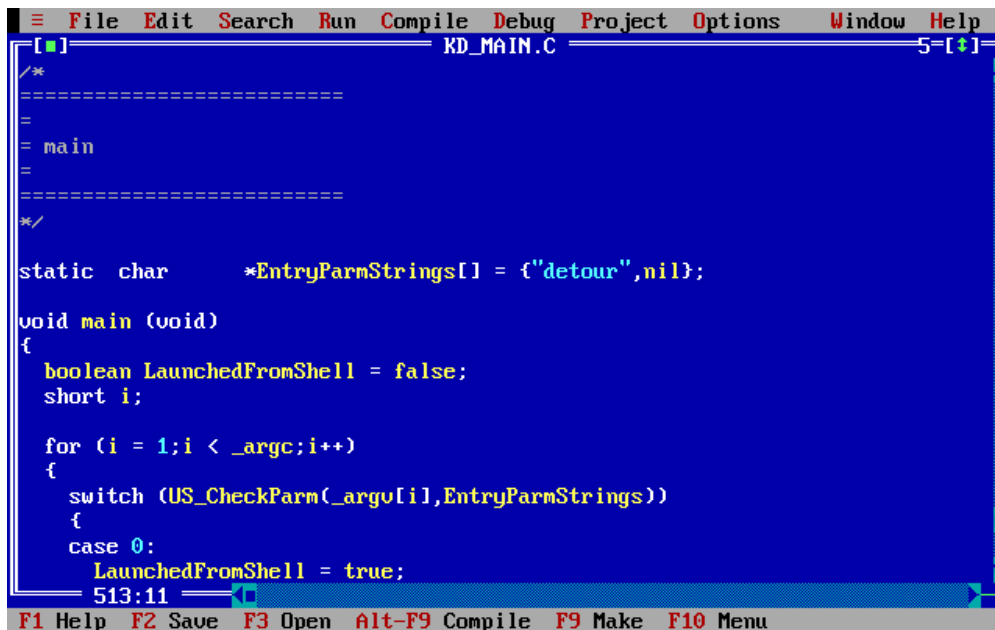# 0.1 Programming

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].



