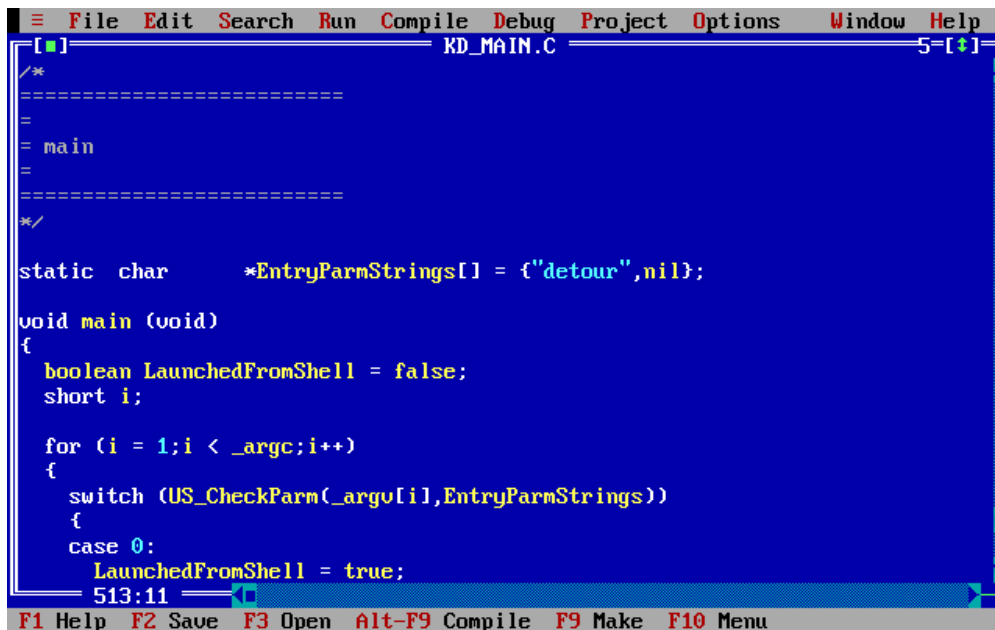## 0.1 Programming

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).

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].

```
  ≡   File   Edit   Search   Run   Compile   Debug   Project   Options      Window   Help
┌─[■]══════════════════════════ KD_MAIN.C ═══════════════════════5═[↕]═┐
│/*                                                                     ▲
│════════════════════════                                              █
│=                                                                     █
│= 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;                                       │
└════ 513:11 ════◄□═════════════════════════════════════════════►═┘
 F1 Help  F2 Save  F3 Open  Alt-F9 Compile  F9 Make  F10 Menu
```

