPPIC        8
    #define CTL_MUSICUPPIC        9
    #define CTL_STARTDNPIC       10
    #define CTL_HELPDNPIC        11
    #define CTL_DISKDNPIC        12
    #define CTL_CONTROLSDNPIC    13
    ...
    #define BOOBUSWALKR4SPR     366
    #define BOOBUSJUMPSPR       367
```

In the engine code, asset usage is hardcoded via an enum. This enum is an offset into the HEAD table which gives an offset in the DATA archive. The HEAD table files are stored in the \static folder as *.KDR files.

0.2.2 Assets file structure

Figure 5 shows the structure of the KDREAMS.EGA asset file.

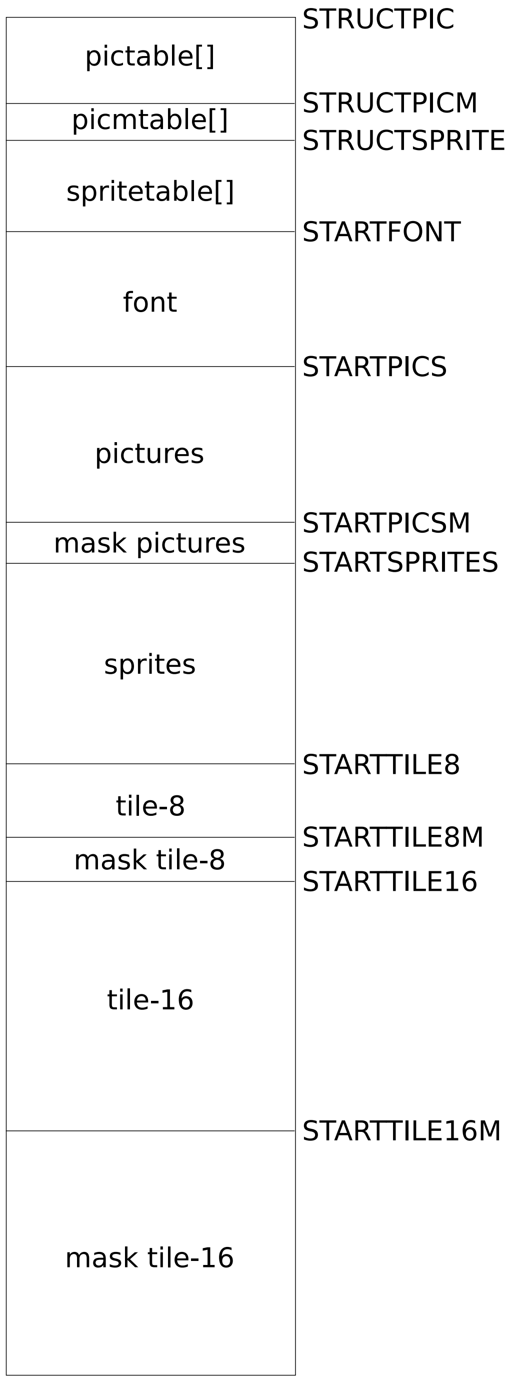


Figure 5: File structure of `KDREAMS.EGA` asset file.

The `pictable[]` contains the width and height in bytes for each picture in the asset file. Note that a width of 5 bytes means a size of 40 pixels on the screen. The same size structure is applied for mask pictures.

index	**width**	**height**
0	5	32
1	5	32
2	5	32
3	5	32
4	5	32
5	5	32
6	5	32
7	5	32
...	...	...
64	5	24

Table 1: content of pictable[].

The `spritetable[]` contains beside width and height in bytes also information on the sprite center, hit boundaries and number of shifted sprites (this will be explained in section xxx).

index	**width**	**height**	**orgx**	**orgy**	**xl**	**yl**	**xh**	**yh**	**shift**
0	3	24	0	0	0	0	368	368	4
1	3	32	0	0	64	0	304	496	4
2	3	30	0	16	64	0	304	496	4
3	3	30	0	32	64	48	304	496	4
4	3	32	0	0	64	0	304	496	4
5	3	30	0	32	64	48	304	496	4
...	...	...	...	...	...	...	...	...	...