***Figure 1:*** Borland C++ 3.1 editor

---

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

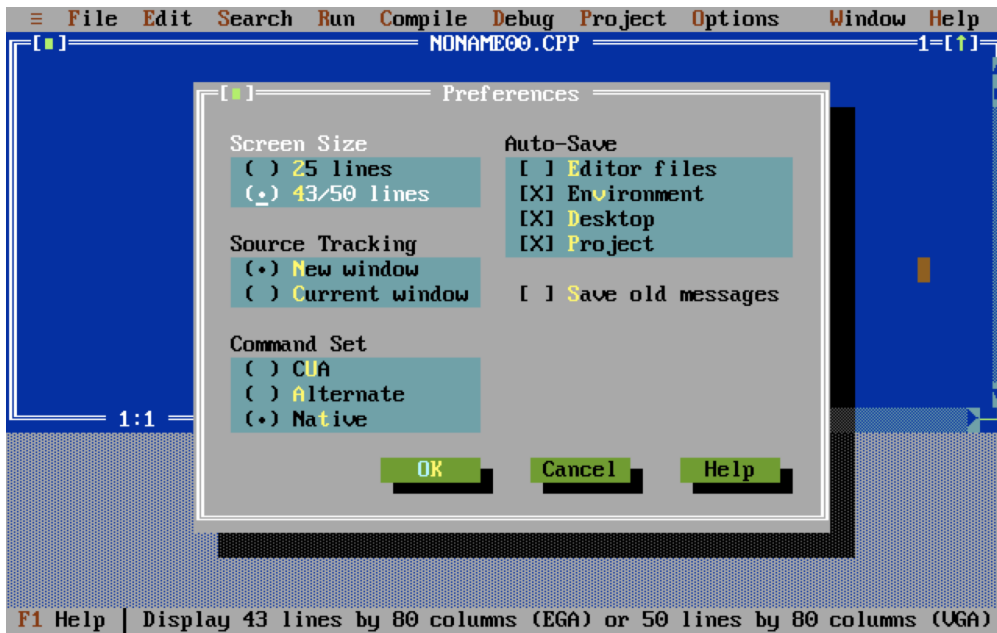There was no need to enter command-line mode however. The IDE allowed to create a project, build, run and debug.
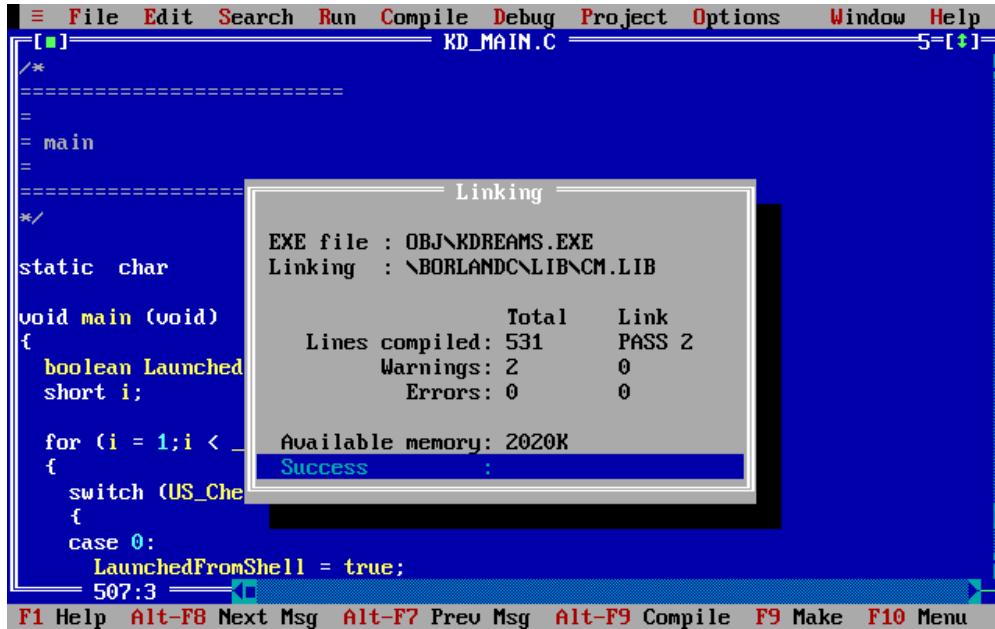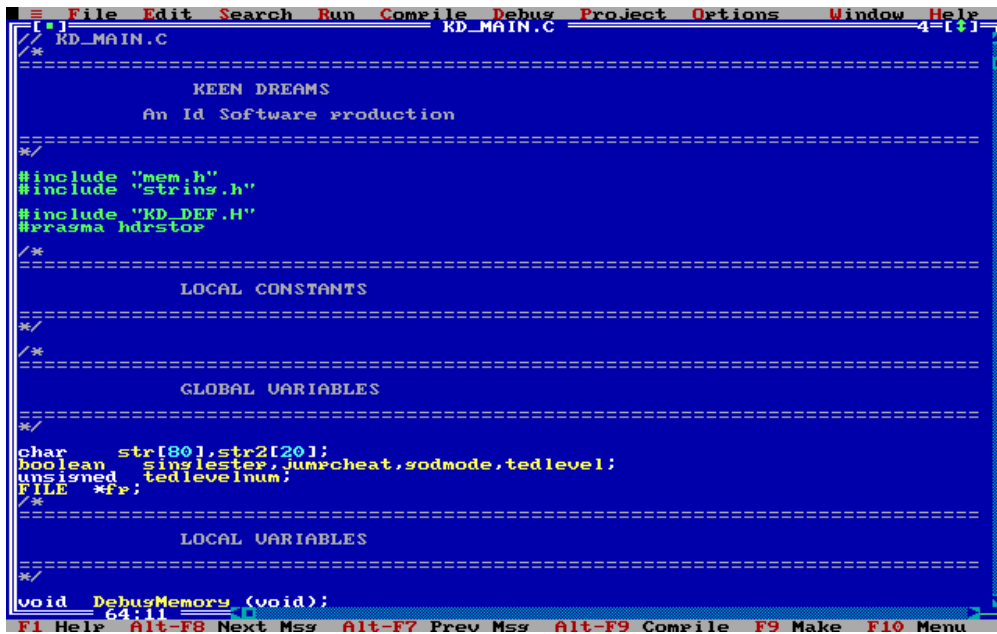