Figure 1: Borland C++ 3.1 editor

---

[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.
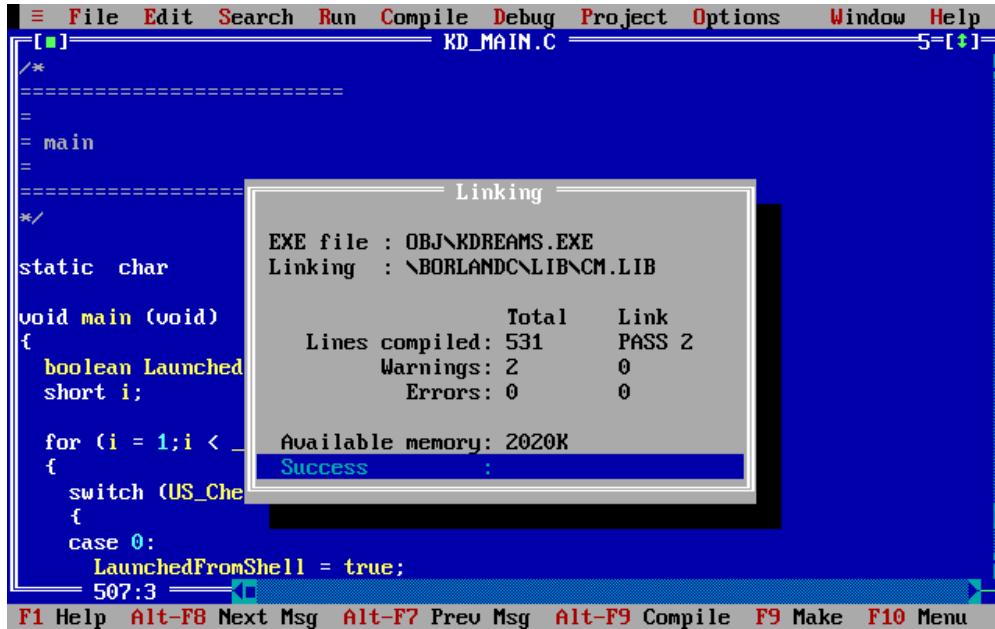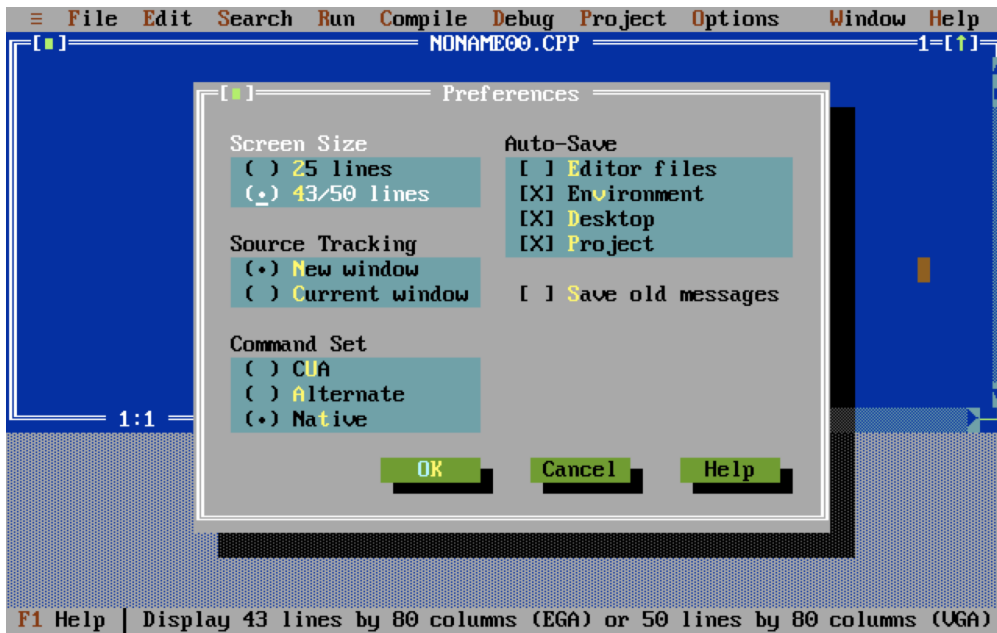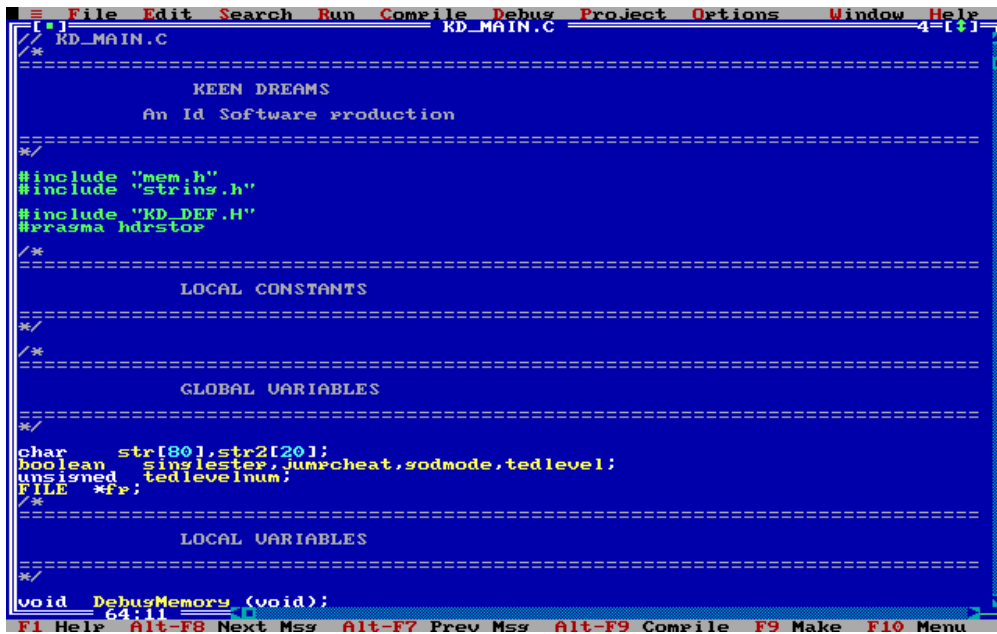

Figure 2: Compiling Keen Dreams with Borland C++ 3.1

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=[↑]┐
        ┌─[■]═══════════════════ 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

                    ┌─OK─┐      ┌─Cancel─┐      ┌─Help─┐

 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=[↕]┐
// 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);
═══ 64:11 ═══
 F1 Help  Alt-F8 Next Msg   Alt-F7 Prev Msg   Alt-F9 Compile   F9 Make   F10 Menu
```

3

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

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 a compressed archive and generated a header table and dictionary file (KDR-format[3]) and C header file with asset IDs. The engine references an asset directly by using these IDs.

---

[2]InterLeaved BitMap.
[3]both KDR-files are located in the `static` folder of the source code.
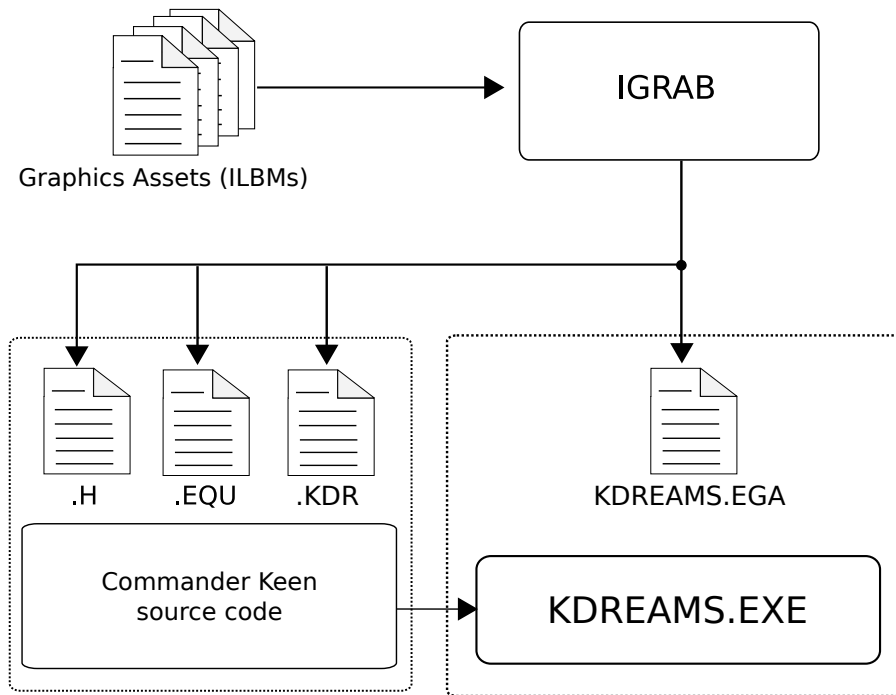
Figure 4: Asset creation pipeline for graphics items

