

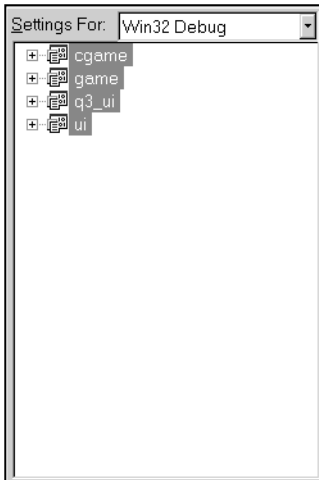


# **APPENDIX A**

# **DEBUGGING YOUR MOD IN VISUAL STUDIO**

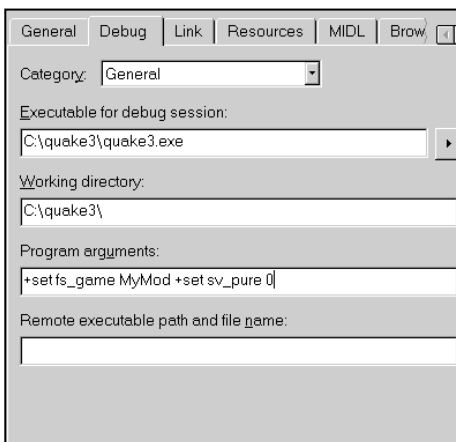
A programmer requires an extensive set of tools to get his job done; working with C can often be a frustrating and confusing process. Bugs that pop up in your code can lead to nightmarish days and endless nights as you try to decipher the simplest of problems. Working with *Q3*'s code base is no different. One of the best tools available to a programmer when working in Visual Studio is the debugger, which lets you stop your program as it runs, watch the values of variables and the flow of logic, locate the dastardly bug, and squash it. Following is a description of how to set up the debugger in Visual Studio so that you can use it for your *Q3* mod development and testing. Start by loading the *Q3* source project into Visual Studio. Then do the following:

1. Open the Project menu and choose Settings. (Alternatively, press Alt+F7.)
2. The Project Settings dialog opens, with the subprojects *game*, *cgame*, *q3\_ui*, and so on visible on the left side, and a tabbed interface on the right. On the left, near the top, select the Settings For drop-down menu and select Win32 Debug.
3. Highlight all the subprojects (*game*, *cgame*, *q3\_ui*, and *ui*) by holding down Ctrl while you click on each subproject (see Figure A.1).
4. On the right side of the Project Settings dialog box, click the Debug tab.
5. You should see a Category drop-down menu; make sure it is set to General.
6. In the Executable For . . . text box, type the full path to your *Q3* executable (or use the right-arrow button next to the text box to browse your hard drive to find it).
7. In the Working Directory text box, type the full path to your *Q3* executable, minus the name of the executable file. So if you said your executable file was at *c:\quake3\quake3*, your working directory should only be *c:\quake3\*.



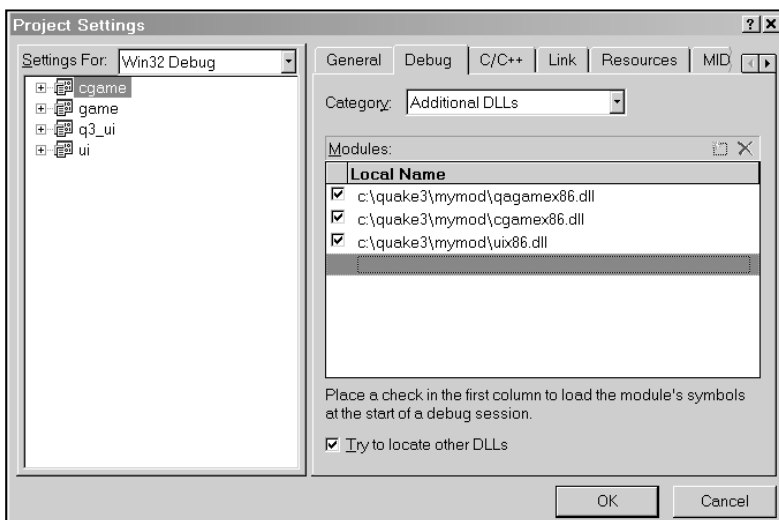
**Figure A.1** *Creating your first IDirect3D8 object*

8. In the Program Arguments text box, type `+set fs_game MyMod +set sv_pure 0`. (You should recognize these parameters as the ones you used during all the development throughout this book; if you use a different subfolder for your specific mod, make the appropriate specification instead of MyMod.) Figure A.2 shows the results of these entries.