**Figure 2:** Compiling Keen Dreams with Borland C++ 3.1

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

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

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





4

## 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.
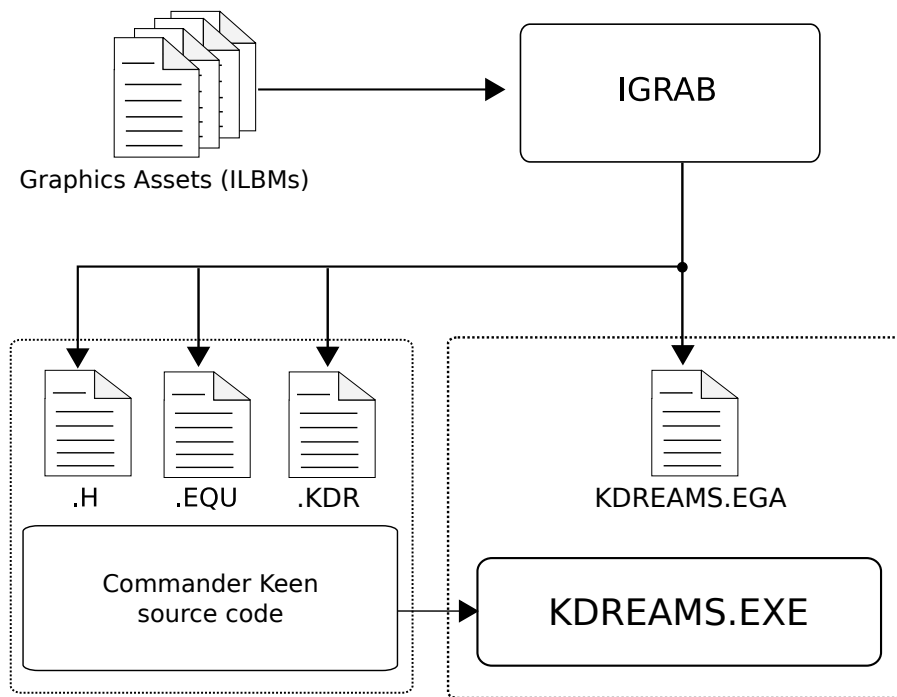
**Figure 4:** Asset creation pipeline for graphics items

```
////////////////////////////////////
//
// 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 contains 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.

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

**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 **??**, page **??**).

| 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[].

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;
```
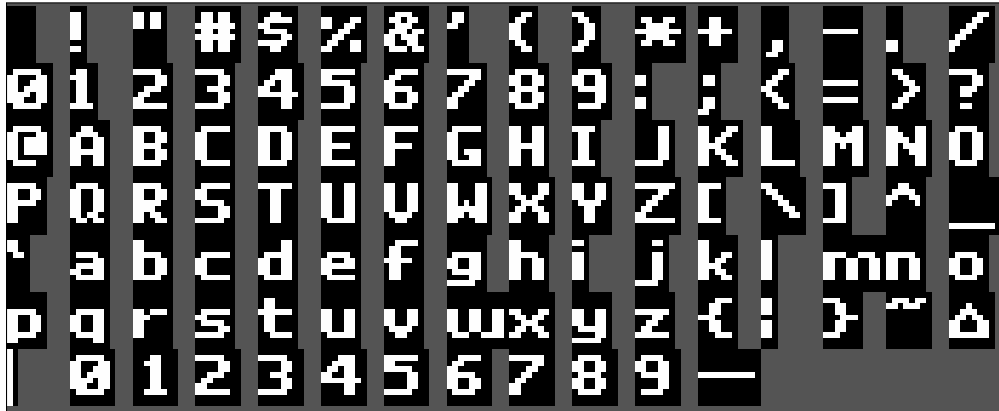


***Figure 6:*** Font asset data.

Since 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 graphical assets are stored. Each asset contains four planes, aligned with the EGA architecture. Foreground tiles and sprites include a mask plane as well.

