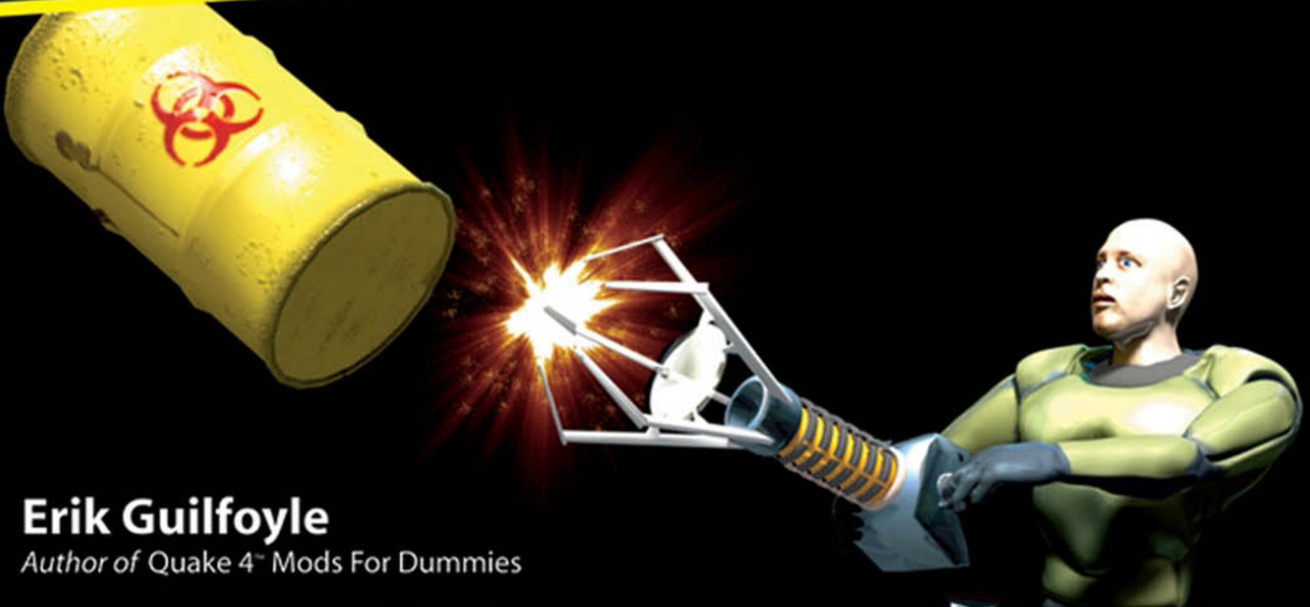
Change the rules of the game
by designing customized levels

# Half-Life® 2 Mods For Dummies®



Erik Guilfoyle

*Author of* Quake 4™ Mods For Dummies

A Reference for the Rest of Us!®

FREE eTips at
dummies.com®

# Half-Life® 2 Mods FOR DUMMIES®

by Erik Guilfoyle

BICENTENNIAL 1807 WILEY 2007 BICENTENNIAL

Wiley Publishing, Inc.

# Half-Life® 2 Mods For Dummies®

# Half-Life® 2 Mods FOR DUMMIES®

by Erik Guilfoyle

Wiley Publishing, Inc.

**Half-Life® 2 Mods For Dummies®**

Published by
**Wiley Publishing, Inc.**
111 River Street
Hoboken, NJ 07030-5774
www.wiley.com

Library of Congress Control Number: 2006936817

ISBN: 978-0-470-09631-4

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1O/SX/RS/QW/IN

WILEY

# About the Author

**Erik "foyleman" Guilfoyle** joined the game-modding scene shortly after the release of *Half-Life* in early 2000 by creating a custom level and supporting material for the game. After that, Erik was hooked on modding games.

Two years and a lot of practice later, *Soldier of Fortune* was released. Erik jumped at the opportunity to map a custom level for this game and was among the first to release a map for the game with custom textures. This led to a flurry of e-mails requesting advice for constructing custom material as well as the beginning of a compendium of game modification tutorials.

Not much later, Erik started his own Web site to host existing and new tutorials. The site grew until he had the largest collection of tutorials on the Internet for *Call of Duty* mapping. Now, Erik runs the MODSonline.com modding community with the assistance of two other administrators and several friends, covering many games, including *Half-Life 2.* As new games are released, he starts the tutorials section with beginner instruction and leads the member forums in preparation for the next up-and-coming game.

In August of 2006, Erik published his first book through Wiley Publishing, Inc.: *Quake 4 Mods For Dummies*.

Aside from his love for games, Erik is also the vice president of an established media company, Tres Inc. His company has produced 3-D models, animations, motion graphics, and Web sites for companies and corporations throughout New Jersey for over five years. Mission Critical Studios, an offshoot of Tres Inc., is now working on a game of its own.

## Dedication

This book is dedicated to the MODSonline.com modding community.

## Author's Acknowledgments

I would like to express my thanks to those that have knowingly and unknowingly helped me to gain the knowledge that I now have in the field of game modding. David Gonzalez helped to get me started in the world of modding by introducing me to the original *Half-Life* game and giving me time to continue even when we had real work to get done. My wife Kate allows me to spend countless hours on the computer rather than with her without too much complaining, and I must thank her for that. Also, thanks to Peter and Cathy Guilfoyle, my parents.

And what would this book be without the *Half-Life 2* game itself? Valve Software built an awesome game and allows people like me to wreak havoc on the code that makes it all work. For this game, the games before it, and the games to come, thank you for making the world that much more fun.

This book would not have been possible if it were not for the kind and talented folks at Wiley. Melody Layne gave me the opportunity to work with Wiley and got me on track with an easy-to-follow format that anyone can read. Christopher Morris and Jeff Salé helped to keep me on that track with insightful and helpful suggestions through editing. I know several others had a part in helping me put this book together, and I extend my thanks to them as well.

Finally, thanks to all the fine members of MODsonline.com. It is with them that I have learned so much and continue to learn more with each game that comes out. I only hope that all the tutorials, forums posts, and everything else I do help to one day make games even more outstanding than they are now.

## Publisher's Acknowledgments

We're proud of this book; please send us your comments through our online registration form located at `www.dummies.com/register/`.

Some of the people who helped bring this book to market include the following:

***Acquisitions, Editorial, and Media Development***

**Senior Project Editor:** Christopher Morris

**Acquisitions Editor:** Melody Layne

**Copy Editor:** Jennifer Riggs

**Technical Editor:** Jeff Salé

**Editorial Manager:** Kevin Kirschner

**Media Development Specialists:** Angela Denny, Kate Jenkins, Steven Kudirka, Kit Malone

**Media Development Coordinator:** Laura Atkinson

**Media Project Supervisor:** Laura Moss

**Media Development Manager:** Laura VanWinkle

**Editorial Assistant:** Amanda Foxworth

**Sr. Editorial Assistant:** Cherie Case

**Cartoons:** Rich Tennant (`www.the5thwave.com`)

***Composition Services***

**Project Coordinator:** Patrick Redmond

**Layout and Graphics:** Jonelle Burns, Carl Byers, Lavonne Cook, Joyce Haughey, Stephanie Jumper, Barry Offringa, Heather Ryan

**Proofreader:** Techbooks

**Indexer:** Techbooks

**Anniversary Logo Design:** Richard Pacifico

***Special Help:*** *Colleen Totz Diamond, Teresa Artman, Andy Hollandbeck*

**Cover Image:** David Gonzalez

---

**Publishing and Editorial for Technology Dummies**

**Richard Swadley,** Vice President and Executive Group Publisher

**Andy Cummings,** Vice President and Publisher

**Mary Bednarek,** Executive Acquisitions Director

**Mary C. Corder,** Editorial Director

**Publishing for Consumer Dummies**

**Diane Graves Steele,** Vice President and Publisher

**Joyce Pepple,** Acquisitions Director

**Composition Services**

**Gerry Fahey,** Vice President of Production Services

**Debbie Stailey,** Director of Composition Services

# Contents at a Glance

# Table of Contents

# Introduction

If this is your first time venturing into the world of game modding, let me post a warning now: You are entering into something that many people (including me) have described as addicting, exciting, and frustrating all at the same time. I know very few people who started on the path of game modification and found it easy to stop working on their projects for even a few days. If you're experienced in game modification, you already know what you're getting yourself into. Either way, I welcome you to *Half-Life 2 Mods For Dummies.*

I've lead many beginners along the path of game modification, and they all say the same thing: This is an addictive hobby, and it isn't easy to put down even for the night. On several occasions, I've sat down in my computer chair to work on my game, planning to accomplish only a few things. The next thing I know, it's 2 o'clock in the morning, I'm still working, and I have to get up for work in another four hours.