```
////////////////////////////////////
//
// 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 hard-coded via an enum. This enum is an offset into the HEAD table which contains an offset in the DATA archive.

0.2.2 Assets archive file structure

Figure 5 shows the structure of the `KDREAMS.EGA` asset archive file. The first sections contain data tables for pictures and sprites, followed by the font and finally all graphs.

pictable[]	STRUCTPIC
picmtable[]	STRUCTPICM STRUCTSPRITE
spritetable[]	STARTFONT
font	STARTPICS
pictures	STARTPICSM STARTSPRITES
mask pictures	
sprites	STARTTILE8
tile-8	STARTTILE8M
mask tile-8	STARTTILE16
tile-16	STARTTILE16M
mask tile-16	

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

Boh the **pictable[]** and **picmtable[]** contain the width and height for each (masked) picture in the asset file.

picture index	width (bytes)	height (bytes)
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

The **spritetable[]** contains, beside width and height, also information on the sprite center, hit boundaries and number of bitshifts. Sprites will be explained in detail in section **??** (page **??**).

The **font** segment contains a table for the height (same for all characters) and width of the font, as well as a reference where the character data is located.

```
// ID_VW.H

typedef struct
{
  int height;
  int location[256];
  char width[256];
} fontstruct;
```
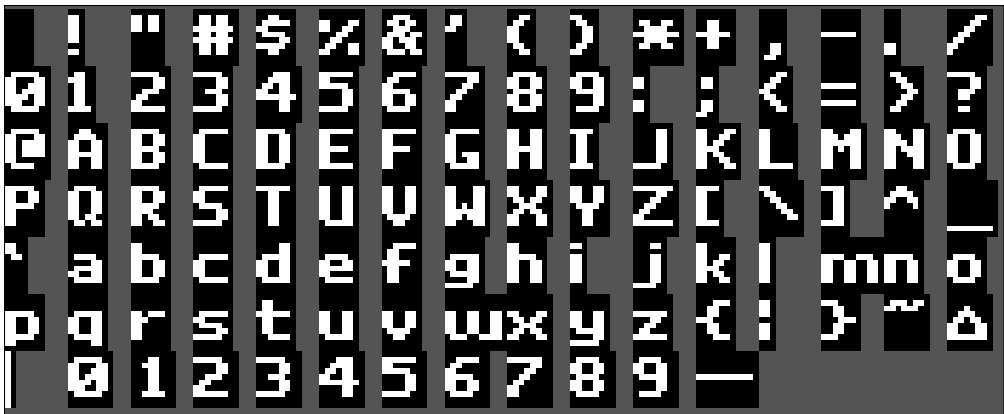
Figure 6: Font asset data.

After the (mask) pictures and sprites graphs, starting STARTPICS, the 8- and 16-bit size (masked) tiles are stored. Each asset contains four planes, aligned with the EGA architecture. All masked tiles and sprites include an additional mask plane as well.

Figure 7: Picture asset data.

Figure 8: Sprite asset data.

	000h	001h	002h	003h	004h	005h	006h	007h	008h	009h	00Ah	00Bh	00Ch	00Dh	00Eh	00Fh
000h																
010h																
020h																
030h																
040h																
050h																
060h																
070h																
080h																
090h																
0A0h																

12

Figure 9: Background (Tile16) and foreground (masked Tile16) assets data.