**Figure 7:** Picture asset data.
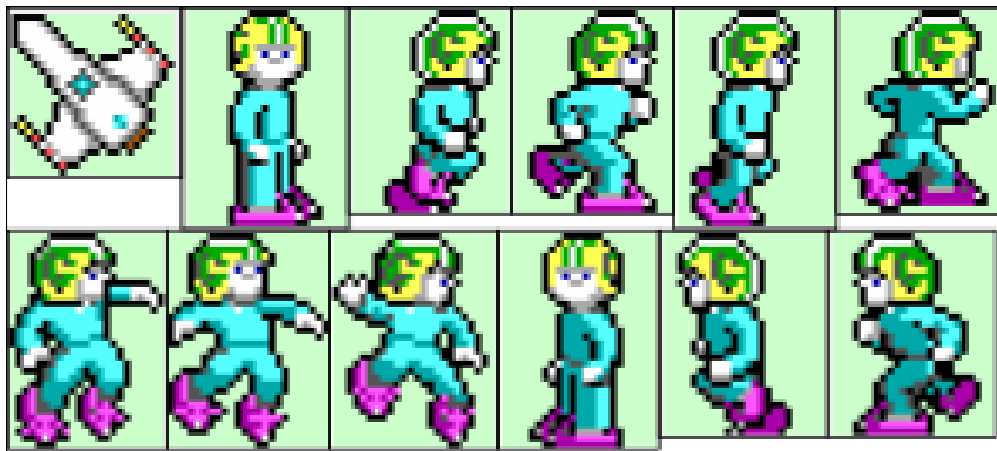


**Figure 8:** Sprite asset data.

**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 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.
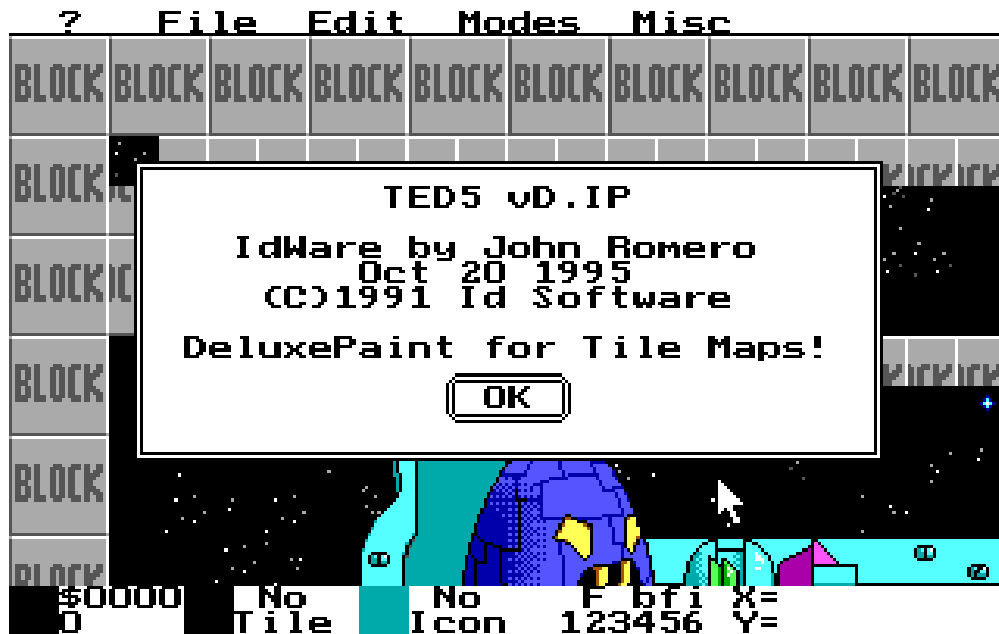