So, heed my warning but also enjoy what is to come. This book gives you the knowledge and skill to mod on your own. I show you not only how to create modifications for the *Half-Life 2* game but also create content that will have you bragging to your friends and family. Although this book can't tell you every aspect of modding *Half-Life 2,* it shows you how to continue on your own and grow into a master of games.

## About This Book

When the first version of *Half-Life* came out, a buddy of mine and I decided to create our own mod for the multiplayer version of the game. From that moment on, I was hooked on modding. There really was no turning back for me, and I just kept on creating more and more content.

Eventually, I moved on to another game — *Soldier of Fortune* — to see what kind of things I could create elsewhere. Quite often, I was unable to find assistance online for creating my custom content, so I decided to learn it on my own. Through much trial and error and many discussions with other modders online, I discovered that I was able to help others with their projects. That's when I decided to dedicate my efforts to teaching and guiding others on the how to mod the *Soldier of Fortune* game.

Over the years, I furthered my exploration and tutelage of modding with other games, and here I am now with *Half-Life 2.* I really enjoy showing others how to make mods for games, and I can do the same for you. All you need to do is read on.

My goal in this book is to show you by example many aspect of modding *Half-Life 2.* I provide you with an understanding of how things work so that you can take what I show you and expand your newly found knowledge on your own. I leave you with not just information but also something you can play in the game and show off to your friends and family.

Some of you might feel that the focus of this book is on mapping rather than modding for *Half-Life 2.* True, I do focus on the most popular form of game modification — mapping —, but mapping is a type of game modification. I also provide some insight that leads you toward more complex modding of the game that you can further explore on your own. The space I have here is limited, but the info in this book will start you on what I find to be the quite enjoyable path of game modification.

## Conventions Used in This Book

I lead you into the world of modding *Half-Life 2* by way of two methods, both of which I feel are equally important. The first and most obvious is by way of example. As I walk you through the different aspects of modding, I show you, step-by-step, the methods to use. I don't leave you guessing what needs to be typed, clicked, or done.

For each chapter that has you creating something for the game, I also supply an example. On the companion CD (as the back of the book), you can find chapter-based files containing all the work that was reviewed. You can use this either for reference of the current chapter or as a starting point for the following chapter, which builds upon the previous.

The second method that I use is that of background. I provide you with all the necessary background for each subject in modding. A statement made to me once a long time ago that I have never forgotten is, "It's easier to drive a car if you know how it works." You can apply the same principle to games. The more you know about how a game works, the easier it is to mod.

Background provides you with understanding. With understanding, you can do more than create. You can also better find out why something doesn't work. There are going to be moments that something doesn't turn out as you expected. With a background of how you got to your issue, you should be able to solve any problems that arise.

## Foolish Assumptions

To find a starting ground upon which to write, I had to make a few assumptions.

First, I assume that you have a basic understanding of your computer. Perhaps not so obvious, I also assume that you're working on a computer that uses the Microsoft Windows operating system. At the time of this writing, you can modify the *Half-Life 2* game only on a Windows installation. If you use a Linux or Mac operating system, I cannot be certain that the chapters contain proper examples.

Because you have a Microsoft Windows operating system, I took the liberty of making a few other assumptions. I figure that you know how to open and use Windows File Explorer for browsing folders and files on your computer. I also figure that you have the basic programs that come with Windows, such as Notepad, installed and accessible.

If you don't have a Windows operating system, the contents of this book can still be of great value. Through example and history, the information I provide you can be useful regardless of the game or platform on which you're working. The discussions here help you to understand the modding process for most games out there. You will be ready to mod, even if you can't actively follow my examples right now.

Second, you must know how to use the keyboard and mouse. You also need a three-button mouse. Although you can follow the majority of this book without a three-button mouse, there are one or two operations that you can't do without this type of mouse.

Finally, because you purchased this book, I assume you know something about games. You should already have a copy of the *Half-Life 2* game, not just in your hands, but installed on your computer. Hopefully, you installed the game by using the default installation paths. After you installed the game, I assume you got it running and perhaps even played it a little bit. You bought the game, so you should spend a little time enjoying it.

## How This Book Is Organized

Although this book is written in a particular order and each chapter builds upon the next, this doesn't mean you can't skip around. If you want to know how to make your own textures, go for it. Head over to Chapter 15 and start on your custom textures. There I show you how to create your own texture from start to finish.

If you'd prefer to better optimize your map, Chapter 10 is the one for you. There you can see how to build you map and modify it for smoother play in the game. (It will have better speed and rely less on your processor.)

As for the chapters that build upon the previous chapters, those are fine to skip around as well. Just load the example files from the previous chapter as found on the CD at the back of the book. Then go on your way as if you've accomplished everything I've written up to that point. If I feel you should reference a previous chapter for clarity, I make sure to mention it.

This book is broken into five parts as follows.

## Part I: The ABCs of Modding

Allow me to introduce you the world of modding. In this part, I help you better understand what it means to modify a game. I point out specific aspects of the game and relate them to something you're familiar with: real life. Then, I briefly walk you through the steps involved in modding a game.

After you're familiar with the concept of modding, I introduce you to some of the tools you'll be using and how you can use them. I acquaint you with some important things that you should remember while modding, and I explain why you want to remember them.

Then, I start breaking down the game. You've perhaps already played through *Half-Life 2,* but I'll bet you haven't seen it the way I'm going to show it to you. I help you look at the game through different eyes.

## Part II: Making Your Own Maps

In Part II, I ease you into the modding process of making your own levels. I start you off with the tools required and show you around. When you're familiar with the popular map editor, I get you started on making your first level. It won't be much, but by the end of this part, you will have a level that you can play in the game.

While making your own level, you get all the basics required. From creating a room and painting the walls to adding light, you build a place to start. From there, you convert your level from the editor load it in the game for some fun.

## Part III: Expanding on Your Creation

Part III goes beyond the basics of building a simple level for the game. This is where you go from a single room to something you can enjoy with your friends online. This is arguably the best part of making a level for *Half-Life 2*.

I start you off with simple additions to the level. You expand from creating a single room to creating something a bit bigger. From there, I show you some optimization tricks. These advancements are some of the most valuable because they can perhaps help the novice and advanced alike.

With your small level in hand, you next turn it into a fun multiplayer level. You add on an outside area with a great sky. Then, I show you how to place items within the level that the player can use to win the battle.

## Part IV: Going Beyond the Basics

Map construction isn't where the fun stops. In Part IV, I take you on a path that can help to improve the level you created in previous chapters and then set it free to the public to enjoy it as you do.

The trip starts with the world of textures. Textures are what can take your player from the dunes of Mars to the beaches of Hawaii. From the confines of a dungeon to the vastness of space, you can create the images that line the walls of everything in the game. Not only are the walls covered in this part but so are the faces and bodies of the players within the game. Textures cover it all.

Closing this part is what you need to know to get your mod out to the world. I show you what's involved in packing up your level for distribution. Then I guide you through what should be done to get your mod onto the Web sites for all to see. Why not let the world have as much fun as you did with your awesome creation?

## Part V: The Part of Tens

Part V puts everything else into perspective. Modding doesn't stop where I left off. There are tons of other things that you can do to customize a game, and I provide you with ideas to get you started. At this point, you have the knowledge to further yourself in the world of modding or to find more help when needed.

I provide you with ideas and examples of modding tips and tricks and where to look for them. Then I provide examples of the different mods currently available online and why I think they're a cut above the rest.

## Icons Used in This Book

As you read through the book, you'll find some fun little icons in the margins. These icons alert you to special content that often highlights the topic at hand. Here are the icons you might see and what they mean:

TIP

Tips provide helpful information about the subject. Although perhaps not immediately relevant to performing the task, these tips will come in handy in the future.

REMEMBER

Remember icons remind you of an important idea or fact that you should keep in mind as you explore *Half-Life 2* mods. They might even point you to another chapter for more in-depth information about a topic.

WARNING!

