Chapter 3

Game Boy Development Tools

Console development was always a difficult process in the past, requiring custom equipment and a special version of the console designed to run programs through a link cable attached to a special cartridge. These hardware kits were supplemented with custom software, along with a compiler, an assembler, and a linker, designed to generate binaries, another term for executable game code, to run specifically on that one console.

The manufacturer of a console designs the development kit, hopefully without making it too difficult to use, in order to attract developers to write games for the console. This chapter explains what home-brew solutions are possible and what hardware and software you will need to write Game Boy Advance programs without an expensive software development kit (SDK) or hardware interface.

Here is a list of subjects you will find discussed in the following pages:

- Game Boy Advance development
- Installing the development tools
- Introduction to the HAM SDK
- Running Game Boy programs

Game Boy Advance Development

Let's come to the point: The Game Boy Advance is an amazing little machine. You are eager to get started writing code, right? Early on in the Game Boy Advance timeline, it wasn't even possible for a hobbyist to write a Game Boy Advance program. Let's digress for a minute to explore how the grassroots Game Boy Advance (GBA) development community produced the tools that made this book possible.

Enterprising programmers were writing GBA programs at least six months before Nintendo released the first GBA unit. How, do you suppose, could that possibly have been done? It's an amazing story, actually. Console and video game fans have been writing emulators for years now for everything from classic coin-op arcade games. The popular emulation program MAME, short for *Multiple Arcade Machine Emulator*, can emulate any old-school arcade machine from the pre-3D era. The same talented group of programmers (I'm generalizing here) who write MAME and many other emulators got started on several GBA emulators very early—some as long as two years before the GBA came out! (How's that for unkempt anticipation?) An emulator is generally developed with the help of several ROM images, such as from a GBA game demo leaked from a third-party developer. Emulator programmers will reverse-engineer these ROMs by examining the codes inside a ROM.

Deconstructing The ARM7 CPU

Here is where things get interesting! Using the specifications of the ARM7 processor, a programmer can deduce what the binary codes inside a ROM are supposed to do. Each byte inside a ROM is significant and represents a binary instruction or piece of data. Since the ROM includes all the graphics and sound effects for the game, some sections of data are reserved in the ROM for data of this type. There are other kinds of data stored in a ROM as well, such as game levels, character stats, and even cheat codes. All of this information is stored in the ROM as a single "image" that is written to a ROM chip stored inside each game cartridge. The emulator will open the ROM file, read the image into memory, and start processing it. The task of the emulator programmer, early in the project, is to decipher the codes stored in the ROM of any new hardware system being emulated, including the GBA. Once the programmer has figured out how the ROM is divided up between instructions and data, the next step is to decipher the bytecodes, each of which represents a single instruction from the ARM7 instruction set.

I won't get into any assembly language so soon, but you may jump ahead to Chapter 11, "ARM7 Assembly Language Primer," if you are curious about it at this point (although I would recommend waiting). Some instructions consist of just 1 byte, while others take up 2, 3, or 4 bytes each in the ROM. The GBA hardware boots up by executing a single instruction at a fixed point in the ROM, which then starts the game running. There is no operating system, no special interrupts, no built-in code libraries—nothing. In a console, all the code is built into the ROM and there is no operating system, although that statement is only relevant to smaller and/or older consoles. Newer consoles do have an operating system built in, because most modern consoles are capable of doing more than just playing ROMs. For example, optical media consoles feature some sort of menu system in order to play CD music or DVD movies. You get the point, right?

Emulation: The Key to Reverse Engineering

The early GBA hobbyists gradually gained experience with Nintendo's proprietary as-yet-unreleased handheld video game console, and some very good emulators were developed long before the actual hardware was released. The legitimate hobbyists do the work of deciphering the ROM the hard way, without accepting any tips from licensed third-party developers, because it is a matter of pride, and the challenge of cracking the code, so to speak, is the whole point. Emulator programmers are among the most talented low-level software engineers in the world; they provide a great service to video game fans. Without emulation, most old arcade games would be lost forever, because the original arcade cabinets are usually not preserved over time. They are recycled for newer games, or they simply break down as the years pass and are then discarded as irreparable. Emulators allow us to play the great classics on a PC monitor; they even support gamepad-style joysticks.

What is the final recipe for the elite emulator programmer? The flash linker. I'll explore this fascinating hardware interface later in this chapter. Suffice it to say, there's no better way to perfect an emulator than by using a real game ROM. Using a flash linker, an emulator programmer can download his or her actual retail game cartridges through a special cable to the PC. Once a real ROM is available, it is then possible to fix all the quirks in the emulator. After all, with the real hardware and game available as a comparison, one can simply tweak the emulator until it runs the virtual ROM just like the real GBA hardware runs the actual cartridge.

Unfortunate Side Effects

Unfortunately, some unscrupulous individuals (that's geekspeak for *losers*) abuse the intended purpose of the emulators and pirate the game ROMs. I understand doing that with video games that are no longer in production (such as MAME ROMs). But Game Boy ROMs are still being sold in retail channels. Don't pirate retail games! Emulators such as VisualBoyAdvance were designed for running *new* code for the GBA, not *existing* code. Recognize that distinction! Yes, there are many professional GBA developers who are using tools like Visual HAM, Hamlib, and VisualBoyAdvance for actual retail GBA games. These tools might be in the public domain, but they are excellent tools that rival the official Nintendo SDK (in some respects). There are many GBA cartridges in stores today with code that was compiled with the very same compiler that you will use while reading along in this book! How do the professionals (and the hobbyists) test their new programs? Emulators.

Know the distinction, and preach it when you can. If you love your GBA, then discourage game piracy when you have an opportunity. Why should you care? It's a tradeoff that console manufacturers weigh when designing a new video game machine. In the case of the Dreamcast and GBA, for instance, an easier programming model was deemed helpful in order to attract third-party developers (such as id Software, Electronic Arts, Activision, Capcom, Square, etc.). Some consoles have a very difficult SDK to master, in comparison, such as Sony's consoles and Sega's early consoles. Some consoles, such as the Sega Genesis, Sega Saturn, and Nintendo 64, were legendary in programming circles for their difficulty. Piracy is usually negligible on closed systems during the peak sales period (the first four years of the console's life), while more open systems are cracked earlier. In the case of the GBA, it was cracked before it was even released! That was certainly not a tragedy; on the contrary, it shows the fervent zeal that fans have for the Game Boy systems overall. You are reading this book because you love your GBA, right? Or perhaps you are a professional

I do not condone the act of downloading game ROMs, whether they are owned or not. I simply disagree with the whole concept, plain and simple. Flash cartridges are prohibitively expensive for making backups of game cartridges, therefore this activity is a slippery slope that cannot be justified. I urge you to maintain a level of professionalism in this matter and purchase any and all games that you enjoy playing on your GBA. Let's face it, if you are reading this book, then you must have an interest in professional GBA development. Please don't undermine the company that created this fascinating and enjoyable machine; cartridge sales are the sole basis of income in the handheld market.

developer. Either way, you must admit that this is a fun system to play as well as to develop for.