**Trivia :** The suffix, "vD.IP", was put in by the Rise of the Triad team in 1994. It stood for "Developers of Incredible Power".
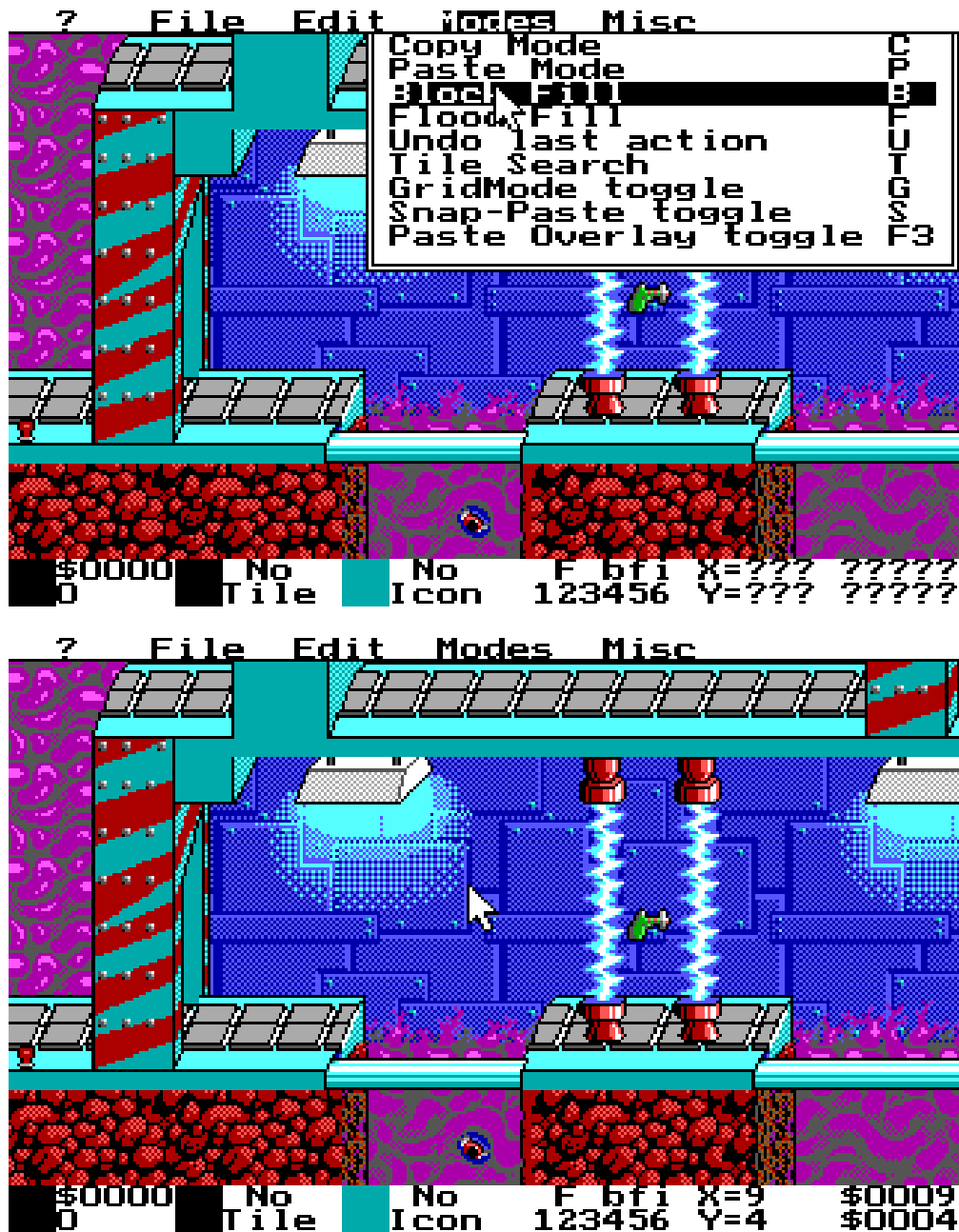
TED5 allows placement of tiles on layers called "planes". In Commander Keen, layers are used for background, foreground and information planes. Note that foreground tiles are