296	12	103	-128	0	256	128	752	1648	4

Table 2: content of spritetable[].

As all tiles have fixed dimension (either 8 or 16 pixels), there is no need to store any tile size table structure. From `STARTPICS` location onwards all assets are stored.

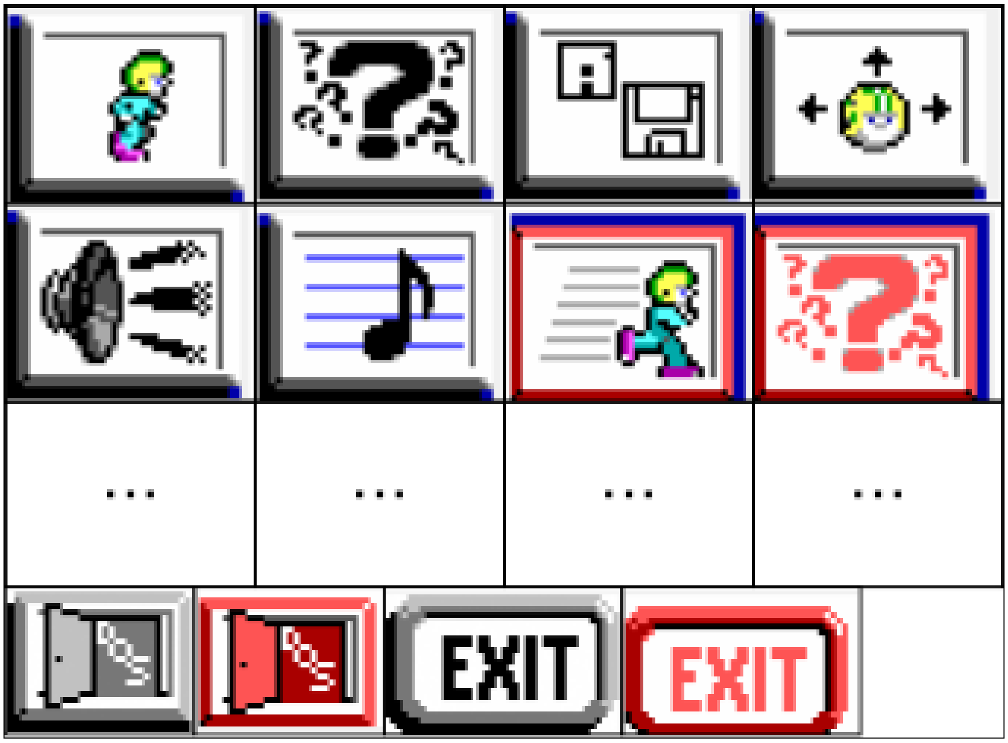


Figure 6: Picture assets data.