Warnings point out specific actions that you want to keep your eye on. Read each of the warnings as you're going through a chapter to make sure that problems don't arise.

TECHNICAL STUFF

Technical Stuff icons provide further insight into the subject being read. You can skip them if you like, but I find them to be quite informative.

# Part I
# The ABCs of Modding

The 5th Wave By Rich Tennant

"Dennis! Dennis!! Where are you?! I'm being attacked! I know, I know, but you're the groom. Tell them to just postpone the wedding a few minutes."

## In this part . . .

Games can be a lot of fun to play. They allow your mind to explore a virtual world as someone or something that you've never been before. From your own home to the jungles and deserts of the world to the infinite possibilities of space, you can go anywhere you want. Your only limitations are the imaginations of those who created the game.

Well, playing games is just where the adventure begins. *You* can be the one who tells the story of a far away place. *You* can create lands and worlds that come from your own imagination. Then, *you* can be the one who takes your friends on an adventure that you created.

In this part, I take you behind the curtain that is your computer screen. I show you that there is no limitation to what can be accomplished with games. The fun doesn't stop when you reach the end credits — this is when the fun just begins.

Chapter 1

# Modifying the Game

***In This Chapter***

- Looking at the game through a modder's eyes
- Finding modding tools that you had all along
- Walking through the making of a mod
- Going public with your creations

Have you ever been playing a video game and thought, "I would have done it differently" or "I could have done it better"? Perhaps you thought, "Wouldn't it be cool if. . . . " Well, you don't have to just think it. You can make changes to games, and you don't have to be a software engineer to do it.

*Game modification* — changing something in a game — has generally been associated with the first-person shooter (FPS) and real-time strategy genres. The change could be very small, such as making a player's outfit orange instead of blue, or the change could be very large, such as creating a whole new environment for the player to explore. You can change almost every aspect of a game and make it look and feel like something completely different. Or, instead of altering an existing part of the game, you could add new elements to it. Anything that in some way modifies a game from what it was when the publisher released it is a *mod.*

Game modification isn't a new practice. However, only recently, with the creation of multiplayer shooters for the PC, has it become popular. It was this genre of gaming that gave people the inspiration to show off. At first, players competed to see who was the best FPS player. Later, when players realized that they could modify a game, the competition grew to include this aspect of the game and to see who could make the most impressive changes to a game.

The FPS game genre was created in early 1990. You play from the point of the view of the in-game character just like in *Half-Life 2.* This gives you your first-person perspective of the game.

The center of action revolves around you, *Gordon Freeman,* as the player, while you use a handheld weapon, such as a pistol. Although your primary objective in the game might change, you're often placed in situations where you must shoot your weapon — thus, making the game a *shooter.*

Controversy surrounds which game was the first FPS game. It's a tossup between *Spasim* and *Maze War,* which were first developed in 1973. Then, later that same year, player versus player game play was tested between two linked computers playing *Spasim.* The following year, both games were introduced to a network, and multiplayer gaming as we know it was invented. Because both games played from the first-person perspective with weapons, this marked the birth of the FPS.

In 1991, id Software released the game *Hovertank 3D,* which was a simple maze game from the first-person perspective. The environment was very flat, and the enemies were nothing more than 2D graphics. Later that year, *Catacomb 3D* was released (as a modified version of *Hovertank 3D*) featuring textured walls as well as showing the player's hand onscreen — like you now see in *Half-Life 2.*

In 1992,VGA (Video Graphics Array) graphics was added to the release of *Wolfenstein 3D.* This game was a huge hit due to both the game play and this higher-quality graphics and inspired more development in the genre. The following year, *Doom* added even more graphical detail. This game offers rooms of various sizes, outdoor environments, and textures that were previously flat surfaces. However, the most important upgrade to this rising game genre was the ability for anyone connected to a network to enjoy the multiplayer aspect.

The first version of *Quake* was introduced by id Software in 1996. It had highly upgraded graphics as well as networking capabilities and was the first game in the genre to gain widespread fame as a multiplayer Internet game. It broke the bounds of its predecessor, *Doom,* by networking globally. To further its success, *Quake* was the first game that offered developer support for user modifications. This was the beginning of mods created by the consumer rather than the industry, and it was also when Valve Software began work on its game, *Half-Life.*

In 1998, Sierra Studios and Valve Software released *Half-Life.* This FPS was based on a heavily modified version of the *Quake* game engine. At first, what made this game a huge hit was the presentation of the game and that it had a storyline with a plot. Not many games before it had actual plots to involve the player; rather, they simply offered the player something to shoot. For the first time, there is a game presenting the player with an interesting story.

Developed on an adaptable game engine, *Half-Life* continued to encourage the gaming and modding community to further develop *Half-Life.* Valve Software provided excellent support from the beginning by including level-design tools with the software. Later, Valve Software released documentation, additional source codes, and tools to further the capabilities of modders.

Then, in 1999, a beta version of *Counter-Strike,* a *Half-Life* mod, was released to the public. This mod grew in popularity like no other mod before it. It was so well received that Valve bought the rights, assisted in its continued development, and released it as a commercial expansion of the game in 2000.

Many other mods were created for *Half-Life*, but *Counter-Strike* set the standard. It was proof that modding could get you into the gaming industry and benefit the game developers.

Valve Software took five of the six years since the release of *Half-Life* to deliver a game of which they were proud. That game is *Half-Life 2* and it has been redeveloped from the ground up to provide us with the best possible game play to date. Now we, as Gordon Freeman, can continue the story where it left off.

## Checking Out Half-Life 2

Whether you purchase *Half-Life 2* with or without the intent to modify it, you should begin by playing around with it. Play through a few single-player missions and then move on to the multiplayer games. If you don't want to play online, start multiplayer games of your own. You might be the only one in the game, but you will still enjoy yourself.

After playing the game and enjoying what the developers were able to deliver, play the game again — but this time, instead of running around and shooting everything that moves, take some time to look around. Stop and look out windows and over railings. Walk around the other players in the game and see what they're wearing. Take a closer look at the walls to see the details that are included, and then see what happens when you shoot them with different weapons. Listen to the sounds the weapons make as well as the sounds all around you.

By investigating the details of the game, you start to see things differently. It's like looking at a room where you live and thinking about painting the walls a different color or moving the furniture. It could also be like considering a different outfit for the day as opposed to the same outfits that you wear every day.

## Adding to or Changing the Game

At first, seeing which game elements you can change can be difficult. However, when you begin to understand the different pieces that make up the game, you will start looking at all games a little differently. You can relate the various elements you see to specific files within the game, and you'll start to know which of those files that you can modify. For instance, look around the physical area in which you're now sitting. Within the area, you see objects, like the book you're holding; a table with some items on it — or, if you're outdoors, maybe some trees. In the game world, each different item could be considered a separate object that the game refers to as an *asset.* Each asset, because it may be used more than once in a game, is defined in files. If you change one of these files — say, to change the book you're holding to a different color — you made a modification.

So what does this have to do with mods and modding? Well, if you modify the game so that it's in any way different from when you purchased it, you create a mod. *Mod* is just a short way of saying *modification.* Then, it stands to reason that the act of modifying is called *modding.*

The mods that you make can be simple or complex. You can make them by adding something new to the game or by changing something that already exists. You could make your changes to provide an improvement to the game, or you could completely change everything and create a *total modification* of the game. You might be surprised to know that many of the games on the shelves are total modifications of another game. The original *Half-Life* game is essentially a total modification of *Quake* from id Software, and *Half-Life 2* was built from the basis of *Half-Life.*

### Finding out what you can mod

Games are just groups of files that are read by one master program that displays those files' contents on the screen. When one or more of these files is changed, the change is reflected within the game. Official game updates and expansion kits can perform these changes, but you can, too. So why not include your own changes to the game to create something completely new?

Upon first glance, you might not realize just how much game content you can mod. Everything, all of what you see from the time you double-click the game icon to the moment you close down the game, can be changed. A short list of moddable things in a game would read like this:

- **Textures and images:** Everything that you see when you play any level in *Half-Life 2* started as an image. Whether it's the bricks on the walls or the face on another player, these are all images that can be added. I show you in Chapter 13 how to create your own textures.
- **Levels:** From multiplayer to single player, you can build completely original levels for the game that you and your friends can play. What could be more fun than playing a multiplayer level together with your friends online?
- **User interfaces:** The selection windows before playing the game and the usable computer screens within the game can be modified. You can set up these screens to better meet your needs or to make things look any way you have dreamed.
- **Coding changes:** Change the way a weapon shoots or how much damage a player receives when they fall from the top of a building. This and much more can be achieved through code changes and *Valve Software* has given us permission to make them.