0.3   Maps

Maps were created using an in-house editor called TED5, short for Tile EDitor. Over the years TED5 had improvements and the same tool is later used for creating maps of both side-scrolling games and top-down games like *Wolfenstein 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.
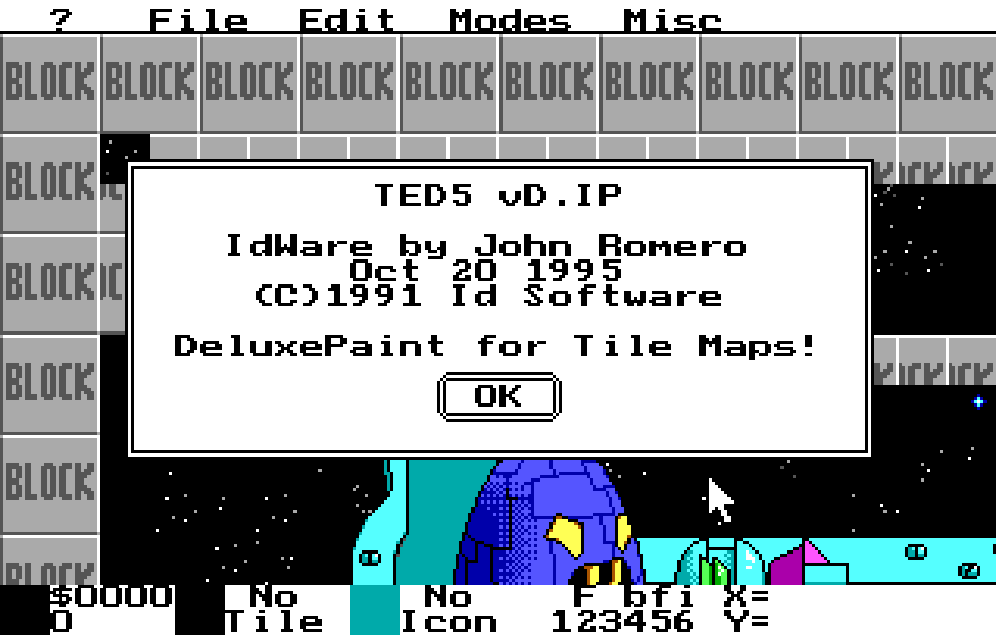
Trivia : The suffix, "vD.IP", was put in by the Rise of the Triad team in 1994. It stood for "Developers of Incredible Power".
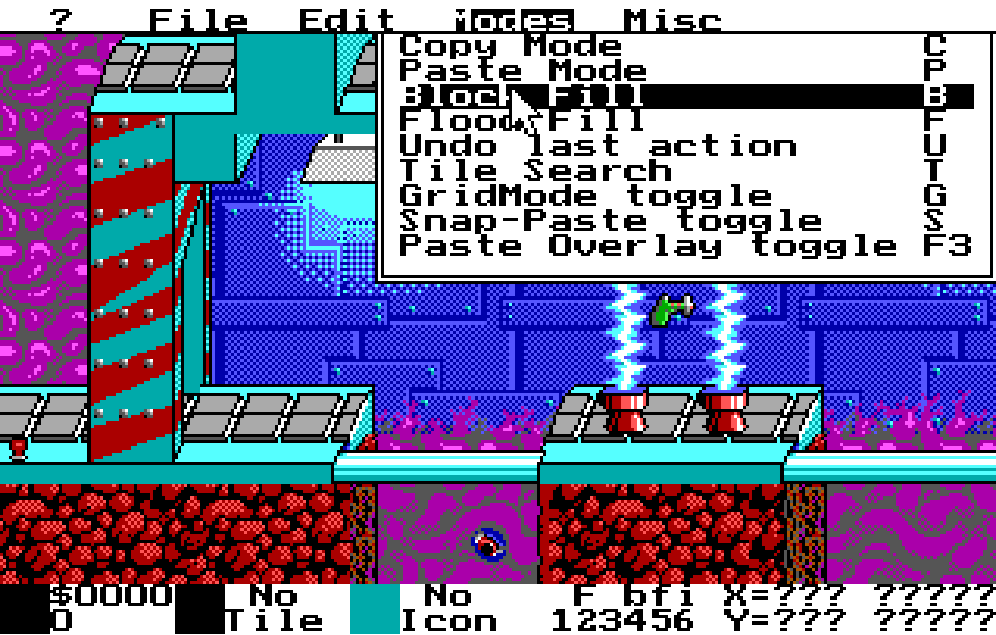
TED5 allows placement of tiles on layers called "planes". In Commander Keen, there are three type of planes:

- Background plane

- Foreground plane, which is a mask over the background plane

- Information plane, containing location of actors and special places

A level is created by placing tiles on each of these three layers. Each foreground and background tile could be enriched with additional tile information. It controls how everything interacts with tiles, like tile clipping, "deadly" tiles and animated tiles.
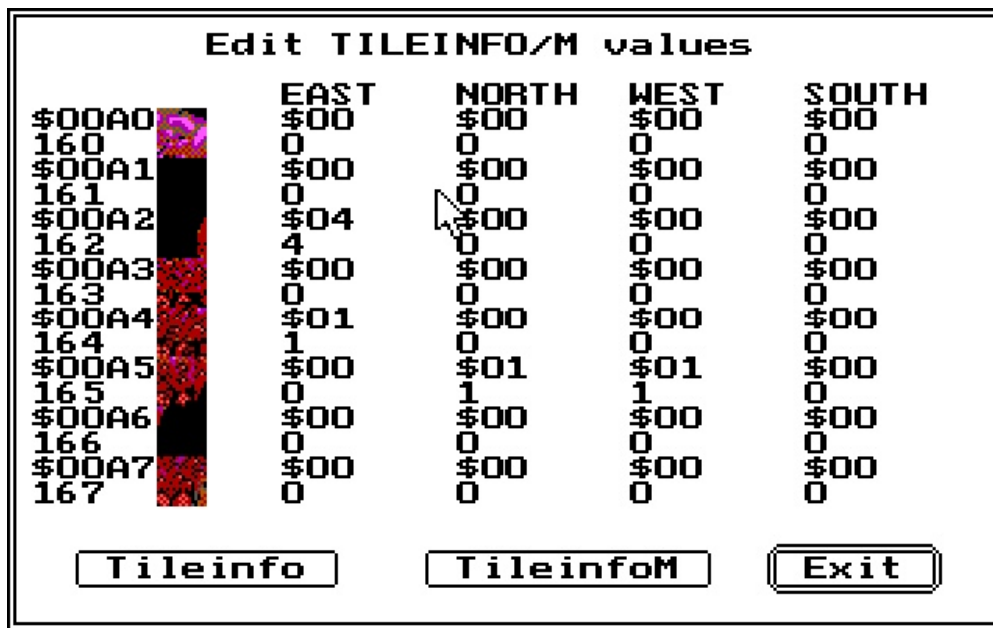
Just like IGRAB, the TED5 tool packed all levels together in a compressed `MAP` archive and generated a header and dictionary file (`KDR`-format).