What About Public Domain Development Tools?

I have only hinted about the development tools up to this point, so you know what they are, without really knowing any of the specifics. Later in this chapter, I will show you how to install and begin to familiarize yourself with HAM, the tool used in this book. If you are a professional developer and already using an official Nintendo SDK or another distribution such as DevKit-Advance, I challenge you to give HAM a whirl. As I have explained, Visual HAM is able to handle code designed for DevKit-Advance (or any other GBA toolkit based on the GNU compilers), and the editor and environment are very nice, with an almost obsessive amount of support by the authors of Visual HAM and the HAM SDK.

Visual HAM And The HAM SDK

As I mentioned in Chapter 2, there are some compilers and assemblers available for the GBA, because it uses the common ARM7 processor, which is also used in hundreds of other consumer electronics devices. This was a good move on the part of Nintendo, as far as keeping costs down by using a powerful and mass-produced processor. Nintendo obviously learned a lesson from the Nintendo 64, which used a custom chipset developed by Silicon Graphics. The Nintendo GameCube, for instance, uses an IBM PowerPC processor with an ATI GPU (graphics processing unit) that are almost identical to the retail processors available in stores! Likewise, the GBA uses an ARM7 processor and features the secondary Z80 processor for backward compatibility. Nintendo opted for familiar and proven technology this time around with both of its latest consoles. The only problem—for Nintendo, that is—is that the widely available tools for the ARM7 processor have made it possible for enterprising programmers (which is geekspeak for *hackers*, in case you weren't paying attention) to quickly adapt the existing tools for the GBA.

This is where things get interesting, because that is precisely what they did. The end result of all the effort that went into the early GBA development toolkits is HAM, by Emanuel Schleussinger, and Visual HAM, by Peter Schraut. These next-generation GBA tools incorporate programs written by several people, providing (in addition to VisualBoyAdvance) a graphics file converter and a complete sound system! (For more information about GBA sound, refer to Chapter 9, "The Sound System.") The one thing I want to emphasize is that you can use Visual HAM and the HAM SDK to compile and run any

GBA program based on GCC, including games developed under DevKit-Advance (another GBA distribution).

What Is HAM All About?

HAM is a distribution package that includes all the tools (including the C/C++ compiler) needed to write, compile, link, and run GBA programs. The term *distribution* comes from the fact that HAM uses the open source GCC compiler (which has a GNU license), along with an ARM7 assembler that was released to the public domain by the chip manufacturer, ARM. The HAM distribution is included on the CD-ROM in complete format that you may install to your hard drive. This includes all of the following tools:

- HAM Game Boy Advance SDK
- HAM code library
- Visual HAM development environment
- PE Map Editor
- VisualBoyAdvance emulator

These are all the tools you need to write fully compliant GBA programs that will run on the actual handheld. These tools were not written by one person, of course, but were developed by hundreds of programmers who have dedicated their skills to the open source community, which has made HAM possible. Despite the large number of people involved in GCC and the various GBA tools, the HAM SDK and HAM library were developed by Emanuel Schleussinger. Likewise, the Visual HAM development environment and PE Map Editor were written by Peter Schraut. I am grateful for their consent to feature these excellent tools in this book.

Visual HAM and the HAM SDK are capable of compiling any GCC-compatible GBA source code. There is no stipulation that you *must* use the HAM library (Hamlib) in your programs. Most of the code in this book is stock GBA, not Hamlib.

HAM on the Web

The Web site for HAM (http://www.ngine.de) is shown in Figure 3.1, while the Web site for Visual HAM (http://www.console-dev.de/visualham) is shown in Figure 3.2. I encourage you to visit the sites to get the latest versions of the development tools used in this book. As the development community tends to be a very dynamic and persistent medium for change, there will likely be new versions of the tools every few months. The version used in this book is HAM 2.7.

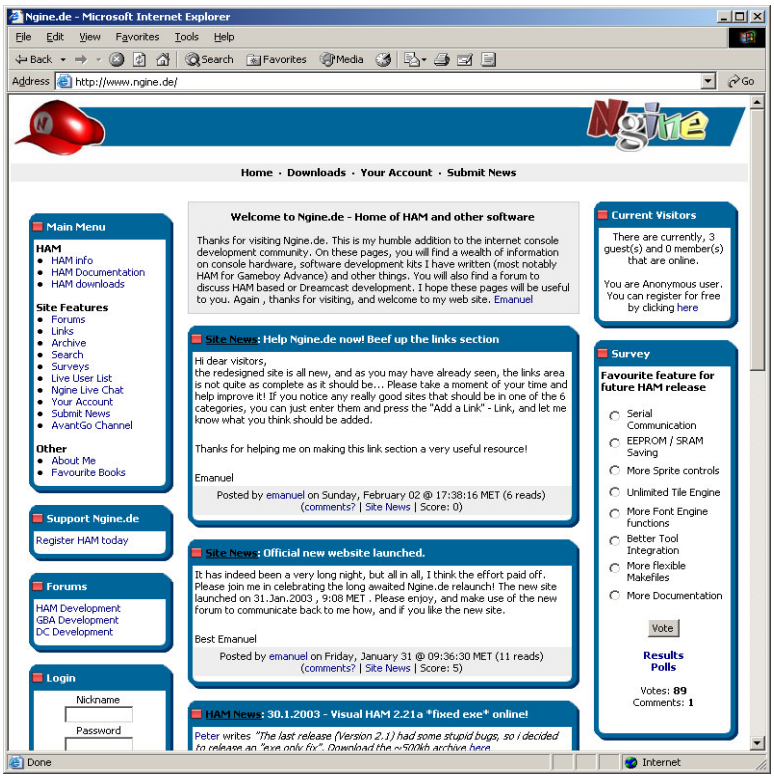

Figure 3.1

The Web site for HAM is located at http://www.ngine.de.

