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).

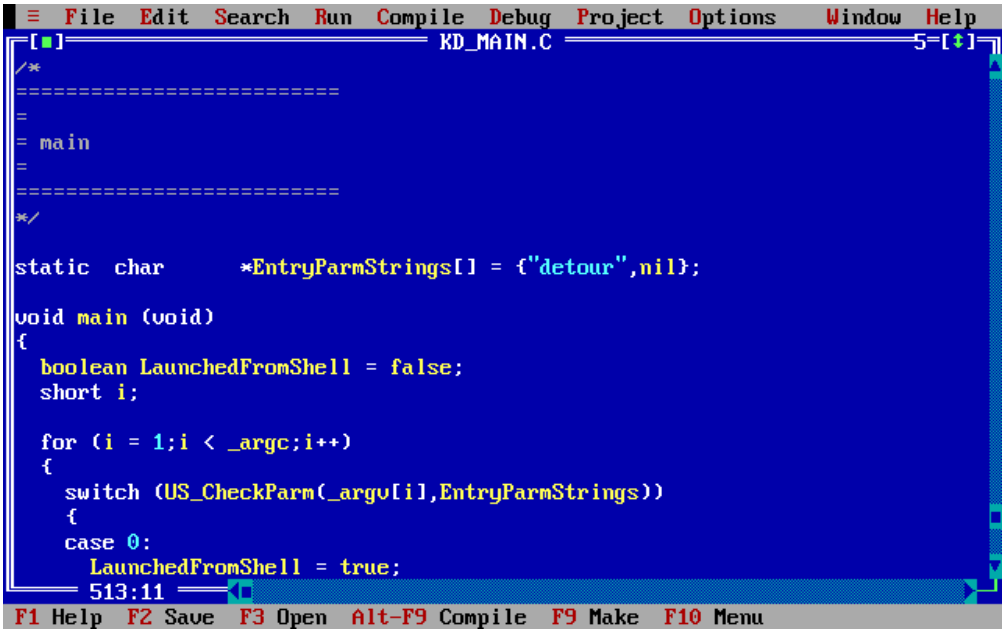


Figure 1: Borland C++ 3.1 editor

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.