0.3.1 Map header structure

The map header structure is part of the engine code and contains a header offset and size, which refers to the location and size in the `KDREAMS.MAP` archive file.

```
typedef struct
{
  unsigned  RLEWtag;                  // RLE flag
  long    headeroffsets[100];
  byte    headersize[100];            // headers are very small
  byte    tileinfo[];
} mapfiletype;
```

A maximum of 100 maps is supported in the game. The `tileinfo[]` data contains information for tile interaction.
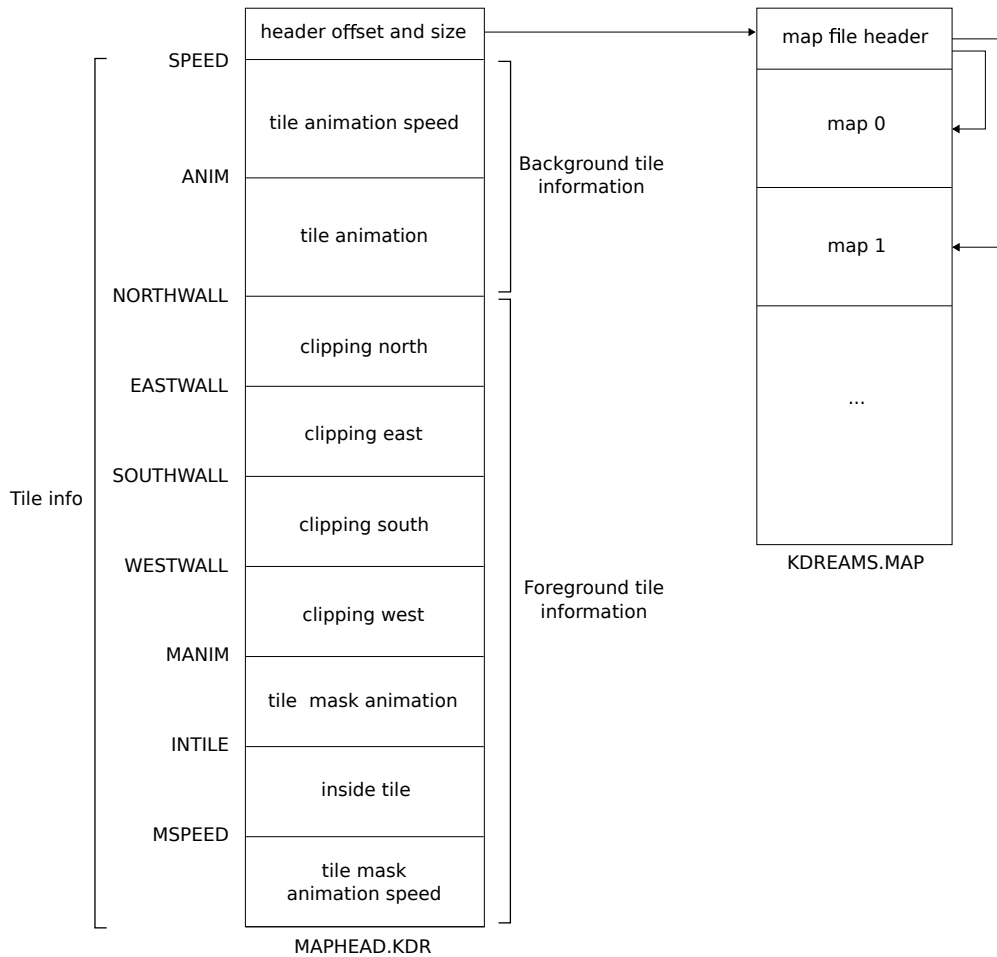
Figure 10: File structure of `MAPHEAD.KDR` header file.

For background tile animation two information tables are defined: **tile animation** and **tile animation speed**. The tile animation refers which tile, relative to this tile, the tile will animate to. So in case of tile #90 (see Table **??**), the next animation tile is #91 (+1), followed by #92 (+1) and #93 (+1). After tile #93 (-3) the sequence is going back to tile #90. The animation speed is expressed in `TimeCount`, which is the number of ticks before the next tile is displayed.

index	tile animation	tile animation speed
0	0	0
1	0	0
...	...	...
57	1	32
58	-1	24
...	...	...
90	1	8
91	1	8
92	1	8
93	-3	8
...	...	...