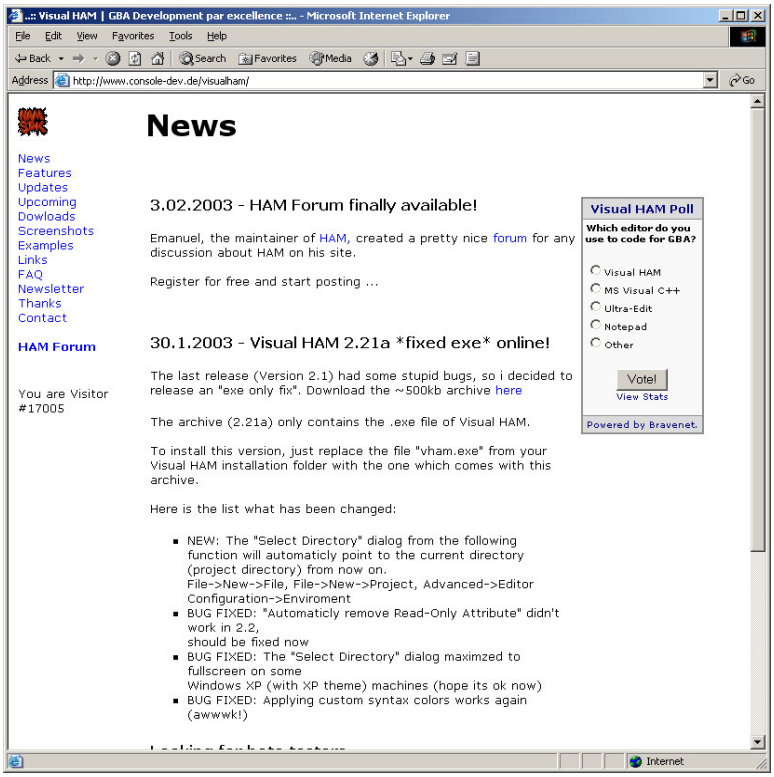

Figure 3.2

The Web site for Visual HAM is located at http://www.console-dev.de/visualham.

The Visual HAM Environment

Visual HAM was not developed by the same person who created HAM (Emanuel Schleussinger), although Peter Schraut works closely with him . The end result is a seamless package (see Figure 3.3) that works so well that you will wonder why these tools are free! The exception is the HAM library (Hamlib). While you can write complete GBA programs without Hamlib, the library is a great help, because it encapsulates the whole GBA SDK within itself and handles much of the hard work for you, behind the scenes. I will show you in the next chapter how to write your first GBA program, with and without Hamlib. Just as an example and case in point, the source code shown in Figure 3.3 is just plain, simple GBA code, without any wrapper. (This program is actually a prototype of the graphics mode 4 pixel-plotting program that you will encounter again in chapter 5!).

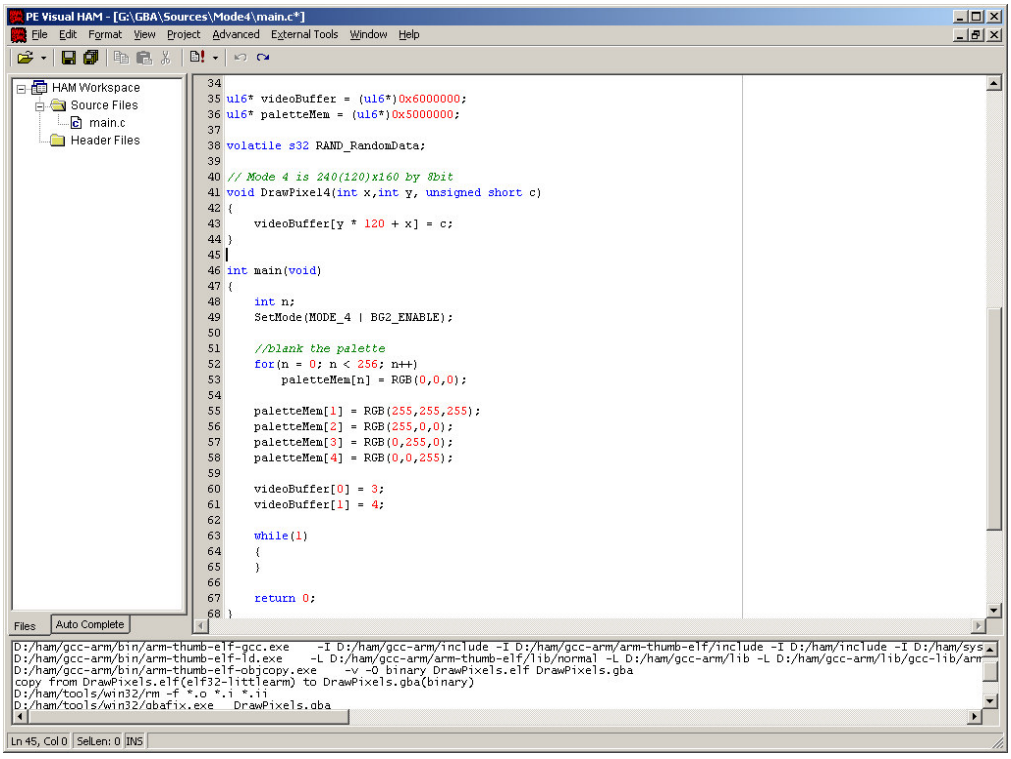

Figure 3.3 The Visual HAM integrated development environment.

It is important to know how to write core code without using any wrapper (which Hamlib provides), especially while learning. Although I use Hamlib to demonstrate how things work in the next few chapters, it is possible to use Visual HAM (the editor and integrated development environment) and HAM SDK to write non-HAM programs. I have downloaded several public domain GBA games written by fans, have loaded the source code into Visual HAM, and was able to compile and run the programs just fine, with no problems! The programs run in the VisualBoyAdvance emulator or on an actual GBA using a multiboot cable

(more on that in the next chapter!). In addition, Visual HAM comes with an excellent map editor that is useful for creating game levels, as shown in Figure 3.4.

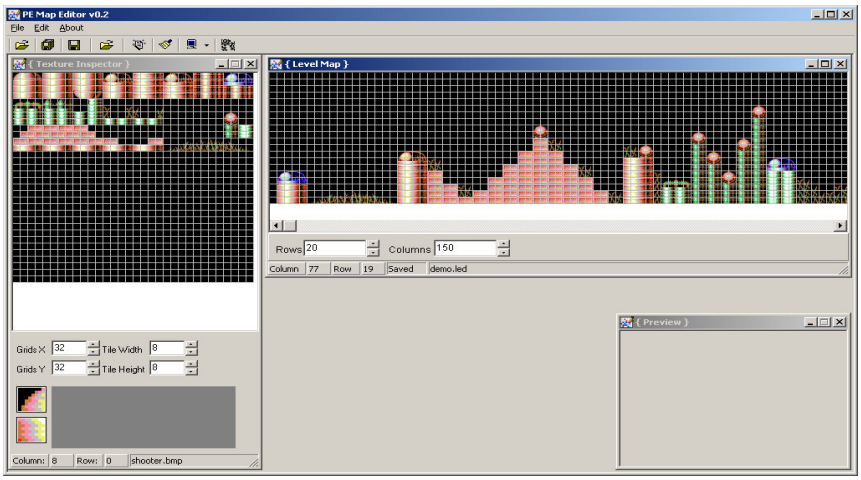

Figure 3.4
Visual HAM comes with a map editor for designing game levels.