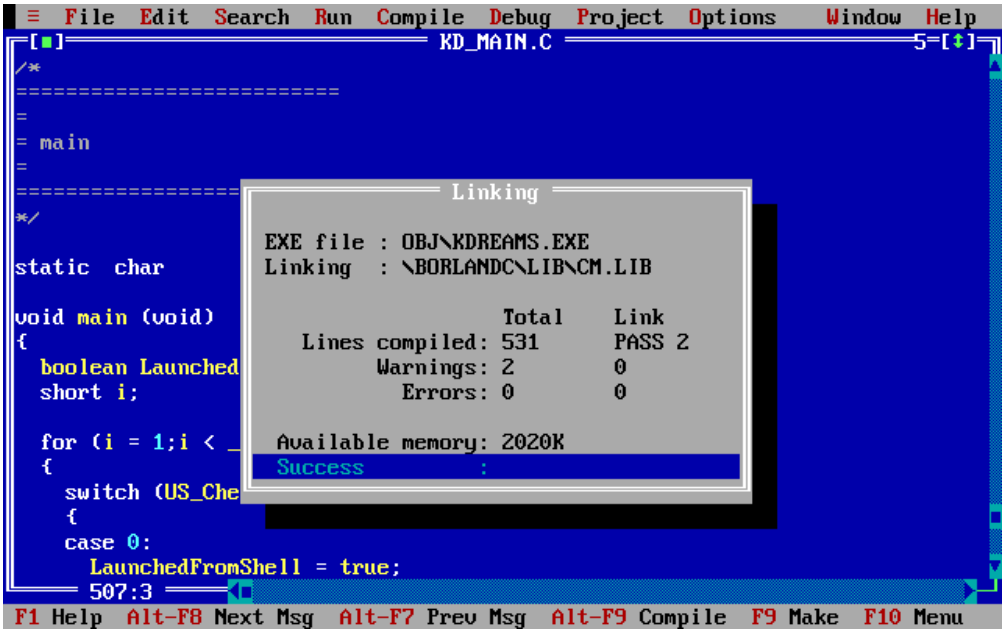
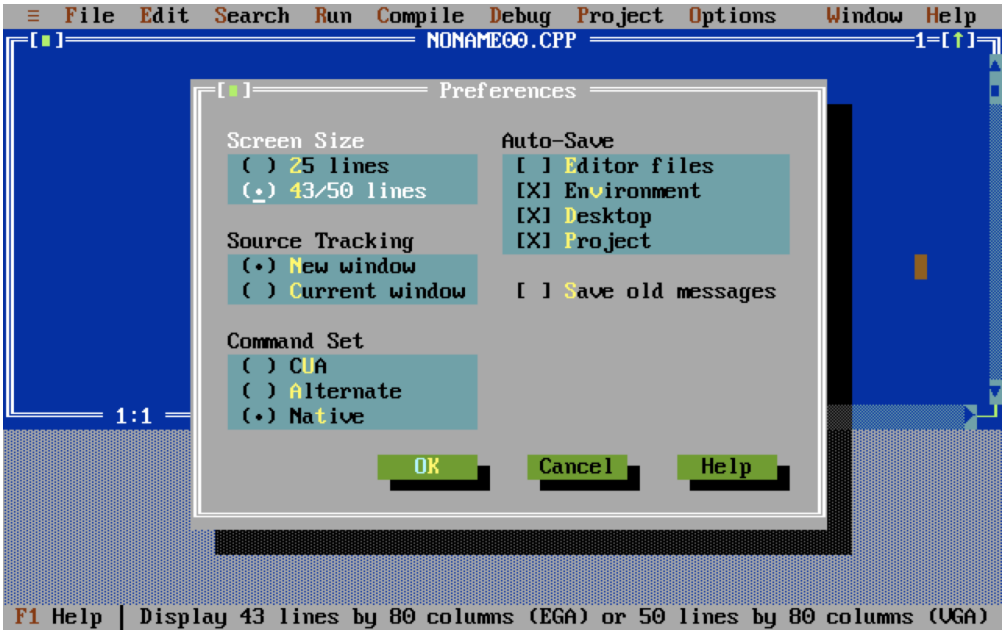
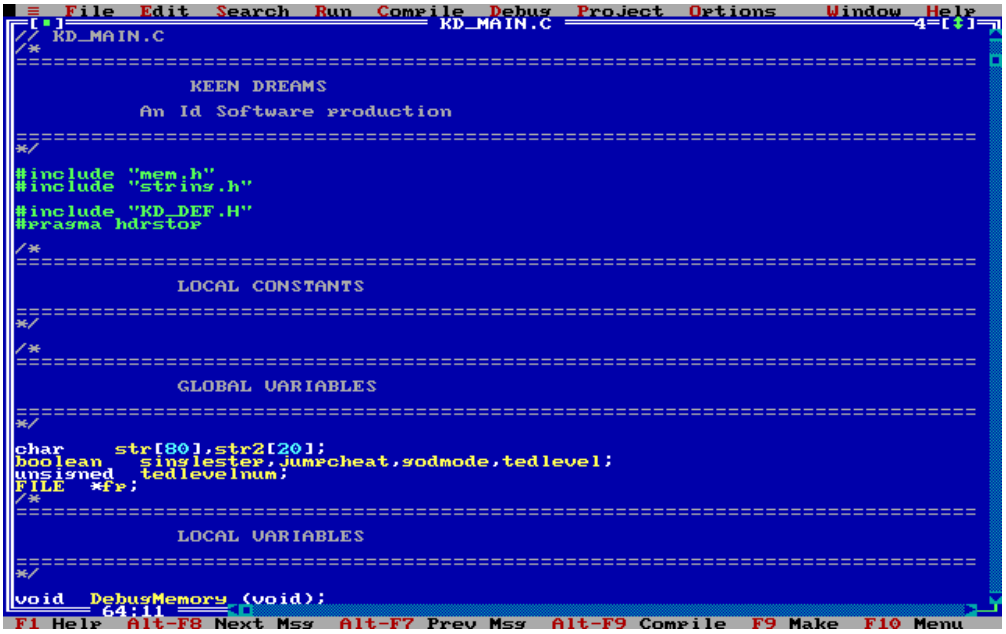