The foreground tiles contain, beside tile animation (similar like background tiles), also clipping and 'inside' tile information.

The clipping tables contain how Commander Keen is clipped against foreground tiles. 'Inside' tile information is used to climb on a pole and mimics Commander Keen going through a floor opening ('inside' the tile). In section **??** (see page **??**) both the clipping and 'inside' logic is further explained.

index	clip north	clip east	clip south	clip west	inside tile
...	...	...	...	...	...
238	0	1	5	0	0
239	0	0	0	0	0
240	0	0	5	0	0
241	0	0	0	0	128
242	1	1	1	1	128
243	1	0	1	0	0
244	0	0	2	0	0
...	...	...	...	...	...

0.3.2   Map archive structure

The structure of `KDREAMS.MAP` archive is explained in Figure 11. For each map there is a small header containing the width, height and name of the map as well as a reference pointer to each of the three planes. Each plane exists out of a map of tile numbers for background, foreground and info.

```
typedef struct
{
  long        planestart[3];
  unsigned    planelength[3];
  unsigned    width,height;
  char        name[16];
} maptype;
```
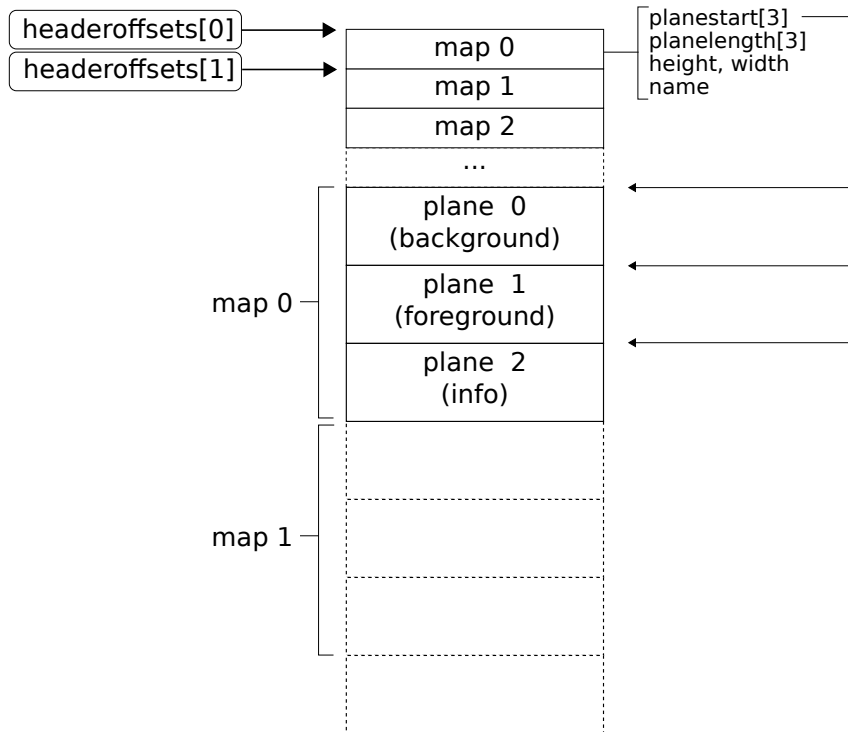

Figure 11: File structure of `KDREAMS.MAP` file.

0.4 Audio

0.4.1 Sounds

The original Commander Keen Trilogy did only support the default PC Speaker. Only with the introduction of Commander Keen Dreams the team decided to support sound cards.

Trivia : Apogee, the publisher of Ideas from the Deep, didn't publish any game with AdLib support until *Dark Ages* in 1991. And even then it still had pc speaker music.

As mentioned in Chapter **??**, audio hardware was highly fragmented. Commander Keen supported three sound cards and the default PC speaker, which meant generating assets multiple times for each and packing them together with an in-house tool called MUSE into an AUDIOT archive (an id software proprietary format):

Figure 12: MUSE splash screen.

id Software intended for Keen Dreams to have music and digital effects support for the SoundBlaster & Sound Source devices. In fact, Bobby Prince composed the song "You've Got to Eat Your Vegetables!!" for the game's introduction. However, Softdisk Publishing wanted Keen Dreams to fit on a single 360K floppy disk, and in order to do this, id Software had to scrap the game's music at the last minute. The team didn't even have time to remove the music setup menu.

Figure 13: Setup music menu in Keen Dreams, although there is no music in the game.

Trivia : The song "You've Got to Eat Your Vegetables!!" would finally make its debut in Commander Keen IV: Secret of the Oracle.

As a consequence, only two sets of each audio effect are shipped with the game:

1. For PC Speaker

2. For AdLib

0.4.2 Sound effects

All sound effects are done by Robert Prince. In the early days of the OPL soundcards, the "gold standard" sequencing software was Sequencer Plus Gold ("SPG") by Voyetra.