Installing the Development Tools

The most remarkable thing about the HAM distribution is just how easy it is to install and use. HAM was designed to be a complete one-stop solution for writing and testing GBA code. As such, it comes equipped with everything you need to write, edit, debug, and run programs with several options for the output. HAM comes with an excellent GBA emulator called VisualBoyAdvance. Although Emanuel has packaged the emulator with HAM, you can still download VisualBoyAdvance from http://vboy.emuhq.com; be sure to check these links from time to time for updates to the tools. Although I have included the very latest on the CD-ROM, these tools are updated and improved over time by the GBA community.

Installing HAM

The downloadable zip file version of HAM has a file name something like ham-2.70-win32-full-distro.zip (note that the version number may change). Extracting the zip archive reveals two folders (ham-installer-files and ham-other), along with the setup file, setup.exe. The zip is available on the CD-ROM in the \HAM folder. Double-clicking on the setup.exe file starts the HAM installer, which is shown in Figure 3.5.

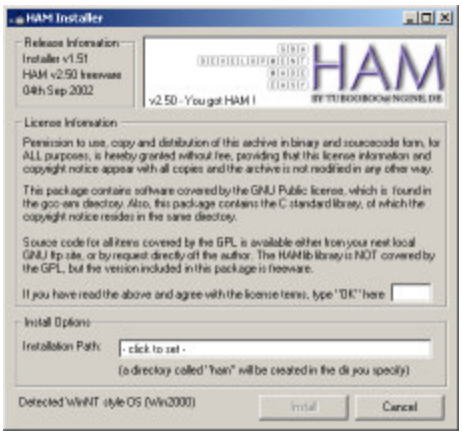

Figure 3.5

The HAM installer is a simple dialog-style window.

After typing "OK" in the box as requested, click in the Installation Path text box to bring up a folder selection dialog box. Select a hard drive where you would like HAM installed. Be sure to just select a root folder, not any subfolder, because HAM only works off the root.

HAM must be installed on the root of your hard drive! Although it will work fine on any hard drive (C, D, etc.), it must be on the root. For example, C:\HAM.

After selecting the pathname where HAM will be installed, the window should look like the one shown in Figure 3.6.

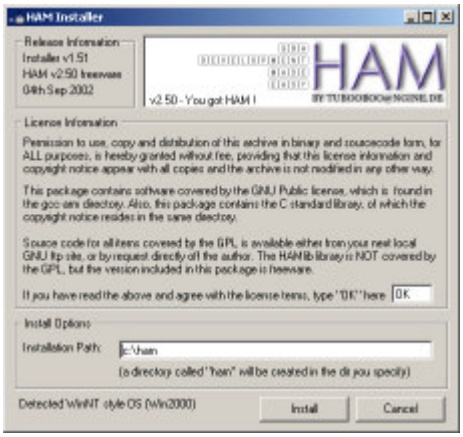

Figure 3.6

The installer dialog box now shows the target folder where HAM will be installed.

The HAM installer usually takes only a minute or two to install, after which the window shown in Figure 3.7 appears.

Figure 3.7

HAM has been successfully installed.

Installing Visual HAM

You may download Visual HAM from this Web site: http://visualham.console-dev.de. Or you may, of course, just install it from the CD-ROM that came with this book. Visual HAM doesn't require any special installer. It just comes as a simple Zip file, although it must be extracted with full folder structure intact. I have provided both the Zip file and the extracted Visual HAM folder on the CD-ROM. You may simply unzip the file to your hard disk drive or copy the folder from the CD-ROM, located at \Visual HAM. Note that the HAM SDK should be installed first, because Visual HAM looks for the HAM SDK when it first runs. Visual HAM is not very useful without the HAM SDK, because that contains the compiler, the assembler, the linker, and so on. Visual HAM is just the IDE, the attractive front end for these command-line tools.

So, assuming that the HAM SDK has been installed to C:\HAM, I would recommend copying the \Visual HAM folder from the CD-ROM to C:\HAM. That way, Visual HAM will be stored conveniently inside the HAM folder at C:\HAM\Visual HAM. Of course, there is no restriction to Visual HAM like there is with the SDK, so you may copy it to the root if you wish under C:\Visual HAM. What really matters is that you can easily find the VHAM.EXE file contained inside this folder, because I want to show you how to create a shortcut to it for your Windows desktop (a shortcut doesn't come with Visual HAM). After you have copied Visual HAM to your hard disk drive, minimize all applications and then right-click on your Windows desktop (anywhere) to bring up the right-click menu. Select New, Shortcut from the pop-up menu (as shown in Figure 3.8).

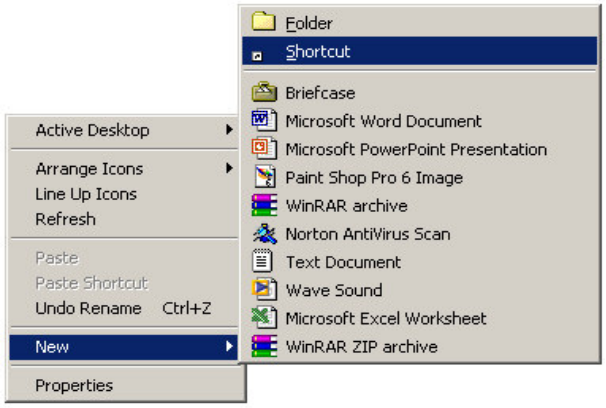

Figure 3.8

The right-click menu from the Windows desktop is used to create a shortcut.

The Create Shortcut dialog box will appear. Click on the Browse button and locate VHAM.EXE inside the Visual HAM folder that you copied to your hard disk drive from the CD-ROM (for example, see Figure 3.9).

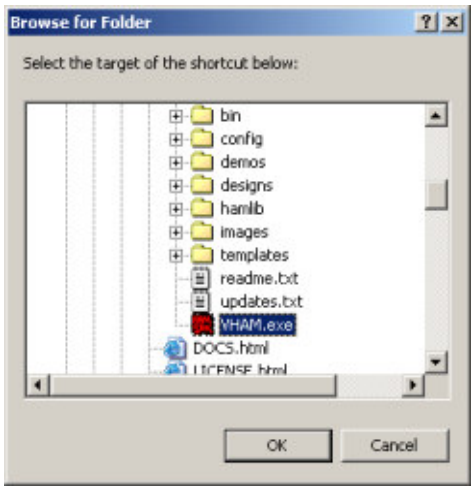

Figure 3.9
The shortcut browser is used to locate the VHAM.EXE file.

The selected executable file is pasted into the location field of the dialog box, as shown in Figure 3.10.