The preceding is just a very short list of what can be modded in this game. As long as you have access to the files that make a game run and you have the tools to change them, you can modify that game as much as you like. You could even turn *Half-Life 2* into a new version of *Donkey Kong* if that's what you want.

The reason *Half-Life 2* can be modded so extensively is primarily due to the developers. Luckily, players have been provided with access to the game files so that we can modify them. Not all game companies permit that kind of access.

## Knowing what tools you need

Tools are available for every job, and game modification is no exception. Some tools are provided for you by the game developers, but others you must obtain. However, you might be surprised to know that most of what you need, if not all, is already installed on your computer. You just need to know which programs you can use to modify each of the different files within the game.

Here are examples of such programs that you might already have:

- **File-compression utility:** If you've been downloading a lot of files, you're certain to have a file-compression utility. The program of choice in this book is WinZip, which is used primarily to open `.zip` files found on most download sites. This utility can easily open the compressed game files that were installed with the game so that you can gain access to all the moddable game files and start having fun.

- **Plain text editor:** Many of the files in the game are written and modified by using a plain text editor. I'm certain that you already have at least one of these installed on your computer. Notepad, for instance, is a perfect program for editing these files. It comes installed with Windows and can read, edit, and save these files without any special setup or instruction.
- **Developer-provided tools:** When you install the *Half-Life 2* game, you also have the option to install some of the modification tools via the Steam engine, *Source SDK.* This software, a developer's kit for the Source engine, allows you to modify the game and create custom game levels to play.

As modding became popular, game developers started to assist the modification community. They offered words of advice and eventually tools and documentation to make more complicated changes. As the modifications became bigger and better, so did the sales of the original game because more and more people wanted to play the game with these new modifications installed. This inspired more participation from developers and publishers who offered even better tools and documentation.

## The Modding Process Goes Something Like This

The most common type of modification is to create a custom level for the game. The process of doing such goes like this:

1. **Plan your custom level with notes and drawings.**

   Write down what you want to include in your custom level and maybe even sketch out how you want things to lay in the game.
2. **Construct the level in a program by building walls.**

   This is a lot like playing with blocks. You create and place your different shaped blocks where you want them in order to create a room, several rooms, or any other structure for the player to roam.
3. **Add some color to the surfaces in the level.**

   Adding color is a simple process of selecting an image and then applying color to the wall, floor, or any other surface in the game.
4. **Place additional elements in the game such as lights, monsters, weapons, or other objects.**

Again, just select elements from a list and place them where you want. Then you can fine-tune the way they work. (For instance, you can change the color of a light.)

5. **Compile and play the level in the game.**

   Choose a compile command from the editor's menu, and it creates all the files required so that you can play your finished level in the game. Then you just load the level and start having fun.

6. **(Optional) Give your level to the world.**

   This optional step puts all the custom files together into a single file that you can place online for download or on a disc to hand to your friends. This way more people can enjoy the work you put into your custom modification.

As you can see, the process isn't all that complicated. In this book, I show you where you can find the necessary tools, how to use them, and the options that each tool has to offer. With this information, you soon will be on your way to making your own custom game levels.

## Sharing the Game with Others

In the list in the preceding section, I mention that the last step of the modification process is optional; however, sharing your creation is most often the purpose of making a mod. I think that it's perhaps the most exciting part. For my part, knowing that many other people are getting enjoyment from something that I built motivates me to do more.

In this book, I not only show you how to package all your files together for distribution, but I also show you where to go from there. I offer advice on where to send your files and how to get them out to the public for all to enjoy.

# Chapter 2

# Getting Familiar with Modding Tools and Techniques

*In This Chapter*

- Exploring your modding tools
- Writing a README
- Backing up and saving your work
- Comparing *Half-Life 2* with other games

Every job requires the right tool. If you were working on the engine of your car, you would use a wrench or a screwdriver. Modding games is no different, but the necessary tools come in the form of programs.

## Gathering the Tools Involved

Some of the tools you use to modify the game have been placed at your disposal by way of the Steam downloader. All you need to do is to tell Steam to download these tools, and suddenly you're up and running. (The details of doing this are explained in Chapter 4.) These downloaded tools include

- A mapping editor
- A model viewer
- A graphical user interface editor

However, you need some tools that don't come with the game. Some tools you already have and just don't know it yet. The remaining tools have been provided on the CD that accompanies this book. Those tools, discussed in detail in the next few sections of this chapter, are

- A plain text editor
- An image editor
- An image compressor
- A compression utility

Each of these additionally supplied tools serves a very specific purpose. You're probably already familiar with most of these tools in some way. However, even though you might have used them previously, you might not have realized how they can be related to modding *Half-Life 2*.

## Writing plain text

The most common tool used in all forms of game modification is the plain text editor. *Plain text* is simply text that is stripped of all formatting. When you use a word-processing program such Microsoft Word, the software introduces unseen formatting to the text. This formatting could be as simple as bold or italics attributes to the more complex font selection. Plain text doesn't have any of this hidden information. The hidden data would only confuse the game when it attempts to read what you've written.

Because you're working with the Microsoft Windows operating system, you already have an excellent plain text editor installed. Notepad comes with your Windows installation and is the tool that I use throughout the chapters of this book. With Notepad, what you see is what you get, and there is no hidden formatting.

## Drawing, painting, and taking pictures

*Half-Life 2* consists of many images — large and small. Everything you see in the game has some sort of image applied to it. From the walls in a room to the face on a character, everything starts as a picture and is then applied to the corresponding piece in the game.

All the images within the game are either provided in the Targa (`.tga`) image format or started as a Targa image. Those that begin as Targas are eventually converted to a different format called the *Valve Texture Format* (VTF). This format contains more details than Targas about how the image should behave within the game environment.

So, when choosing your image-editing software, choose one that can work with Targa images. The `.tga` image format is the image format primarily used with this game.

The most widely used program for working with images is Adobe Photoshop, and a trial version is supplied on the CD in the back of the book. I use Photoshop in the examples in this book. Although you could use other software applications for editing images, I recommend that you install and use Adobe Photoshop while following this book. This helps avoid miscommunication, and you might find that you really like this program.

Another image application that I use in this book is a VTF image converter called the *Valve Texture Tool* (VTEX). The VTF image is a special image *Half-Life 2* uses that adds additional elements to help the image display easily. Such elements might be reflection, transparency, or bumping as explained in further detail in Chapter 13. With VTEX, you can take your Targa image and convert it into the VTF image for use within the game.

VTEX is supplied on the CD located in the back of this book. It is freeware, so it won't cost you anything to use and it's provided by Valve.

## Packing up your work

After you complete your mod and are ready to show it off, you need to package all of its various components into a simple file that can be easily distributed. Using this single file not only makes it easier to distribute your mod to your adoring public, but it can also save space. By packaging the files appropriately, you can compress them to a filesize that's smaller than the total sum of the separated files.

One of the things you want to consider when creating your package of files is the ease at which the end user can access its contents. For this purpose, I recommend using WinZip. It is easily available online and comes with a free trial so that anyone can download and use this program. Also, it's compatible with a number of other compression utilities such as WinRar and the zip compression agent found within Windows XP.

You can use a number of possible compression utilities to package your files. Some people like to use WinRar, and some use the compression utility that comes with Windows, the Microsoft Zip utility. However, I find that WinZip is the simplest utility to use, and also the one that results in the fewest complications.

TIP

If you don't have WinZip installed, you should install the trial version now. It's provided on the CD in the back of this book. You can also download it from the Web site, `www.winzip.com`.

## Using Best Modding Practices

Modding is still a fairly new activity. You don't have many strict guidelines that you have to follow. However, some very advisable practices are in place. These practices are primarily to avoid conflict with other mods when sharing your final work with others as well as to prevent loss of work because your files weren't properly saved to disk.

### Following standard naming conventions

Whether you're creating a single level or a complete modification of the game, naming your work is very important. You want to come up with an original name for your modification — whether it's a single file or a group of files — that most likely hasn't been used by someone else. Otherwise, when two mods with the same name are downloaded and installed, files will be overwritten and lost. Try doing a little research on similar mods that are already available on the Internet before coming up with the final name of your work to avoid such issues.