Figure 2: Compiling Keen Dreams with Borland C++ 3.1

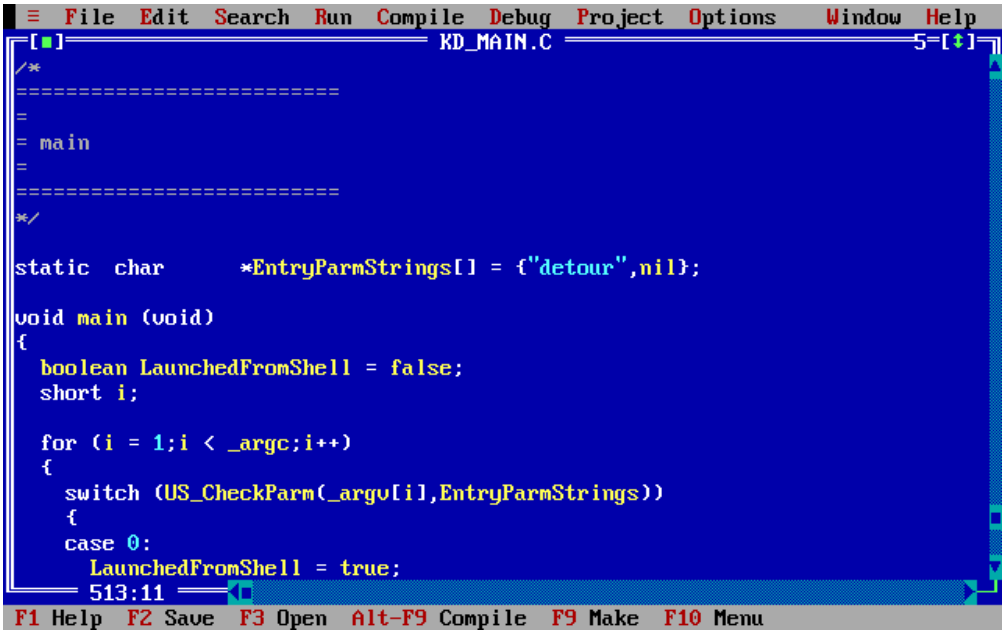
Another way to improve screen real estate was to use "high resolution" 50x80 text mode.

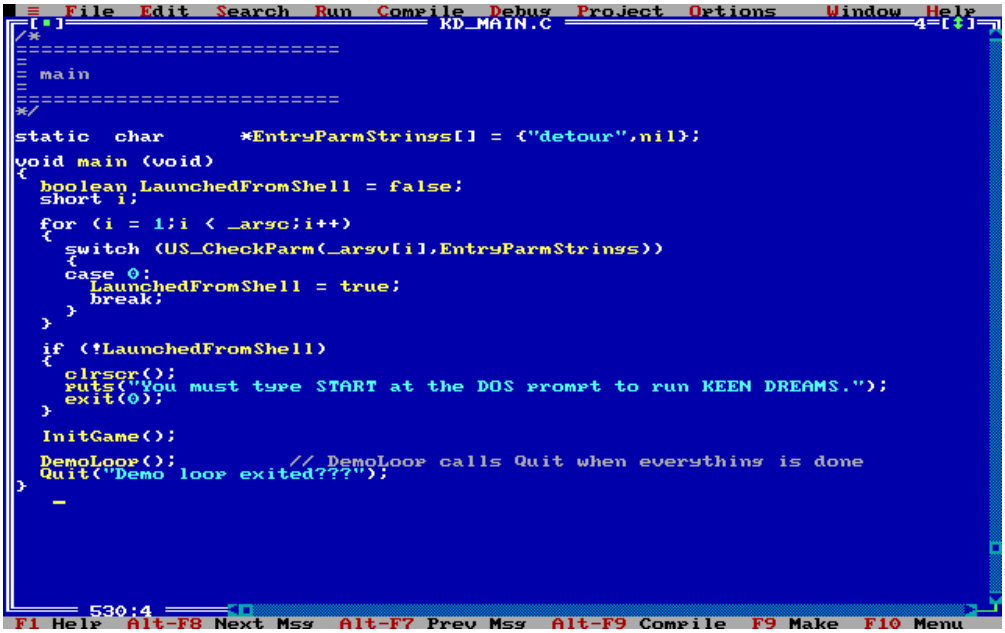


The comments still fit perfectly on screen since only the vertical resolution is doubled.