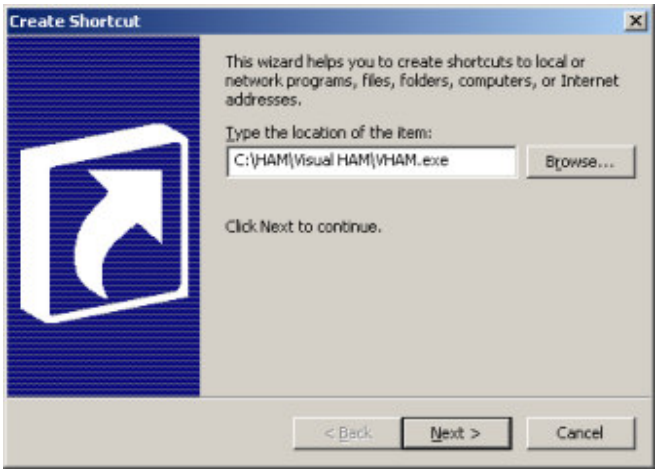

Figure 3.10
The executable file name is now shown in the location text box of the Create Shortcut dialog box.

Next, you can select a title for the new shortcut. I have entered the title "Visual HAM - Game Boy Advance IDE", as shown in Figure 3.11.

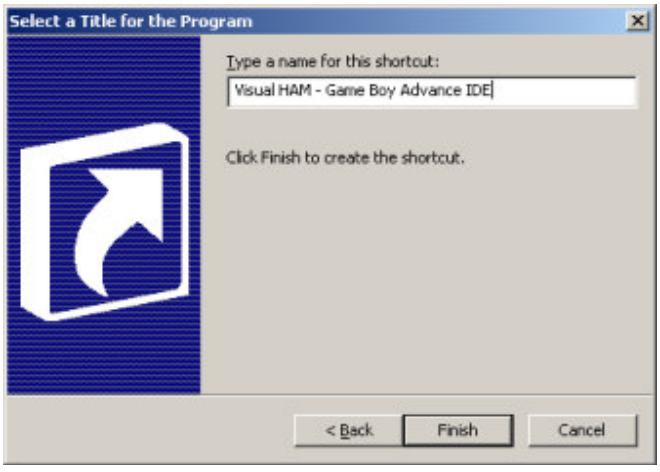

Figure 3.11
Typing in the title for the new shortcut to Visual HAM.

You now have a shortcut on your desktop for starting the Game Boy Advance development kit! Just double-click on the icon to open Visual HAM. The first time it runs, you will see a message stating that the program has configured itself and needs to be restarted. Just close Visual HAM, and then run it again, and you'll be all set and ready to go.

Configuring Visual HAM

The HAM SDK comes with a batch file called startham.bat that sets up the correct path for running the various HAM command-line utilities, as well as the compiler, assembler, and linker. You will need to run startham.bat anytime you use a command-line tool (such as the GFX2GBA.EXE program that you will be using frequently in coming chapters). However, for normal code editing and running of programs, you don't need to run the startham.bat file, because Visual HAM knows where all the tools are located. When you first install the HAM SDK, the installer creates a simple entry in the Windows Registry that specifies the version number and location of the HAM SDK. I will show you how to load a project into Visual HAM and run it in the next section of this chapter.

Once you have restarted Visual HAM, you are presented with a blank screen, as shown in Figure 3.12.

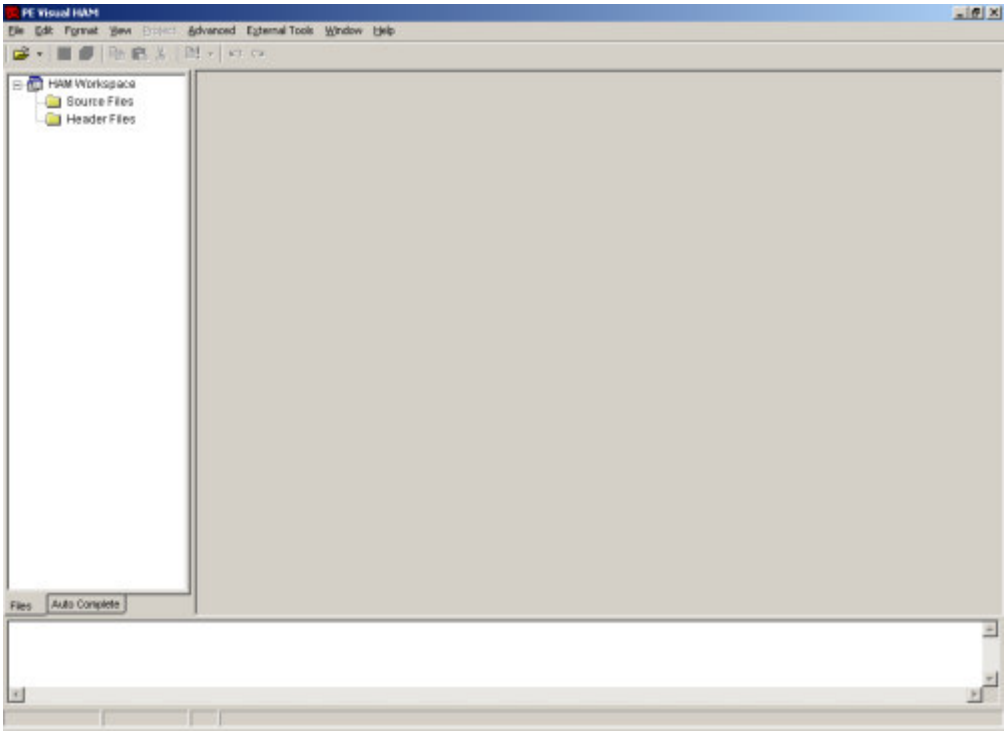

Figure 3.12
Visual HAM just shows a blank editor window and empty project when started for the first time.

Now, let's make sure everything is configured properly. Visual HAM has a dialog box that shows information about HAM. To bring up this screen, which is shown in Figure 3.13, open the Help menu and select Info about recent HAM installation. Note that in this case, it shows an install path of G:\ham\. I happened to have an unused hard disk drive partition that I decided to dedicate to Game Boy Advance programming, so that is why it shows the G drive here. Your installation will look different (most likely C:\HAM\). Take note also of the version, which is shown as 2.52 here. The version is arbitrary, and will change often as the development tools are improved (which is fairly consistently in the online community).

Version numbers change sometimes on a weekly basis in the development community, so just make sure you are using the latest versions of everything by browsing the HAM SDK and Visual HAM sites frequently, as well as the Web sites for support tools like VisualBoyAdvance and GFX2GBA. You will find the complete list of Web sites for these tools in Appendix B.

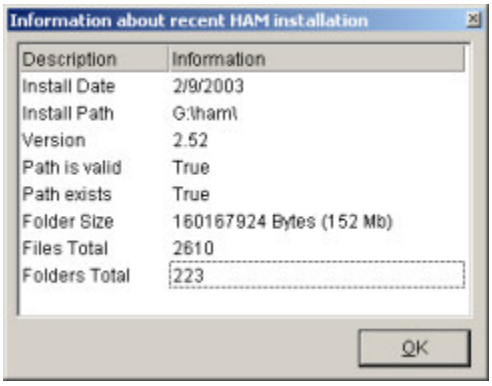

Figure 3.13

This dialog box shows information about the recent HAM installation, including version number.