For example, say that I release a custom game level under the name MyLevel, and I name the file `MyLevel.map`. If you decide to name your level the same thing, there could be a conflict. If the same person downloads and installs both custom levels of the same name, one of them is going to overwrite the other. This might seem trivial at first, but it happens quite often and is somewhat avoidable.

### Instructing the end user

Always include a README file with your final work. The *README file* is a text file that contains information about you and your mod. With it, the user knows who to contact for help or kudos, what your mod consists of, and how to properly install and use it with the game.

Don't ever assume that the end users know what to do with your files. Rather, assume they know nothing other than how to access your README file, and then instruct them with what to do from there. If your reader doesn't know how to install the file and isn't tempted enough to open the README text file and actually read it, at least you've done what you could to inform the end user to the best of your ability. I explain this file in more detail in Chapter 19.

## Including all the files required

When making your mod, check to see whether any of the files you used are part of a mod that you previously installed. If you installed a level that comes with custom images, for example, make sure to include those images with your mod. You don't want to distribute a mod that doesn't have all the required files.

If you do include files that were previously provided via another mod, make sure you give credit to the author of those images. Make mention of where those unoriginal files came from within your README. Then, make sure those files are also included with your distribution in case the user doesn't have the same mods installed.

## Avoiding the overwriting headache

In Chapter 17, I introduce you to creating your own mod folders within the game. Primarily, you do this so that you don't overwrite the original game files, thereby destroying the original game. You don't want to make any permanent changes to the game. Instead, create a mod folder of your own and place your altered files in there. This prevents you from having to tell users that they have to reinstall their game to get it back to the way it was before they installed your mod.

## Saving and saving again

Game modding is still a young hobby. As such, the available tools aren't quite perfect. Errors can occur when using modding tools that can render your files useless. Although I mention ways to avoid the most common errors, some errors occur seemingly without reason.

To avoid letting these errors ruin the effort you put into your mod, make sure that you save your work regularly. Save it often. Save it under different names for different versions. Then, when you've saved it a number of times after a week or more of work, back it up to another location. Put it on CD, DVD, or other removable media.

One day, you'll think back to these few paragraphs about saving your work, and you'll be thankful. You'll be thankful that you have a backup of your work that you can go to after a disaster, however minor your changes between backups may be.

## Differences between Half-Life and Other Games

Games from different developers are created differently. Although modding practices are relatively the same among the various games, some fundamental differences can have a big impact on the way you plan your mod.

When it comes to level design, the *Half-Life* game series is roughly the same between versions 1 and 2. You create a box and place all your buildings, rooms, halls, models, and everything else inside it. This big box must be made without any gaps because the box defines the boundaries of your level. Without boundaries, the game would crash because it would contain too much information for your computer to process.

You're presented with an empty area like the void of outer space. You create your level by adding to this space blocks such as the walls of your buildings. I like to refer to this process of level creation as *adding to the void.*

Some other games (such as the *Unreal* series by Epic Games), however, work this process in reverse. They present you with a giant block that is like a block of clay. You then carve out your level from this solid block.

With *Half-Life 2,* you add to your environment. I find this a much simpler concept to understand. Just like building a house in the real world, you add a wall, add a room, and add details by building up from nothing.

# Chapter 3

# Breaking Down the Game

### In This Chapter

- ▶ Making your own maps
- ▶ Placing textures in the game
- ▶ Adding interactive elements as entities
- ▶ Scripting your way to reactive elements
- ▶ Using graphical user interfaces to make a more interactive world

*Half-Life 2* consists of several elements that all come together to create what you see on the computer. There are sights, sounds, and interactive elements. All these pieces that make up the game can be found within one common location in the game: the game level. However, before it becomes a level, it starts as a map.

## Making Maps and Playing Levels

The *level* is the virtual world in which you exist as the player. After you start the game and make a selection from the available menu, be it a single-player or multiplayer game, your player is released into the level. Here you can explore the game environment very much the same way you would explore a local park, school, or other environment in the real world.

Whether you're playing a single-player mission or a multiplayer battle, each environment between loading screens is a level.

Loading screens appear in both the single-player and multiplayer game types. The multiplayer games make the screen obvious with the display of a loading bar. However, the single-player loading screens are often hidden with short movies called *cut-scenes.* These span the time between the end of one level and the start of another in the hopes of keeping you further submersed in the game play.

### Leveling levels

The term *level* most likely came from the predecessor of the shooter gaming genre, the role-playing game. The goal of role-playing games is to increase the level of a player by progressing through the increasingly difficult environments of the game. Therefore, the two terms became synonymous. You would level up your player by progressing through varying levels of difficulty in each environment.

Before an environment becomes a playable level, it starts life as a *map*. The map is named such for a few important reasons:

- When you create a playable environment, you must map out the position of everything within your environment. Thankfully, the mapping editor takes the tediousness out of this task.
- Although the extension of your saved file is now `.vmf`, in the previous version of *Half-Life* you would then save the map with the `.map` extension.

After the map file has been converted into a playable level, the map file is no longer needed. I'm not saying that you should delete it. You might want to hold onto it so that you can make changes to it later. However, you don't need to distribute this file to the public unless you want them to have the source for creating your level.

## Making Your Own Maps

I find that one of the biggest thrills of modding is the ability to make my own maps and then play them in the game. I really enjoy being able to create a virtual environment, make it unique, and then offer it to others to enjoy.

Map making requires more than just the placement of a few buildings. It's more like creating a real building or small community and then furnishing it. You must place all the walls, ceilings, floors, and other structural elements to make up the buildings in your game. However, you must also paint the walls, add tables and chairs, and drop in enemies to fight. You must consider every object or feature that's to be represented while playing the game.

I could just jump right into the map-making process with you, but I think that you should first have a good understanding of the game environment. To make a good map, you should understand how your placed objects and features are perceived within the game. Don't worry: This doesn't mean you need to go back to school and learn new theories and rules. Because you live in a three-dimensional (3D) world, relating the real world with the virtual one is fairly easy.

## Building blocks of a map

Adding walls, ceilings, and other structures to your map is much like playing with blocks or Legos. You lay each block next to another to create a sealed room where the player can have fun. The blocks can be various shapes and sizes and placed together in just about any orientation you can think of, allowing limitless possibilities for the construction of your map.

The blocks placed to make up the structures and constraints are *brushes.* Although brushes can be made into various shapes and sizes, they must be solid in form. This means that they must have at least four sides, like a three-sided pyramid with a floor.

With at least four sides, they can be defined as solid, structural blocks in the game, which is important. The game performs some optimization techniques to help it to run smoothly. Part of this optimization depends on the ability of the game to define what the player can and cannot see. Because the game assumes that the player cannot see through solid brushes, this helps with that optimization.

Another restriction that brushes have is that they must be convex in shape. This means that you can't have a single brush with a concave or U-like shape. This limitation reduces the work required by your processor during game play, and because you can place multiple brushes together to create any shape imaginable, it's a fair compromise to make.

## Setting boundaries

Within each map, you must create specific boundaries, usually through the placement of brushes. When creating buildings and other things in your map, you're confined to a single space. You have a large area in which to build — kind of like building in outer space. However, if your computer had to calculate a virtual world that went on without boundaries, it would quickly run out of free memory and processing power and then crash.

To avoid crashing the game, you must set the boundaries of your map. By creating a large, sealed room around all your other structures and elements, you can define these boundaries. Then, when the game begins putting together your 3D world, it won't lose its cool and crash on you. This game-creation process is *rendering,* which comprises making your data visible on the computer screen. Your computer takes the code that has been written to the map file and turns it into visual data that makes up your game.

## Seeing in three dimensions

Earlier, I refer to the environment as a 3D world. What I mean is that just like the world we live in, the game world also has three dimensions — the X, Y, and Z planes. One dimension runs from left to right; in *Half-Life 2*, this is the X plane. Another plane, the Y plane, runs from back to front. Last, the plane that runs from bottom to top is the Z plane. Together, they make up the three planes of the 3D environment. Figure 3-1 helps to illustrate these three planes.