The reason for this was it had an OPL instrument/instrument bank editor. To rough out compositions, Robert used Cakewalk ("CW").
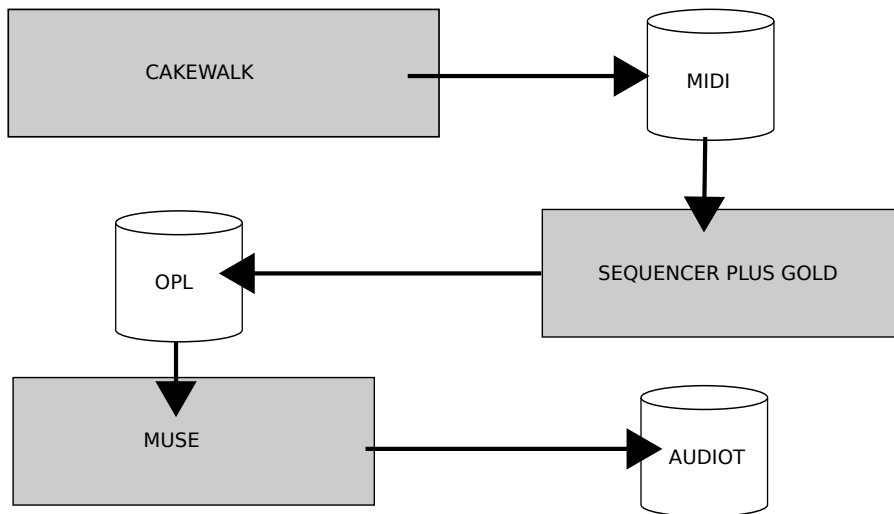
Figure 14: Music and sound effect pipeline as used by Bobby Prince.
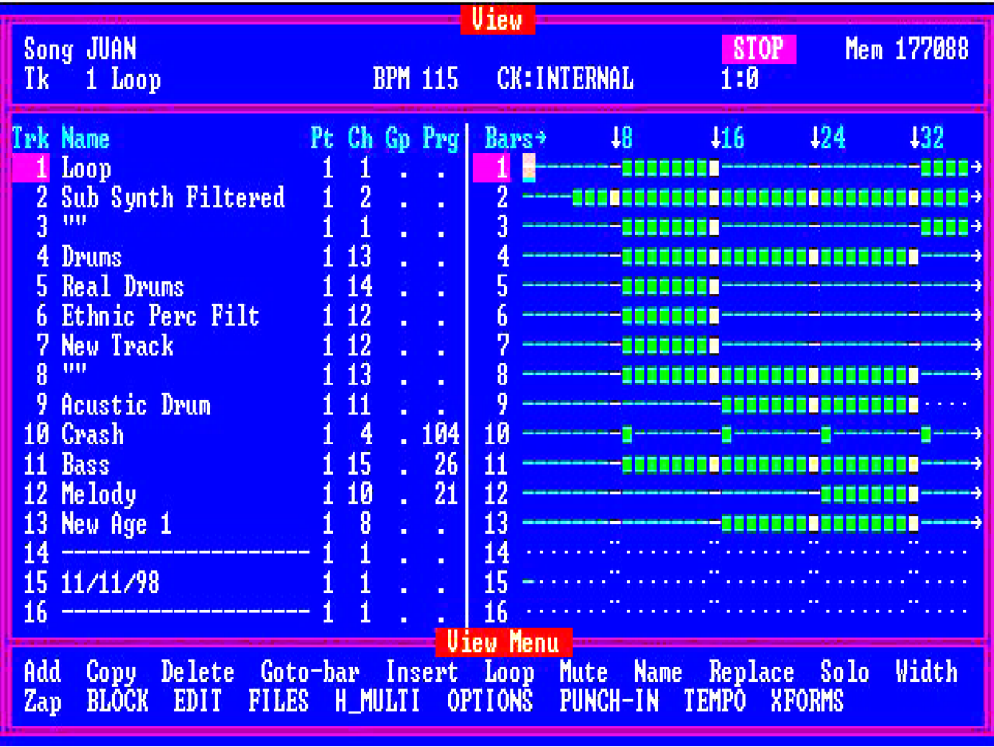
Figure 15: Sequencer Plus Gold ("SPG") by Voyetra.

What goes in the `AUDIOT` archive is a music and sound effect format called IMF[4]. As it supports only the YM3812, it is tailored for the chip with zero abstraction layers. It consists of a stream of machine language commands with associated delays[5].

The stream pilots the nine channels in the OPL2. A channel is able to simulate an instrument and play notes thanks to two oscillators, one playing the role of a modulator and the other the role of a carrier. There are many other ways to control a channel such as envelope, frequency or octave.

The way a channel is programmed is described in detail in Section **??**, "**??**" on page **??**.

---

[4]Id software Music File.
[5]IMF format is explained in detail on page **??**

Figure 16: Architecture of a YM3812 channel.