Now, there is one more thing that I would like you to do, and then you will be finished with the installation. If you opened the Help menu a moment ago to display the HAM version information, you should have noticed a menu item called Associate 'vhw' Filetype. This is a very useful thing to do, so go ahead and click on that menu item. Nothing will be displayed, but Visual HAM actually just added an entry to your Windows Registry associating itself with all .vhw files (which stands for Visual HAM Workspace). You may now double-click on any workspace file directly from Windows Explorer to open Visual HAM. Trust me, you will be doing this a lot as time goes on, because it's far easier to double-click on a workspace file than to run Visual HAM, click on the Open icon, search for the correct folder, and then open the workspace. That's two clicks versus about seven.

Running Game Boy Programs

To demonstrate the capabilities of the HAM SDK and Visual HAM IDE, I've written a short program that displays a picture on the GBA screen. I will explain how to run this program in the VisualBoyAdvance emulator.

For starters, let me show you how to open and run this program, called ShowPicture, in Visual HAM. Please don't be concerned with the source code at this point. Although it's a short code listing, this program is for demonstration purposes only. I will explain the ShowPicture program in the next chapter (along with several more sample programs!).

First, I'll assume you have Visual HAM and the HAM SDK installed. Next, you will need to access some files on the CD-ROM that came with this book. Look for a folder called \Sources\Chapter3. Inside this folder you will find the project ShowPicture. If you haven't already, you will want to copy the project from the CD to your hard disk drive. (I recommend just copying the entire \Sources folder to your hard disk drive in order to gain access to all the projects in one fell swoop—be sure to turn off the read-only attribute on the folder and all subfolders and files.) This is necessary because Visual HAM (and the compiler) need to update files for the project as it is compiled and run. You can run the program directly off the CD-ROM by opening ShowPicture.gba, or by opening the ShowPicture.vhw (Visual HAM Workspace) file in Visual HAM.

If Visual HAM ever seems to stop compiling programs correctly, in that it no longer invokes the compiler, linker, and so forth, then most likely it is just not hooked up to the HAM SDK (for example, if you moved the HAM folder). There is an easy way to fix this. From within Visual HAM, open the Advanced menu and select Editor Configuration. Select the Environment tab, and then click on the Auto Detect button. Visual HAM should be able to locate the HAM SDK (using the Windows Registry) and fill in the root and path for HAM. This feature is likely to change in future versions.

Now that you have the ShowPicture project loaded into Visual HAM, let's try to compile the program. First, refer to Figure 3.14 to see what the project should look like on your screen. If Figure 3.14 looks the same as what's on your screen, then let's proceed. Otherwise, you may want to double-check the project you have loaded.

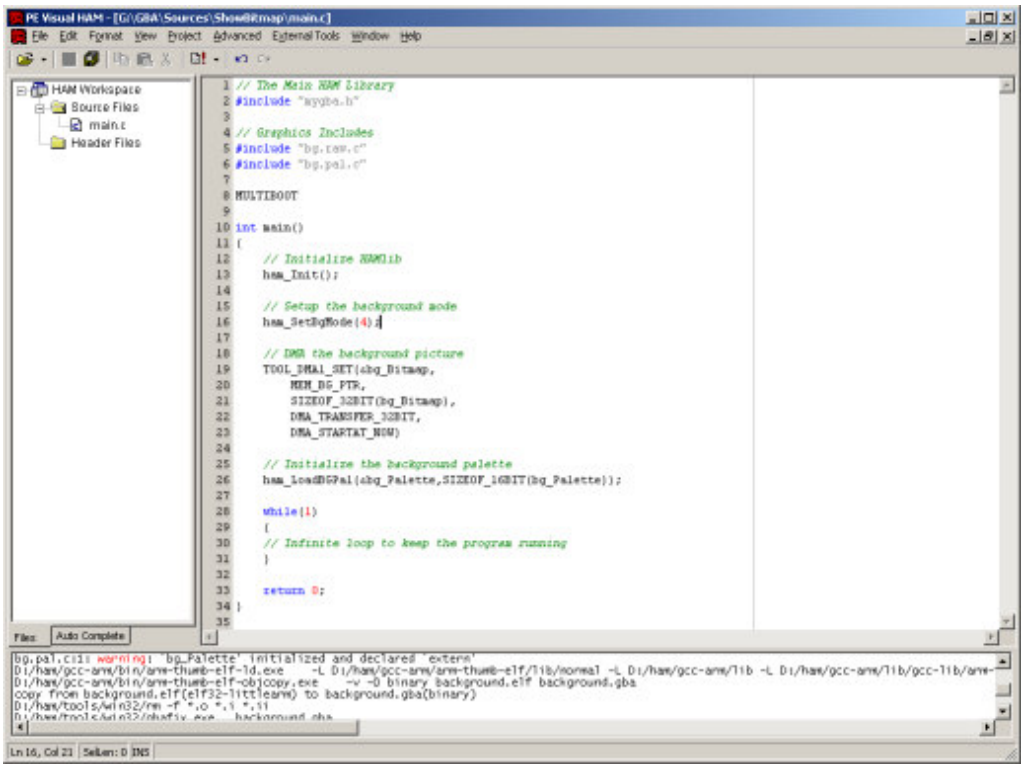

Figure 3.14

The ShowPicture project loaded into Visual HAM.

Now you can compile and run the program. To do this, open the Project menu, as shown in Figure 3.15. Select the Build + VBA option. This will compile the project and then run it in the VisualBoyAdvance emulator (which was included with the HAM SDK).