**Figure A.1** *Creating your first IDirect3D8 object*

9. In the Debug tab, click the Category drop-down menu and choose Additional DLLs. The text boxes are replaced with a gray box labeled “Modules”; this is where you can specify any DLLs that are required during debugging.
10. You cannot add DLLs to all subprojects at once, so start by highlighting `cgame` on the left. The gray Modules box on the right should now be white, indicating that you can add DLLs.
11. Click the Add a New DLL button (the small dashed box next to the red X to the far right of the Modules label; the red X is the Remove a DLL button).
12. A new text box appears next to a new check box. Type the full path to the specific DLL you want to add (or browse your hard drive to locate it by clicking the button marked with an ellipsis). You should specify all three *Q3* DLLs in your MyMod folder—`qagamex86.dll`, `cgamex86.dll` and `uix86.dll`—as shown in Figure A.3.
13. After all three DLLs are specified, repeat the process for `game`, `q3_ui`, and `ui`.
14. Click the Link tab.
15. Make sure the Category drop-down list is set to General.



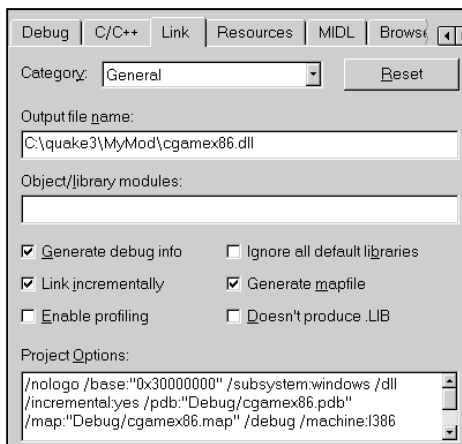
**Figure A.3** Specifying additional DLLs for a project debug session

16. In the Output File Name text box, type the full path to the final DLL that will be built when you compile.
17. Repeat step 16 for each subproject in turn, because they all build to different file names. When you're finished, you'll have specified the full path for `cgamex86.dll` for `cgame`, `qagamex86.dll` for `game`, and `uix86.dll` for both `q3_ui` and `ui` (see Figure A.4).
18. Click the OK button in the lower-right portion of the dialog box.

**NOTE**

Make sure your output path matches your Q3 directory and your specific mod directory, which in this example is "MyMod."

To actually debug your code, you will need to set *breakpoints* in your code, and possibly use *watches* to look at current values of variables. A breakpoint refers to a specific line number within code that stops execution of the program when Visual Studio reaches it. You are able to set multiple breakpoints, if needed. Once a breakpoint is hit, the debugged program pauses, and Visual Studio takes control, highlighting the line of code that execution stopped at. A watch is used to tell Visual Studio that you want to observe the value of a specific variable



**Figure A.4** The link tab, with an output file name specified

as a program executes. You're able to drag-and-drop variables right into the watch window, as well as type them in manually.

To quickly try this out, open `g_items.c` and scroll to line 398, where you should find the opening to the function `Touch_Item`. Set a breakpoint on this function by clicking the Open Hand icon on your Visual Studio toolbar (see Figure A.5), or by simply pressing F9.

The result of this key press is that a maroon dot is placed to the left of the line of code on which you set your breakpoint. Pressing F9 again will remove the dot. With the maroon dot in place, do the following:

1. Start a new debug session by clicking the Go button (see Figure A.6) or by pressing F5.
2. Visual Studio launches *Q3*, as shown in Figure A.7. With a live debug session in progress, load a map and touch an item.
3. The *Q3* window disappears, and Visual Studio takes over, using a bright yellow arrow to point to the line in your code where the program has paused. You can now step through the code, line by line, by clicking the Step Into button (see Figure A.8) or by pressing F11.