TIP

Those of you with a background in mathematics or modeling might be confused by this configuration of axes. You would be more familiar with a vertical *x* axis and a horizontal *y* axis. However, in the world of gaming, these axes are reversed. This is because many games, such as *Half-Life 2,* use a Cartesian coordinate system, which defines the third axis — the height — as the *z* axis.

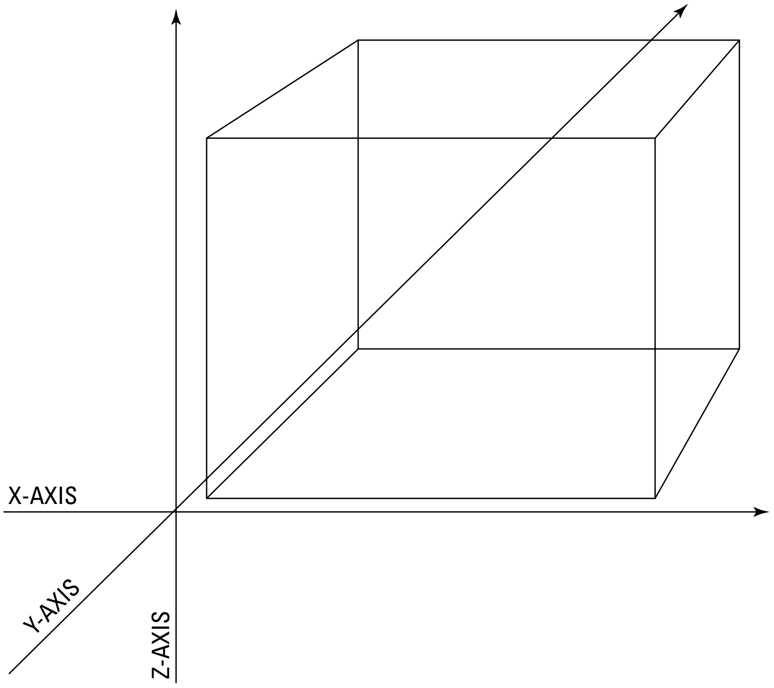


**Figure 3-1:** This figure is pretty plane, but it should get the point across.

If you were to measure a box, like the post office does before shipping, you measure in these three dimensions. The width, length, and height of the box relate to the X, Y, and Z planes, respectively.

### Measuring in units

The game world has its own way to express distance. Instead of using inches or centimeters, it uses units. Although it isn't easy to picture how big a unit is in reference to real-world objects, it does relate to another digital medium of measurement: the pixel. One unit in the game is equal to 1 pixel. This might be confusing at first, but after a week of being submersed in the mapping world, it becomes second nature to you.

Measuring things this way makes it easy to create images for use within the game, but not everyone likes it. A large number of modders would prefer that this measurement relate directly to real-life measurements so they can more easily reproduce in the virtual world real-life environments, such as their homes or offices. If you are one of these modders, the real-life conversion of units to inches is listed in Chapter 16. There you can also find other helpful measurements for reference when building your maps.

## Toying with Textures

The virtual world of games is full of images. I don't just mean the images invoked in your mind of the wonderful things you can do. Everything that you see in the game starts as unpainted objects. The walls begin as plain boxes, and the players start out lacking color, definition, and everything else that makes them look like players.

### Painting the walls

When the game is being built, the person doing the building puts the color into the game. This is done by placing images on everything, but these are not the same kind of images you can get with your camera. These images have additional features specified within them. These features define how light bounces off the image, how the image can convey the appearance of little bumps and scratches and other similar things. Altogether, these images and features make up a *texture.*

Textures carry a very important role in the game: They provide the color and other features that would otherwise be missing. Without textures, everything you see in the game would look flat, colorless, and plain.

### Building interest

Textures do a lot more than just add color to a scene. They also define bumps, scratches, and other forms of dimension. Sure, color is the primary function of a texture, but it also does quite a bit more.

While you're playing a game, your computer has to render the 3D environment. The more rendering that your computer has to do, the harder it has to work. So, if you have a wall with complex trimming mounted near both the floor and ceiling and if you also have wood paneling with more trim along the middle of the wall, all this adds up to a lot of dimension. All the trim would stick out from the surface in different ways, and indentation would occur between each of the boards of wood paneling. This level of detail causes a great deal of rendering.

With textures, you can fake most of this bumping so your computer doesn't have to do it. The less work your computer has to do when creating its environment, the faster it can provide you with an exciting and fun game. This also means the structural part of your wall doesn't have to be so complex. You can create a flat wall and make it look more complex simply with the use of textures.

I show you how to create these textures in Chapter 15. I show you how to go from simple color to the highly detailed bumping and shining that a texture can bring forth in the game.

## Evoking Entities

*Entities,* in very basic terms, are the opposite of structural brushes. Brushes are the building blocks of a map. They help to define what the player can and cannot see within the game.

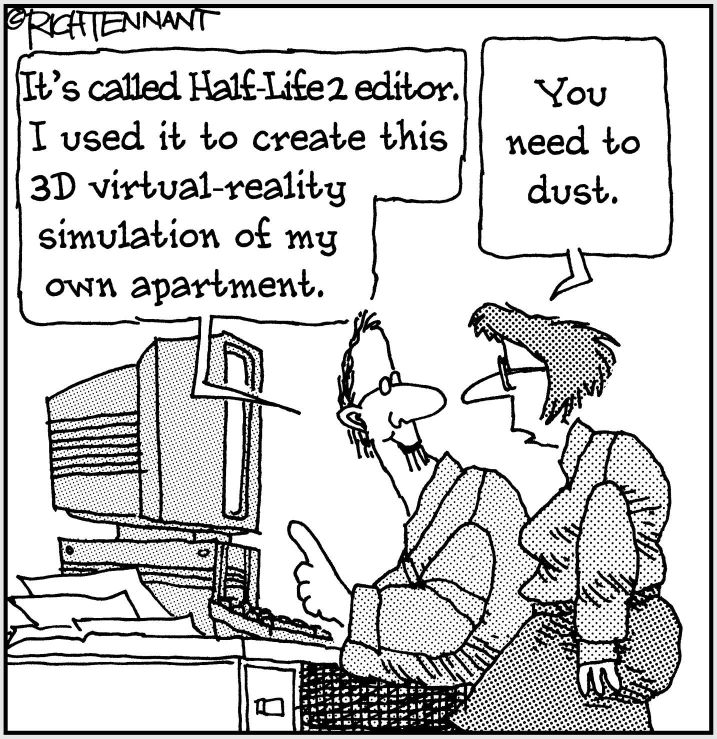
Entities, on the other hand, don't provide any structure to the game. However, they do provide activity and interactivity. If you walk up to a door and it opens or moves as a result of something that you did, that door is an entity. If you walk up to a control panel or an elevator or anything else that reacts to your action, it is an entity within the game.

That door or control panel is a type of entity known as an *active brush;* however, another type of entity is a point entity. A *point entity* has an effect on the environment in some way, but it isn't quite as interactive. Point entities are small, single points that affect the game environment (for example, a light). Alternatively, a point entity can allow another entity to focus on it. You could add a light to the game, instruct that light to act as a spotlight, and point to another point entity as a focal point. In both cases, the point entity exists as a nonsolid object in the game that exists in a single point as opposed to the door, which is solid and consists of several points for each of its corners.

# Part II
# Making Your Own Maps

The 5th Wave

By Rich Tennant


©RICHTENNANT
It's called Half-Life 2 editor. I used it to create this 3D virtual-reality simulation of my own apartment.
You need to dust.

## *In this part . . .*

Game modification is an extremely broad subject. It can span from programming lines of code to creating the cut scenes that play between each level. However, the most popular mods for almost any game out there take players someplace they've never been before.

It's time to get set up and into gear. In this part, you install some modding tools and start making your first custom game level. By the end of this part, you'll have something to show your friends, and you will officially be a modder.

Chapter 4

# Getting Set Up for Mapping

### In This Chapter

- ▶ Downloading the Hammer Editor and tools
- ▶ Launching and configuring the editor
- ▶ Getting to know the interface
- ▶ Using shortcuts

Before you can begin working on your new level, you need to install the Hammer Editor. Thanks to Steam, locating and installing this editor is really quite simple.

To make changes to the first *Half-Life* game, you had to download and install WorldCraft. This third-party map editor makes you follow specific (and sort of tricky) installation instructions. If you hadn't worked with WorldCraft before, installation wasn't a trivial process. You had to install the editor in the correct directory, on the same hard drive as the game, and with specific settings. Unless you did everything perfectly, the editor wouldn't work.