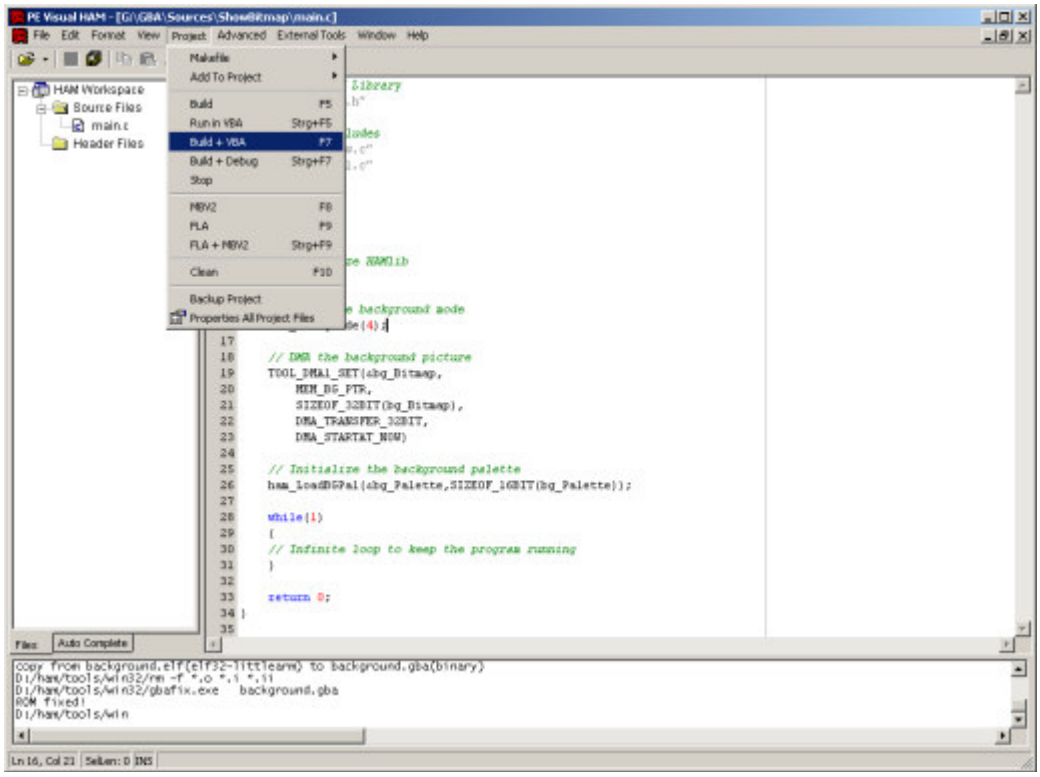

Figure 3.15

The Project menu in Visual HAM is used to compile and run programs.

If all goes as planned, you should see the ShowPicture program start running in VisualBoyAdvance, as shown in Figure 3.16. How do you like that? You have just run your very first Game Boy Advance program, and it only required a few mouse clicks!

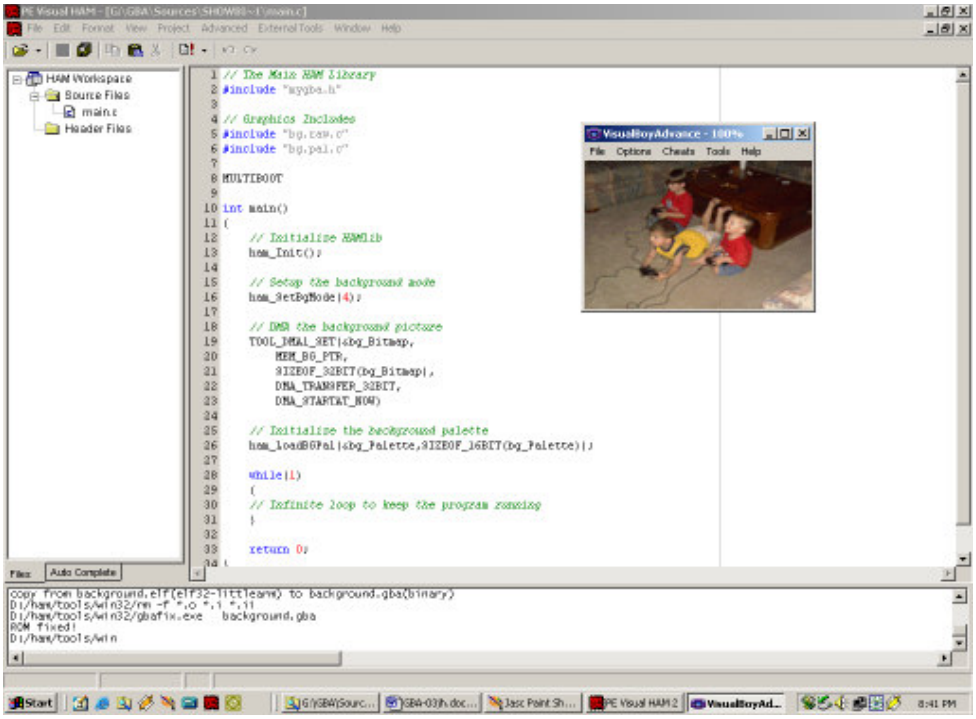

Figure 3.16

The ShowPicture program is running inside the VisualBoyAdvance emulator.

I don't know about you, but I was absolutely *thrilled* the first time I compiled and ran my first Game Boy Advance program! It's an incredible feeling, like taking that first step into previously unscathed terrain, where only a select few have trod. Most important, if you have made it this far, then that is positive proof that you have installed the GBA development tools correctly and may proceed through the rest of the book, running programs, writing code, learning secrets of GBA programming, and basically having fun!

Game Boy Emulation

VisualBoyAdvance is the emulator included with the HAM SDK, and the one I use in this book, although there are other GBA emulators available. The reason VisualBoyAdvance is preferred is because it includes some features for development, such as the ability to print out debug statements at the bottom of the VisualBoyAdvance window using the printf function (a standard C function).

As with all of these development tools, VisualBoyAdvance is constantly being updated with new features, more optimized code, and so on. The version that comes with HAM SDK is

probably not the latest and greatest, so I highly recommend that you visit the VisualBoyAdvance Web site (http://vboy.emuhq.com) frequently to check for updates. At the time of this writing, the latest version of VBA is 1.4, however, the version included with HAM SDK is version 1.0, and it is very limited. Later versions added some great new features, such as the ability to change the window size. I use this feature most often, because it is nice to see a larger version of the GBA screen; you will see sample screen shots using this feature in later chapters.

To upgrade the copy of VisualBoyAdvance that comes with HAM, you'll want to first download VisualBoyAdvance from the Web site shown above, and then extract the VisualBoyAdvance.exe file from the downloaded zip file. Now, the version included with HAM is just vba.exe, so you will want to rename VisualBoyAdvance.exe to vba.exe. After you have done that, you can copy the new vba.exe file over the old one. It is located in C:\ham\tools\win32.

Running Programs on Your GBA

In addition to emulation, there are also options for running your compiled programs on your very own GBA! There's nothing quite as fascinating as watching your own custom-written code running on an actual video game machine, even a small handheld like the GBA. This is where it's really at—where you can truly demonstrate your skills. While you may be just a hobbyist, there are many aspiring game programmers that I would like to speak to now. There is no better way to find a good job as a game programmer (and more specifically, a console programmer) than by showing a potential employer your demo programs running on a real GBA.

Now, one of the things I am recommending is that you produce a demo CD-ROM with VisualBoyAdvance and all of your precompiled GBA code, so you can send it to potential employers. This is not a book about finding a job, although I will give you pointers now and then, but I can tell you definitively that project leads and development studio managers and owners want to see demos before they see your qualifications on paper—assuming you have no experience, that is. Obviously, if you have experience in the game industry already, then you

Again, I am only talking about the hypothetical situation where you are an aspiring game programmer and you are honestly looking to get into the industry. If you are a hobbyist with no such aspirations, then please look over such advice columns and realize that there are a great many programmers in the world with *precisely* that kind of dream.

know what I'm talking about intrinsically. Nothing quite beats some actual demos that will tangentially demonstrate what you are capable of doing.