### TIP

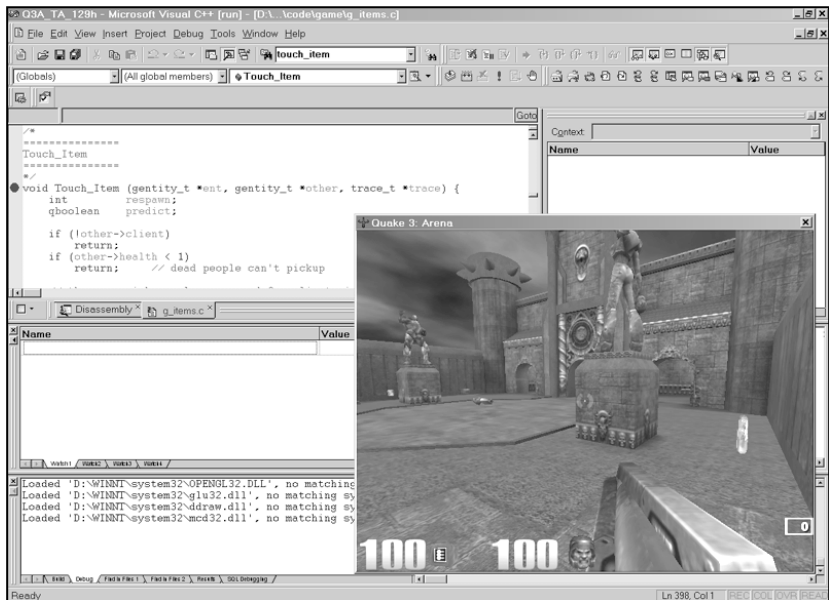
I highly recommend going to the Q3 system settings and disabling full-screen mode so that you can debug Q3 while it runs in a window. This will make switching back to Visual Studio extremely easy when a breakpoint is hit. You can toggle the full-screen option in Q3's System menu, under Graphics.



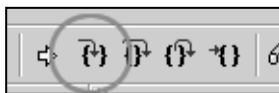
**Figure A.5** *The Breakpoint button*



**Figure A.6** *The Go button*

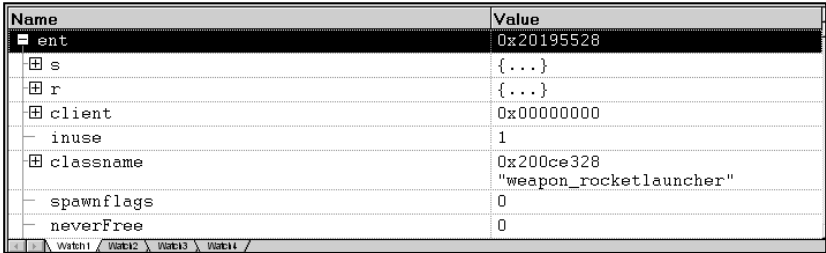


**Figure A.7** A Q3 debug session in action



**Figure A.8** The Step Into button

4. To examine the specific values of variables during this pause in the execution of *Q3*, simply highlight it and drag it into the Watch debug window. (To make this window visible, press Alt+3.) As shown in Figure A.9, the Watch debug window has two columns, *Name* and *Value*, which refer to the variable and its contents. Because *Touch\_Item*'s first input parameter is *ent*, which should represent the touched item in question, select the word *ent* and drag it to the Watch window. It should populate into a tree hierarchy, which you can break out by clicking the + sign next to each complex variable.
5. When you are ready to return to *Q3*, simply click the Go button or press F5, or stop debugging altogether by opening the Debug menu and choosing Stop Debugging or pressing Shift+F5.



The screenshot shows the 'Watch' window in Visual Studio. It contains a table with two columns: 'Name' and 'Value'. The 'ent' variable is expanded, showing its properties: 's' (empty), 'r' (empty), 'client' (empty), 'inuse' (1), 'classname' (0x200ce328, 'weapon\_rocketlauncher'), 'spawnflags' (0), and 'neverFree' (0). The 'Watch' tab is selected at the bottom.

Name	Value
ent	0x20195528
s	{...}
r	{...}
client	
inuse	1
classname	0x200ce328 "weapon_rocketlauncher"
spawnflags	0
neverFree	0

**Figure A.9** *The Watch debug window*

I encourage you to spend time getting to know the debugger and reading up on the subject, if possible. There are many more details that the debugger can manage; if wielded properly, it is a powerful tool for fixing code. A good starting point is Microsoft's MSDN website, which offers detailed information on Visual Studio's debugger, at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebug/html/vchowViewingModifyingData.asp>.