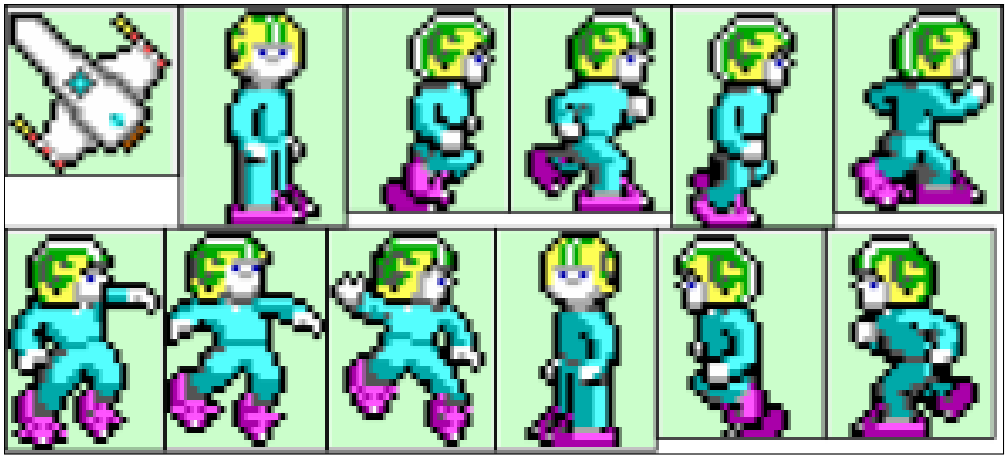
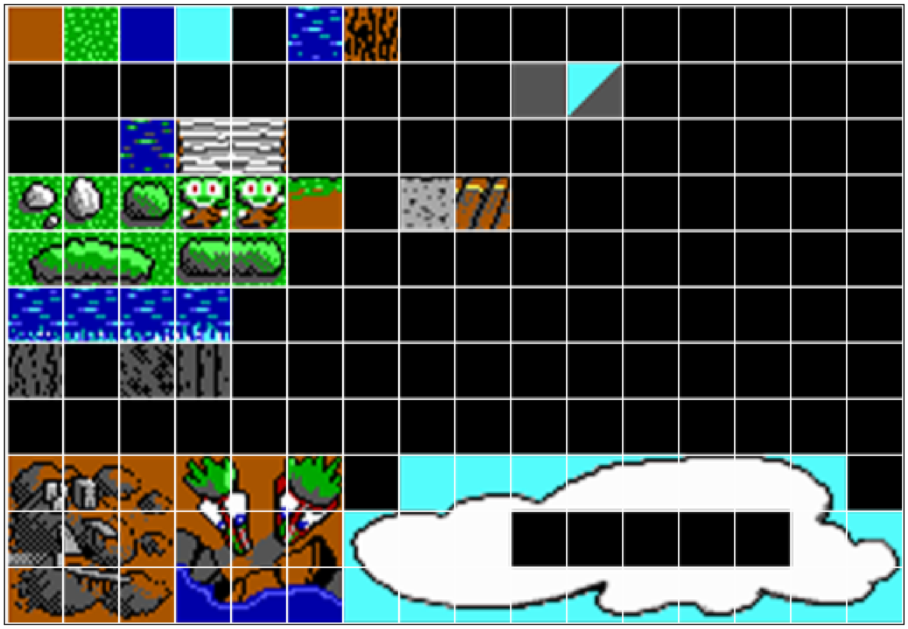


Figure 7: Sprite assets data.



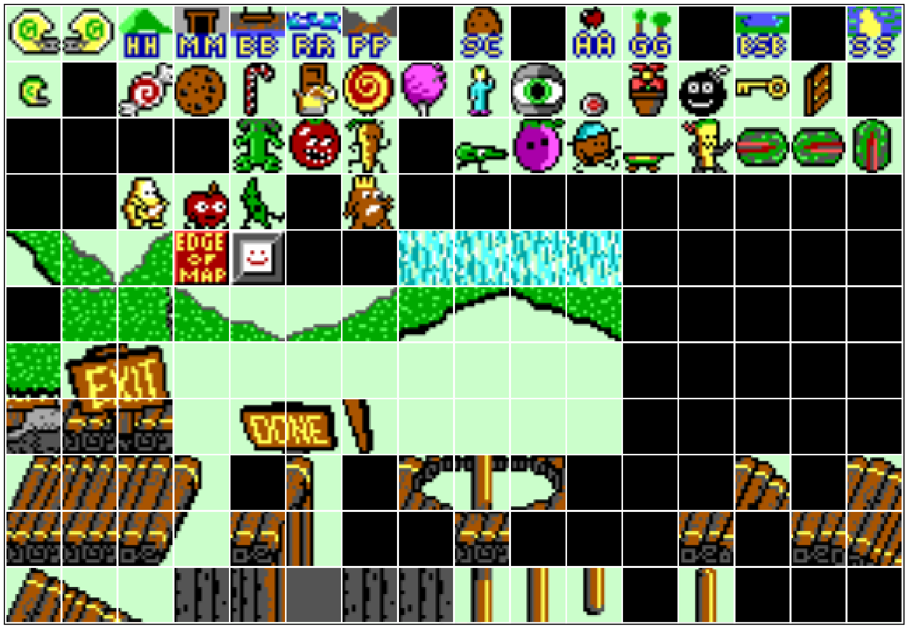
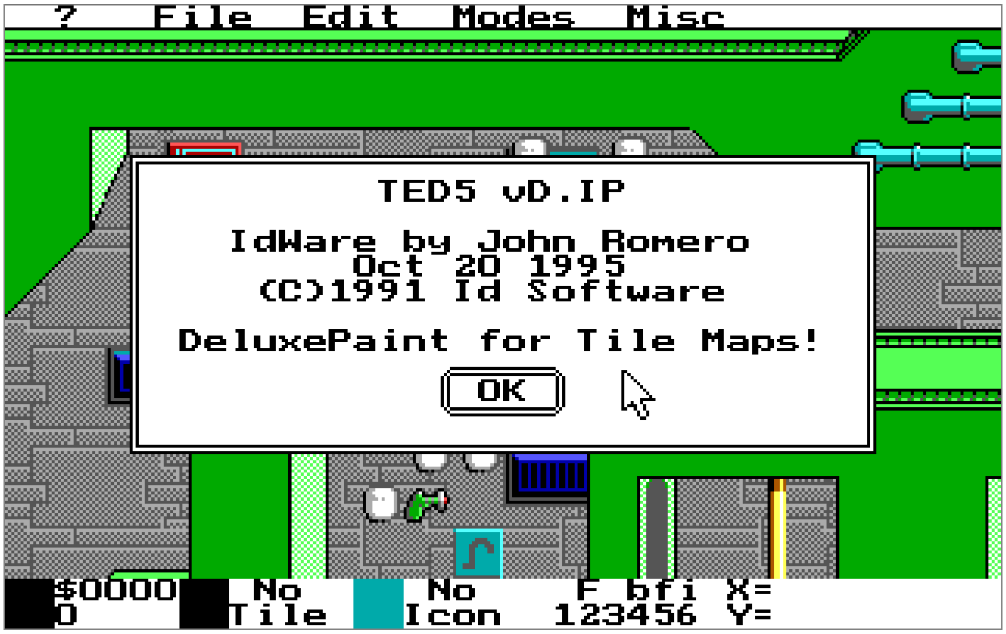


Figure 8: Tile-16 and masked tile-16 assets data.