However, imagine getting an interview request after sending a demo CD-ROM and your resumé to a potential employer. Why not take your own GBA to the interview with a flash card containing all of your demo programs and games so you can show the interviewer your work firsthand? That is possible by using a flash linker, which is a device that plugs into the parallel or USB port on your PC and is capable of writing a ROM directly to a flash cartridge (which houses flash memory, sort of like SmartMedia or CompactFlash cards). However, the flash cartridge looks like a real GBA game cartridge and actually functions as one when plugged into a GBA. What a great way to impress an interviewer by showing him or her your demos running on a real GBA.

To impress a potential interviewer even *more*, consider installing an Afterburner backlight to your GBA (assuming you don't own a GBA SP model with a built-in backlight). The better your demos look on the screen, the better your chances for a second interview and/or a job offer!

What hardware options are available for running your programs on the actual device? There are two primary means of doing so: a multiboot cable and a flash cartridge. The next two sections simply present an overview; you may look to the next chapter for a complete tutorial on compiling and running programs with a multiboot cable or a flash linker.

Using A Multiboot Cable

Multiboot cables, such as the popular Multi Boot Version 2 (MBV2), take advantage of the multiboot capability of the GBA, where two or three players can connect to another GBA running a game that supports multiple players.

What happens is that the other players are connected to the primary GBA using link cables, which are available at any video game store. GBA games such as *Mario Kart Super Circuit* with link cable support actually have small miniature versions of the game stored on the cartridge, and that mini-ROM is sent to the linked GBAs when they power up (note that no cartridge is needed for linked play). The multiboot cable takes advantage of the capability by fooling the GBA into thinking it is linking up to a multiplayer game, when in fact it is simply linking up to your computer. The MBV2 software detects the signal coming from the GBA, requesting the ROM, and sends the ROM to the GBA using the same protocol that the GBA uses when playing a multiplayer link game.

The GBA has only 256 KB of memory available for running multiboot games, which is a serious limitation for any large game project. However, many hobby projects (including the sample programs developed in this book) require less than 256 KB of memory, so they work fine as multiboot games. What happens, though, when you've written a really great game that takes more than 256 KB of memory? For instance, what if your game has a lot of cool graphics and sound effects and requires several megs of memory? A game that big won't work with the multiboot cable. That is a job for a flash linker.

Using a Flash Advance Linker

A flash linker is a device that reads and writes flash cartridges; there are such cartridges available that are compatible with the GBA game cartridge slot. This makes it possible to copy your compiled programs to a blank flash cartridge and then insert the cartridge and run your programs on the GBA—which is as close to the real thing as you can get. While a flash cartridge uses rewriteable memory, retail GBA games have PROM (programmable read-only memory) chips that are basically "write-once/read-many" in nature. Once a PROM has been burned, it is permanent. But flash memory is rewritable, and the technology is similar to EPROM (erasable programmable read-only memory), despite the apparent incongruity of terms. It actually does make sense when you consider that the target device can only read the memory chip, while a burner is capable of erasing and rewriting the contents of the EPROM chip.

A flash linker connects to the parallel or USB port on your PC and is capable of both reading and writing to or from the flash cartridge (which looks just like a regular GBA cartridge). There are some legal considerations when using a flash linker that are simply not relevant with the MBV2, because a flash linker can be used to copy a retail GBA game cartridge just as easily as it is able to read a rewritable flash cartridge. It is therefore possible to copy a retail GBA game ROM to a storage medium on a PC (such as the hard drive), which then makes it possible to make copies of the game. For this reason, the use of a flash linker is somewhat questionable and may not be legal in the country or state where you live.

I will assume you are a professional if you are reading this book and you have no desire to pirate games in this way. If you *are* interested in acquiring games by illegal means, I urge you to consider buying used games instead of copying them. For one thing, used GBA games are very affordable, while blank flash cartridges are very expensive (on the order or $150 and above). The implied consideration is also one that focuses on the concept of biting the hand that feeds you. If you are an aspiring game developer, show respect for the industry and discourage illegal software piracy by setting an example for your peers to emulate, and

demonstrate the benefits of purchasing legitimate games, with the obvious bonus of having a collectible game library.

Summary

This chapter was a critical step in the process of learning how to write GBA programs, providing an overview of the development tools, a tutorial on installing and configuring Visual HAM and the HAM SDK, and even a walk-through of loading and running an actual GBA sample program. As many GBA programmers have used various development tools (including the official Nintendo SDKs), this chapter helps to normalize expectations and establish the basis for the remaining chapters. Once it is understood what development tools are used to write GBA programs, the programmer may delve into later chapters knowing what to expect. For the beginner, this chapter was especially important and useful because arguably the most difficult aspect of getting started writing GBA programs is deciding on a development tool and then installing and configuring the compilers and other utilities. This chapter cleared up the issue and demonstrated the power and flexibility of the HAM distribution.

Chapter Quiz

The following quiz will help to further reinforce the information covered in this chapter. The quiz consists of 10 multiple-choice questions with up to four possible answers. The key to the quiz may be found in Appendix B.

1. What is the name of the software development kit used in this book to compile Game Boy Advance programs?

    A. HAM
    B. Visual HAM
    C. Hamlib
    D. DevKit-Advance

2. What is the name of the integrated development environment used for editing source code and managing development projects?

    A. HAM
    B. Visual HAM
    C. Hamlib
    D. DevKit-Advance

3. What is the name of the comprehensive Game Boy Advance programming library featured in this chapter?

A. HAM
B. Visual HAM
C. Hamlib
D. DevKit-Advance

4. What does MAME stand for?

A. Military Aftermarket Mobile Emitter
B. Multiple Arcade Machine Emulator
C. Multiple Arcade Maker Emulation
D. Maintenance Associate Market Examiner

5. How much does a licensed copy of the HAM SDK and Visual HAM cost?

A. $25
B. $50
C. $100
D. Free

6. Who developed and released the HAM SDK into the public domain?

A. Peter Schraut
B. Emanuel Schleussinger
C. Harvey Minefield
D. Shigeru Miyamoto

7. Who developed and released Visual HAM into the public domain?

A. Rayford Steele
B. Harvey Minefield
C. Peter Schraut
D. Emanuel Schleussinger

8. What is the name of the Game Boy Advance emulator used in this book?

A. DevKit-Advance
B. BoycottAdvance
C. VisualBoyAdvance
D. Big Fat Emulator

9. What is the URL for the HAM SDK Web site?

A. http://www.ngine.de

B. http://www.console-dev.org

C. http://vboy.emuhq.com

D. http://www.nintendo.com

10. What is the name of the device that reads and writes Game Boy Advance flash cartridges?

A. Multi-Boot Version 2

B. Game Boy Flasher

C. Cartridge Flash Linker

D. Flash Advance Linker