This process is much simpler now thanks to the Steam downloader. If you installed the full version of *Half-Life 2,* you installed Steam and then used it to register your *Half-Life 2* game.

## Installing the Hammer Editor

Hammer Editor is the Valve Software mapping editor. With this utility, you can make levels for the game. Before you can access this utility, though, you need to install it — and as with all the games delivered by Valve, via Steam.

To begin, start the Steam application. *Steam* is the central location to find and launch your games as well as update existing games and download new ones.

By default, Steam is set up to automatically start with Windows. You know it's running when you see its icon, as shown here, displayed in your system tray.

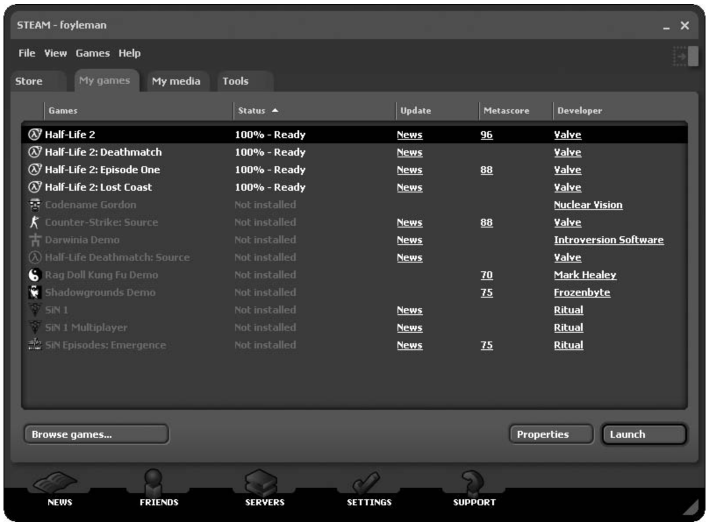
If Steam isn't already running, you launch it. Steam's location on your Start menu differs depending on how you installed *Half-Life 2.* If you installed Steam before installing *Half-Life 2,* you find the launch icon by choosing Start➪All Programs➪Valve➪Steam. If you installed *Half-Life* 2 before installing Steam, you find the launch icon by choosing Start➪All Programs➪Steam➪Steam.

Before installing the tools that you use for your mod creations, make sure you already installed the game types for which you want to create mods and that you have played them at least once. In this book, I take you through the steps of creating mods for *Half-Life 2: Deathmatch.* Although modification for any of the game types is pretty much the same, I figure you want to be able to show off your custom content with your friends and family in a multiplayer environment. The deathmatch game type is perfect for this.

If you have not played the game types that you plan to mod since the installation of the game on your computer, do so now. If you don't play them before attempting to edit them, errors will occur. Certain game configuration files must be created to avoid editing errors, and these configuration files are created automatically for you when you play each game type.

However, because you're going to be working on mods for the deathmatch game type, you must make sure that *Half-Life 2: Deathmatch* has also been installed. Here's how.

From within the Steam application, open the My Games tab. Within the list of games on this page, make sure that *Half-Life 2: Deathmatch* is listed with a status of 100% - Ready, as shown in Figure 4-1. If it isn't, right-click *Half-Life 2: Deathmatch* and then choose Install Game.



| Games | Status | Update | Metascore | Developer |
|---|---|---|---|---|
| Half-Life 2 | 100% - Ready | News | 96 | Valve |
| Half-Life 2: Deathmatch | 100% - Ready | News | | Valve |
| Half-Life 2: Episode One | 100% - Ready | News | 88 | Valve |
| Half-Life 2: Lost Coast | 100% - Ready | News | | Valve |
| Codename Gordon | Not installed | | | Nuclear Vision |
| Counter-Strike: Source | Not installed | News | 88 | Valve |
| Darwinia Demo | Not installed | News | | Introversion Software |
| Half-Life Deathmatch: Source | Not installed | News | | Valve |
| Rag Doll Kung Fu Demo | Not installed | | 70 | Mark Healey |
| Shadowgrounds Demo | Not installed | | 75 | Frozenbyte |
| SiN 1 | Not installed | News | | Ritual |
| SiN 1 Multiplayer | Not installed | News | | Ritual |
| SiN Episodes: Emergence | Not installed | News | 75 | Ritual |

**Figure 4-1:** The My Games tab lists games that have been (and can be) installed.

When installation of *Half-Life2: Deathmatch* is complete, you're ready to install the modding tools for the game. Steam makes it just as easy to install the tools as it does to install the various games by combining the downloading and installation process into a single step. Here's how:

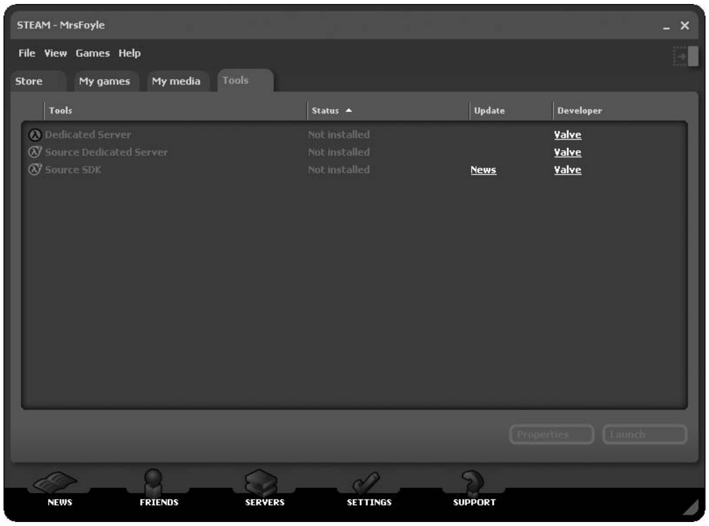
1. **Make sure that Steam is open.**

   If Steam isn't open, right-click the Steam tray icon and then choose Games from the list. This launches a window with a group of tabs at the top, like the ones shown in Figure 4-1.

2. **Select the Tools tab from the Steam window.**

   This displays a list of installable tools for the different games available through Steam, as shown Figure 4-2. The tool that you need is Source SDK. SDK refers to the Software Developers Kit, which contains all of the applications and files you need to mod the game.

3. **Right-click Source SDK and then choose Install Game from the list of options.**



| Tools | Status ▲ | Update | Developer |
|---|---|---|---|
| Dedicated Server | Not installed | | Valve |
| Source Dedicated Server | Not installed | | Valve |
| Source SDK | Not installed | News | Valve |

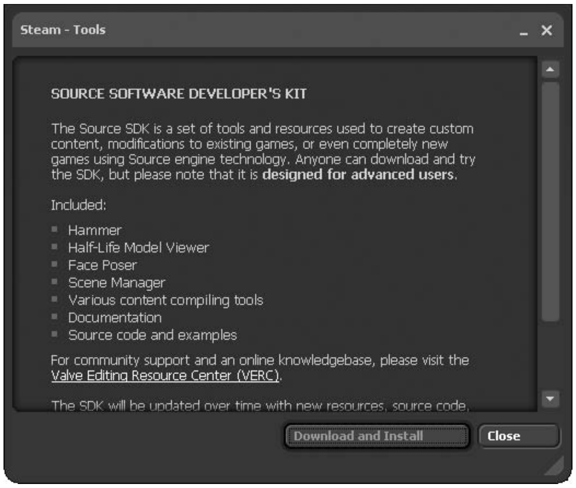
**Figure 4-2:** Right-click and install Source SDK from the Tools tab in Steam.

REMEMBER

If you haven't installed and activated *Half-Life 2* through your Steam account, you won't have the option to select and install Source SDK. This option is available only after installing *Half-Life 2*.

A new window pops up in which you must confirm that you want to download and install Source SDK. (See Figure 4-3.)

4. **Click the Download and Install button.**


Steam - Tools

SOURCE SOFTWARE DEVELOPER'S KIT

The Source SDK is a set of tools and resources used to create custom content, modifications to existing games, or even completely new games using Source engine technology. Anyone can download and try the SDK, but please note that it is **designed for advanced users**.

Included:

- Hammer
- Half-Life Model Viewer
- Face Poser
- Scene Manager
- Various content compiling tools
- Documentation
- Source code and examples

For community support and an online knowledgebase, please visit the Valve Editing Resource Center (VERC).