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


File Edit Search Run Compile Debug Project Options Window Help
KD_MAIN.C
/*
============================
=
= 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


File Edit Search Run Compile Debug Project Options Window Help
KD_MAIN.C
/*
============================
=
= 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???");
}
530:4
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

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.

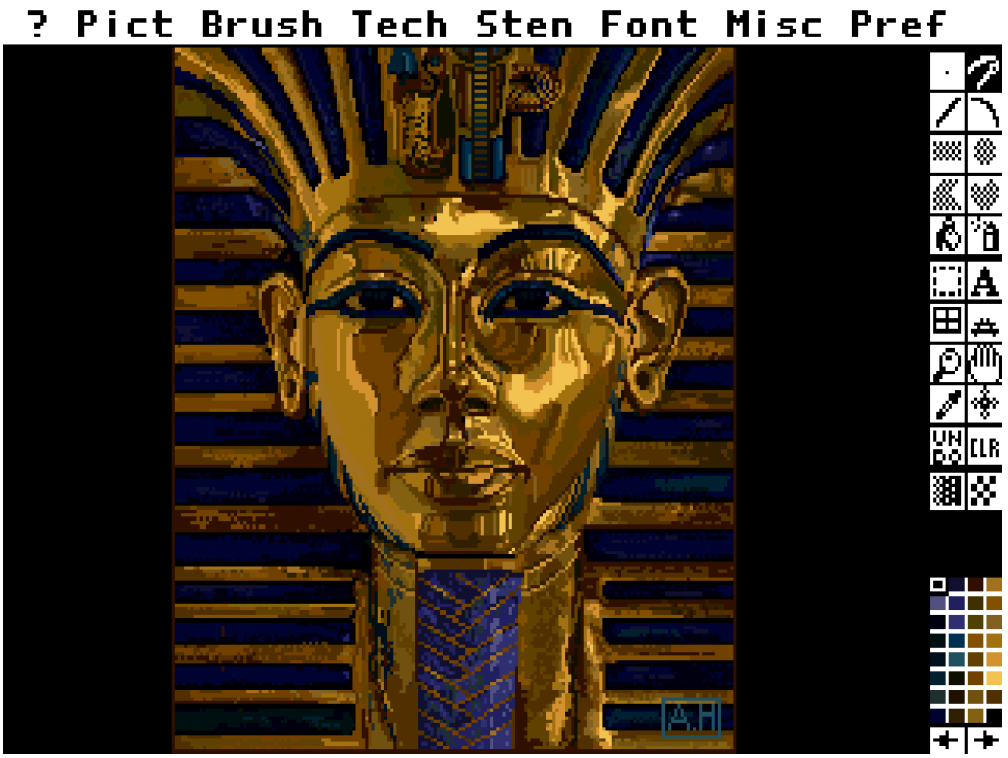


Figure 3: Deluxe Paint was used to draw all assets in the game.

0.3 Assets Workflow

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

[2] InterLeaved BitMap.