always using a mask as they are overlayed with the background. The info plane contains the location of actors and special places. Each foreground and background tile could also be enriched with additional tile information such as tile clipping and animated tiles. Just like IGRAB, the TED5 tool generated a header table file (`KDR`-format) and a `*.MAP` file containing the levels. Figure 10 shows the map header table structure, which is hard-coded in the source code.
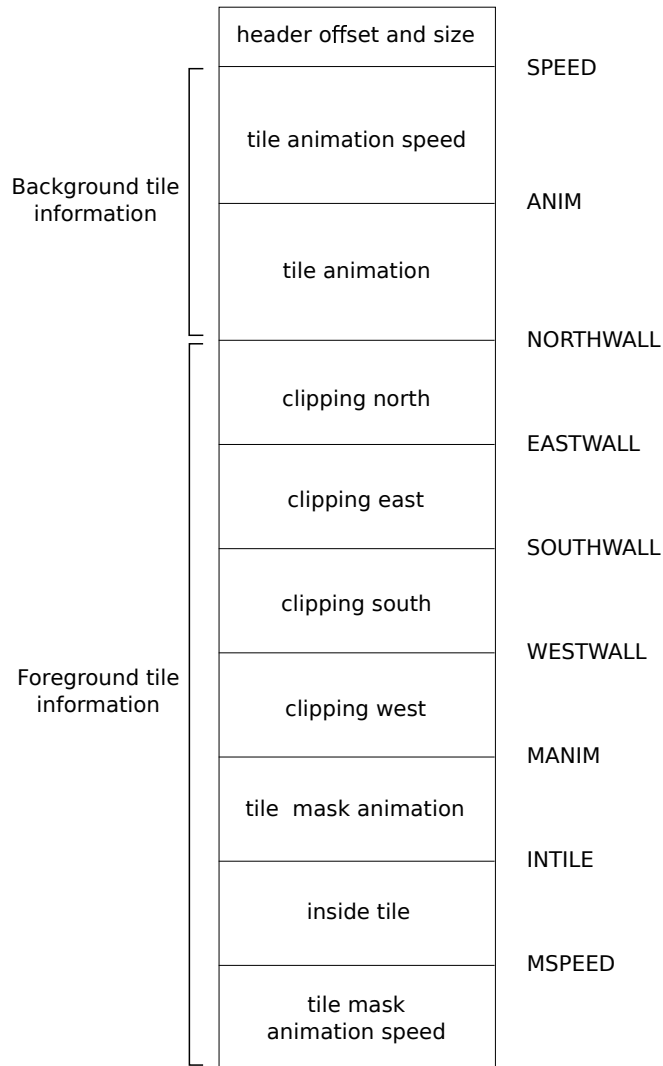
**Figure 10:** File structure of `MAPHEAD.KDR` header file.

15

### 0.3.1  Map header structure

The header offset and header size refer to the location and size in the `KDREAMS.MAP` file. A maximum of 100 maps is supported in the game.

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

             LOCAL CONSTANTS

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


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

### 0.3.2  Background tile information

For background tile animation two information tables are required: tile animation and tile animation speed. The tile animation refers to the next tile in the animation sequence. So in case of tile #90 in Table 3, 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 |
| ... | ... | ... |

*Table 3:* Background tile animation.