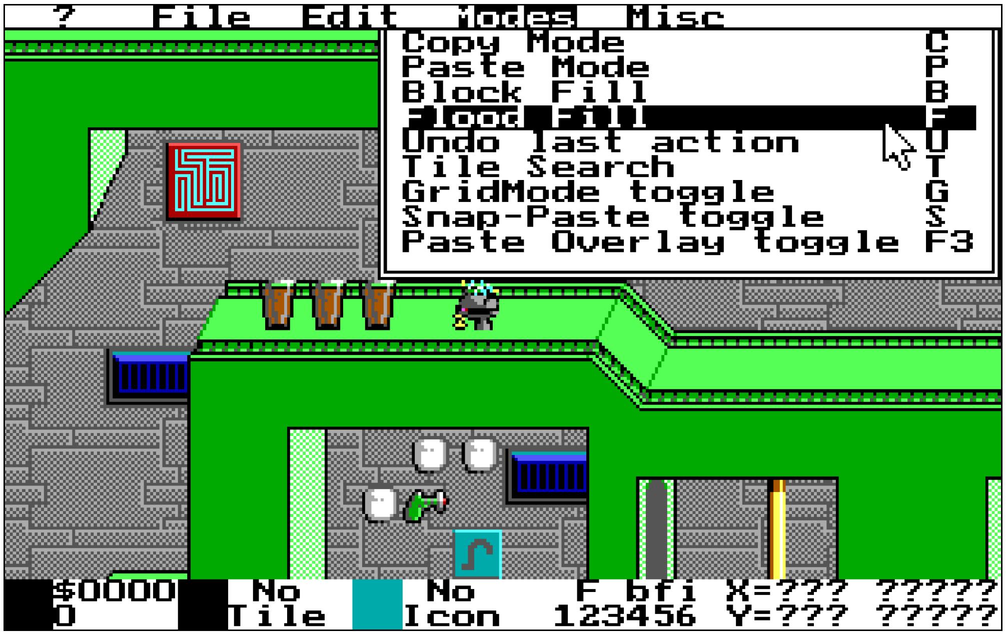
0.3 Maps

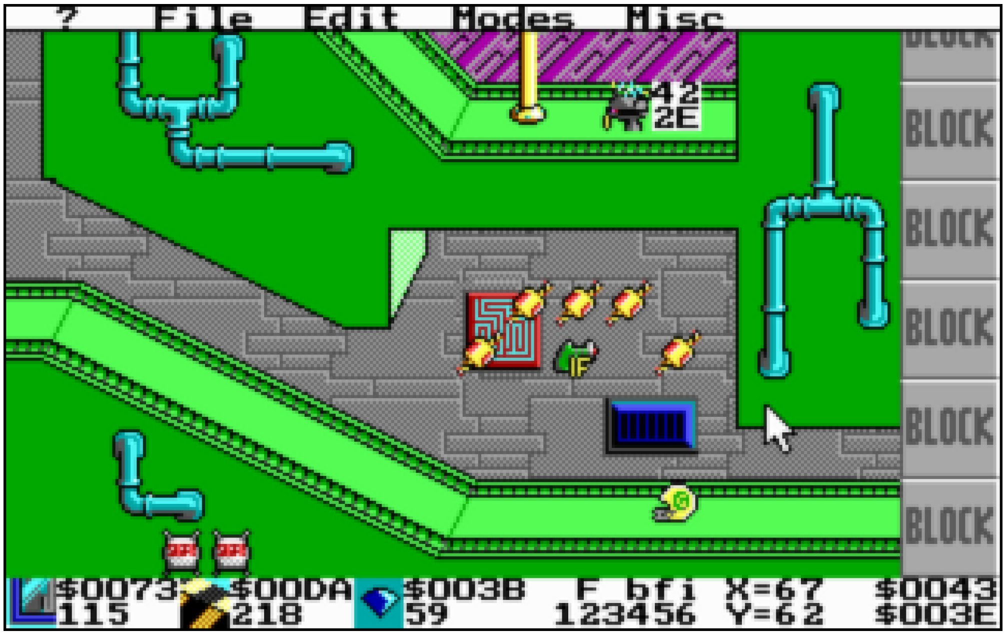
Maps were created using an in-house editor called TED5, short for Tile EDitor. TED5 was not created specially for Wolf 3D, but was originally made for the Commander Keen series and improved over the years. It was a versatile tool since it allowed for creating maps of both side-scroller games and top-down games like Rescue Rover and Wolf 3D.