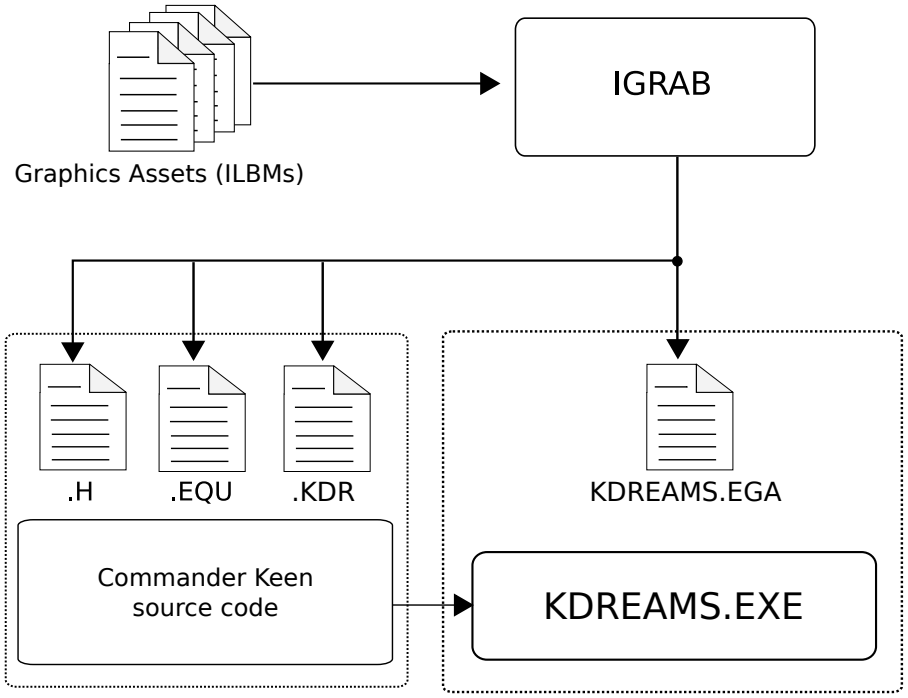


Figure 4: Asset creation pipeline for graphics items

```
//////////////////////////////////////
//
// 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 archive `HEAD` table which gives an offset in the `DATA` archive. The archive header files are stored in the `\static` folder.

Figure xx shows what is inside the `KDREAMS.EGA` asset file

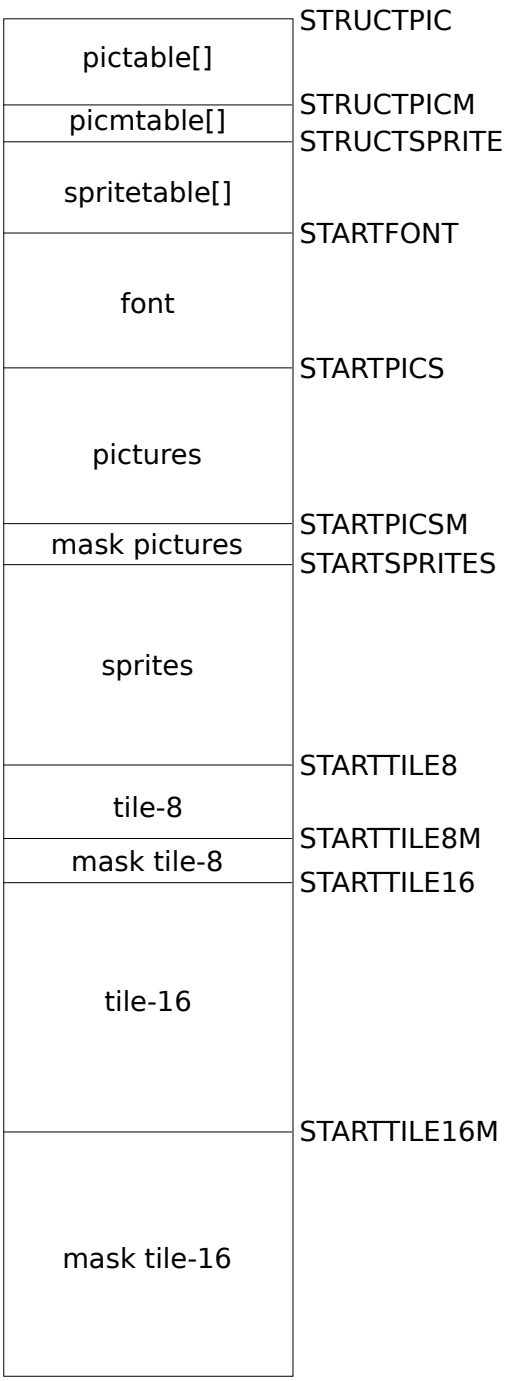


Figure 5: File structure op `KDREAMS.EGA` asset file.

The `pictable[]` contains the width and height for each picture in the asset file. The actual picture data is located in the `pictures` location of the file. The same structure applies for the mask pictures.

index	width	height
0	12	13
0	16	18

(a) pictable[].

Figure 6: Tile clipping map

The `spritetable[]` contains beside width and height also information on the sprite center, boundaries and if a shifted sprite must be created.