### 0.3.3   Foreground tile information

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 **??** both the clipping and 'inside' logic is further explained.
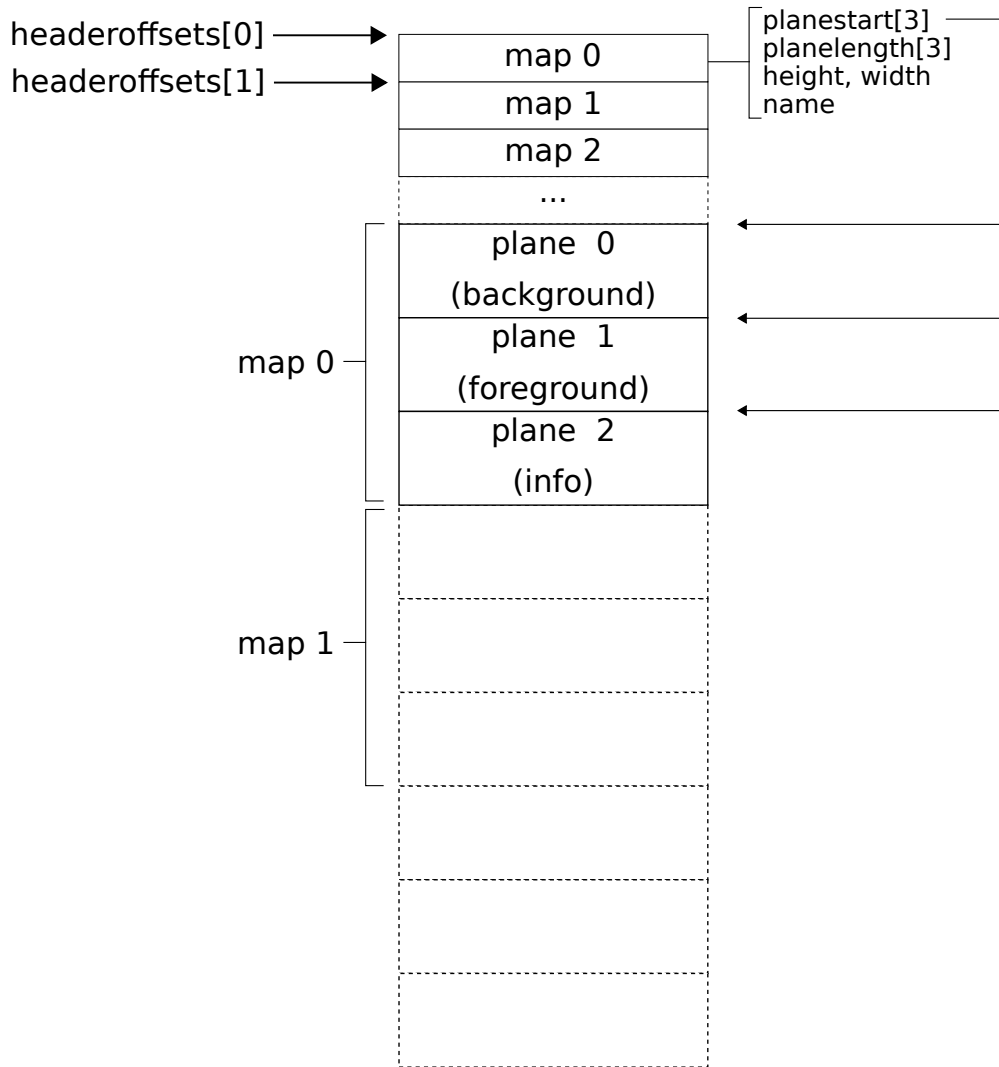
| index | clip north | clip east | clip south | clip west | inside tile |
|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... |
| 238 | 0 | 1 | 5 | 0 | 0 |
| 239 | 0 | 0 | 0 | 0 | 0 |
| 240 | 0 | 0 | 5 | 0 | 0 |
| 241 | 0 | 0 | 0 | 0 | 128 |
| 242 | 1 | 1 | 1 | 1 | 128 |
| 243 | 1 | 0 | 1 | 0 | 0 |
| 244 | 0 | 0 | 2 | 0 | 0 |
| ... | ... | ... | ... | ... | ... |

*Table 4:* Foreground tile clipping and 'inside' tile information.

### 0.3.4   Map file structure

The structure of `KDREAMS.MAP` 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 foreground, background and info.

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

headeroffsets[0] ⟶

headeroffsets[1] ⟶

planestart[3]
planelength[3]
height, width
name

map 0

map 1

map 2

...

plane  0

(background)

plane  1

(foreground)

plane  2

(info)

map 0

map 1

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

## 0.4   Audio