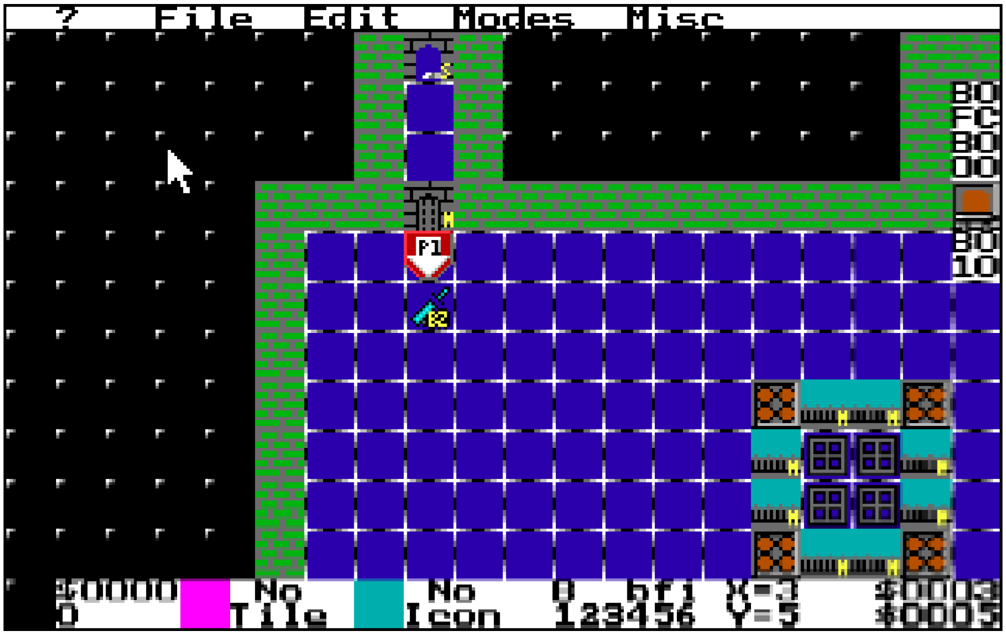
TED5 is not stand-alone; in order to start, it needs an asset archive and the associated header (as described in the graphic asset workflow Figure 4 on page 6). This way, texture IDs are directly encoded in the map.



<u>Trivia :</u> The suffix, "vD.IP", was put in by the Rise of the Triad team in 1994. It stood for "Developers of Incredible Power".


? File Edit Modes Misc
Copy Mode C
Paste Mode P
Block Fill B
Flood Fill F
Undo last action U
Tile Search T
GridMode toggle G
Snap-Paste toggle S
Paste Overlay toggle F3
$0000 D
No Tile
No Icon
F bfi 123456
X=??? Y=???
????? ?????


? File Edit Modes Misc
42 2E
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
$0073 115
$00DA 218
$003B 59
F bfi 123456
X=67 Y=62
$0043 $003E



TED5 allows placement of tiles on layers called "planes". This layered approach proved powerful and versatile. In Commander Keen, layers are used for background, tiles which the hero can stand on, generating bonuses and so on. For Wolfenstein 3D, two layers are used: one for walls, and one to place bonuses and enemies on.

Reusing TED5 was a double win. Not only did it save tool development time, but ramp-up time was also reduced as all team members had been using it for years. TED5 was so good at doing the job that it allowed designers to make a level within minutes[3].

> After talking with Romero and Tom, Scott learned that it was taking the group only about one day to make a level of the game. Ka-chung! Dollar signs! Instead of just three episodes, why not have six? Scott said, "If you can do thirty more levels, it would only take you fifteen days. And we could have it where people could buy the first trilogy for thirty-five dollars or get all six for fifty dollars, or if people buy the first episode and later want the second episodes it will be twenty dollars. So there's a reason to get them all!" After some consideration, id agreed.
>
> **- Masters of Doom**

[3]TED5 was used in 33 commercially shipped games (including Rise of the Triad in 1994).

John Romero and Tom Hall did all the map design with TED5. Bobby Prince helped too and is credited for maps E6M2 and E6M3.

Trivia : The source code of TED5 was released several years later. Among the source code made of `.C` and `.H` was a mysterious `_TOM.PIC`[4]. It turned out to be an adult caricature of Tom Hall made by Adrian Carmack. The explanation came from John Romero:

> "Hahahaha! Wow, I forgot all about that picture. I can't believe it's in the TED5 source files! It's basically a pic that Adrian drew of Tom [...] saying "Sorry!".
>
> It's because Tom and Adrian used to share a worktable together. Tom would always bump the table while Adrian was drawing graphics with the mouse and Tom would say, "Sorry!".
>
> **John Romero - Programmer**

[4]Intentionally not reproduced here.