Trivia : The YM3812's unmistakable sonority is due to its peculiar set of waveform transformers (they are right after the output of each oscillator in the drawing). Four waveforms are available on the OPL2: Sin (1), Abs-sin(2), Pulse-sin (3), and Half-sin (4).
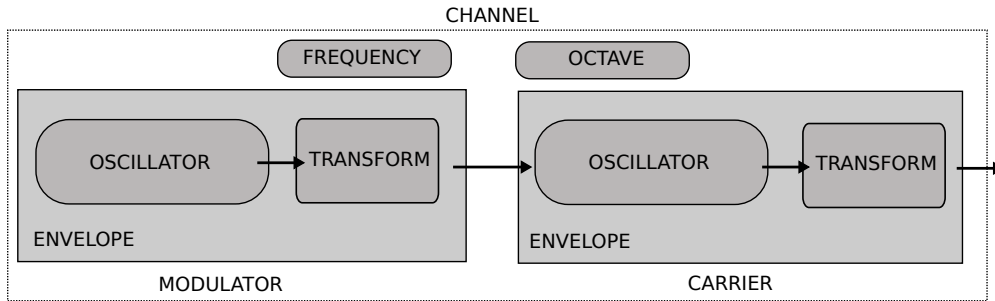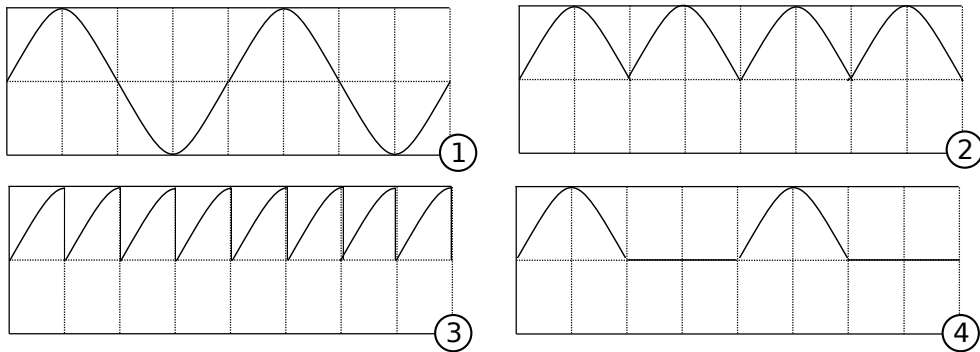

Figure 17: The four waveform transforms available.

0.5   Distribution

On December 14th, 1990 the first episode was released via Apogee. Episodes 1-5 are all published by Apogee Software. The game engine and first episode were given for free and encourages to be copied and distributed to a maximum number of people. To receive the other episodes, each player had to pay $30 (for Episode 1-3) to *ideas from the Deep*.

Commander Keen in Keen Dreams was published as a retail title by Softdisk, as part of a settlement for using Softdisk resources to make their own game[6].

---

[6]The settlement with Softdisk is explained in Appendix **??**

Figure 18: Retail version of Commander Keen in Keen Dreams by Softdisk.

In 1990, the Internet was still in its infancy and the best medium was the 3½-inch floppy disk. The game shipped as follows:

The files can be divided in seven parts:

- KDREAMS.EXE: Game engine.

- KDREAMS.EGA: Contains all the assets (sprites, tiles) needed during the game.

- KDREAMS.AUD: Sound effect files.

- KDREAMS.MAP: Contains all levels layouts.

- KDREAMS.CMP: Introduction picture of the game, which is a compressed LBM image file.

- Softdisk Help Library files, which are text screens, shown when starting the game
    - START.EXE: Decompress and show user guide, and then start the game.
    - LOADSCN.EXE: Shows the closing screen when quitting the game. Decompresses the ENDSCN.SCN chunk in LAST.SHL
    - *.SHL: Text user guide assets.
- Several *.TXT files which can be read in DOS by typing the corresponding *.BAT file.

```
Directory of C:\KDREAMS\.
.               <DIR>              16-05-2023 21:04
..              <DIR>              16-05-2023 20:52
BKGND    SHL              285 16-05-2023 20:59
FILE_ID  DIZ              508 16-05-2023 20:59
HELP     BAT               34 16-05-2023 20:59
HELPINFO TXT            1,038 16-05-2023 20:59
INSTRUCT SHL            2,763 16-05-2023 20:59
KDREAMS  AUD            3,498 16-05-2023 20:59
KDREAMS  CFG              656 16-05-2023 21:00
KDREAMS  CMP           14,189 16-05-2023 20:59
KDREAMS  EGA          213,045 16-05-2023 20:59
KDREAMS  EXE          354,691 04-05-2023 19:40
KDREAMS  MAP           65,673 16-05-2023 20:59
LAST     SHL            1,634 16-05-2023 20:59
LICENSE  DOC            8,347 16-05-2023 20:59
LOADSCN  EXE            9,959 16-05-2023 20:59
MENU     SHL              447 16-05-2023 20:59
NAME     SHL               21 16-05-2023 20:59
ORDER    SHL            1,407 16-05-2023 20:59
PRODUCTS SHL            4,629 16-05-2023 20:59
QUICK    SHL            3,211 16-05-2023 20:59
README   TXT            1,714 16-05-2023 20:59
START    EXE           17,446 16-05-2023 20:59
VENDOR   BAT               32 16-05-2023 20:59
VENDOR   DOC           11,593 16-05-2023 20:59
VENDOR   TXT              810 16-05-2023 20:59
   24 File(s)        717,630 Bytes.
    2 Dir(s)     262,111,744 Bytes free.

C:\KDREAMS>
```

Figure 19: All Keen Dreams files as they appear in DOS command prompt.