The SDK will be updated over time with new resources, source code,

**Figure 4-3:** Click the Download and Install button to continue with the installation.

5. **Pass through a series of windows to confirm the installation of the Source SDK tools as follows:**

   a. *Confirm your drive space and download time.*

      If you have enough disk space on your hard drive and the estimated download time is acceptable, click Next to continue. If you don’t have enough drive space, you might need to install another hard drive in your computer to continue.

   b. *Select the Create a Desktop Shortcut to Source SDK option.*

      Although you can access these tools right from the Steam application, I recommend that you select the option to create a desktop shortcut. While you’re getting familiar with everything, the shortcut makes things a lot easier. Click Next to continue.

   c. *Finish and begin downloading.*

      Leave selected the option to open the My Games tab and then click Finish to begin your download.

When Source SDK completes its download, it’s listed in the Steam Tools tab with a status of 100% - Ready, and you have the option to launch the program.

## Launching for the First Time

You now have a couple options for launching Source SDK. If you took my recommendation to check the “Create a Desktop Shortcut” option, you have a quick and easy way to launch the program from your desktop. Otherwise, you can also always find your Steam installed applications via the Steam application itself. Source SDK is listed in the Steam Tools tab.

Choose either option to launch Source SDK.

The first time you launch Source SDK, you have to go through a few steps to set up everything for use. This setup provides information on how the program should run the next time it's launched. The setup process goes like this:

1. **Confirm that the program is ready to launch.**

   The program is ready to launch if the status is listed in your Steam window as "100% - Ready."

2. **Wait while the program copies files to your hard drive.**

   When you download Source SDK, all the installation files are compressed into a single file to make them easier to download. The first time you launch Source SDK, this compressed file will decompress and place the individual files where they need to be on your hard drive. This is the copying process.

3. **Launch the Hammer Editor.**

   Double clicking the Hammer Editor application will launch the editor. If Hammer Editor loads up without any warnings, you're ready to go. Otherwise, continue to the next step.

4. **If an error regarding missing configuration files pops up, reset your configuration files.**

   If no configuration files are found for the deathmatch game type, you are notified and have to resolve the issues with a couple of steps outlined in the next part of this chapter.

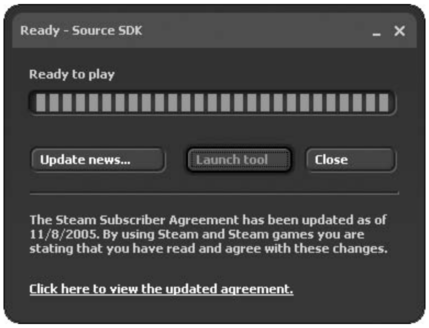
5. **Manually configure Hammer Editor, save the settings, and finally load the editor.**

   After the game has installed the basic configuration files, you may want to manually adjust some of the configuration settings. These settings are defined in the part, "Manually configuring Hammer."

After you perform the preceding steps, the first window you see lets you know that the application is ready to run. (See Figure 4-4.) Because you just downloaded this program, the window is nothing more than a confirmation window. Click the Launch Tool button to continue.

After launching the tool, the necessary files are automatically expanded and copied to the proper locations. You just have to sit back and wait a minute or two while this happens.

After the files are done copying, the Source SDK window appears listing all the available tools you have to work with. (See Figure 4-5.) The links and applications listed here are what you primarily use to create your mods for *Half-Life 2.* The list of options and what they do are as follows:


Ready - Source SDK

Ready to play

The Steam Subscriber Agreement has been updated as of 11/8/2005. By using Steam and Steam games you are stating that you have read and agree with these changes.

Click here to view the updated agreement.

**Figure 4-4:** The first time Source SDK runs, a window tells you it's ready to go.

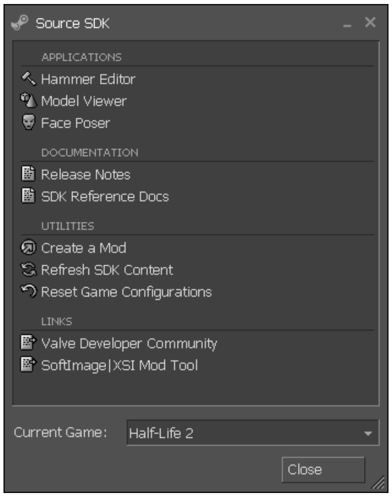


**Figure 4-5:** Source SDK has another menu of options from which you can select.

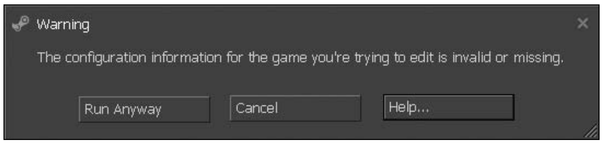
- **Hammer Editor:** This application is the map editor that allows you to create new levels for the game.
- **Model Viewer:** This application allows you to preview the models that are used within the game in a separate 3D environment. You can look around the model at all angles. This utility is especially useful when you create and add your own models to the game. It allows you to make sure that everything looks the way you want it to before you place it in the game.
- **Face Poser:** In the single-player game, you might want to add a character that says something. This application allows you to sync the character's mouth and lips to what he or she is going to say.
- **Release Notes:** This is a link that opens in your browser and takes you to the developer Web site for Valve. This page lists the features and upgrades that have occurred in each version of Source SDK that has been released.

- **SDK Reference Docs:** Another online document, this page provides you with additional documentation, tutorials, and other helpful information to assist you in your quest to make great mods.
- **Create a Mod:** This utility assists you in the creation of a new modification for the game. It sets up folders and options for the applications that are necessary when you modify the game's original content. If you're simply adding content to the game, this utility isn't necessary. However, if you're modifying existing files in the game, this comes in handy so you don't end up making permanent changes to the game.
- **Refresh SDK Content:** Some configuration files and related programs used by the application and utilities in the SDK can become corrupt. If they do, you can use this utility to restore the original files to their original condition when you first installed the SDK.
- **Reset Game Configurations:** If you run into problems with the applications in this SDK, you might find that resetting the configurations helps. This utility resets only the configuration files to their original condition when you first started using the SDK.
- **Valve Developer Community:** This is a Web link to the Valve Developer Community main page. This is a good place to find links, tutorials, and other helpful material related to *Half-Life 2* and modding.
- **SoftImage|XSI Mod Tool:** This is a link to a modeling program. You can use it to create new models for the game, such as weapons, characters, or anything else that's not already available. This program doesn't come with Source SDK because it's not a product directly supported by Valve. However, this link is provided to give you a head start on making custom models.
- **Current Game:** Here you can choose the game that you want to work on in the various applications listed earlier.

As I mention in the preceding list, the first application to explore is the Hammer Editor. This application allows you to make custom levels for the game. To continue the setup process required for the first use of Hammer Editor, launch the application. You can do so either by double-clicking its name in this window or by right-clicking its name and choosing Launch from the list.

## Dealing with configuration errors

If you launch Hammer Editor and see a pop-up warning window like that shown in Figure 4-6, something needs to be fixed before you can continue. Generally, this error shouldn't pop up. If it does, though, the necessary configuration files haven't been automatically generated yet. Therefore, you have to do a couple things to get the game to generate them manually.



Warning

The configuration information for the game you're trying to edit is invalid or missing.

**Figure 4-6:** A warning window might occur the first time you launch Hammer Editor.

Most commonly, this error occurs if you try to load Hammer Editor without first accessing the game. So, load *Half-Life 2: Deathmatch* and then close it by following these steps:

1. **Click Cancel to close this warning window.**
2. **Click Close to close the Source SDK window.**
3. **If your Steam window isn't open anymore, open it by double-clicking the icon in your taskbar.**
4. **Select the My Games tab and double-click Half-Life 2: Deathmatch to launch the game.**
5. **After the game completes loading, close the game.**

   To close the game, click Quit in the game.
6. **Reopen the Source SDK window. Go back to the Tools tab within the Steam window and then double-click Source SDK to launch it.**
7. **Before launching the Hammer Editor, first double-click the Reset Game Configurations link.**

   This makes sure that your configuration files are reset and ready to run. A window pops up asking you to confirm that you want to reset your configuration.
8. **Click Reset to continue and then click OK in the next window that is confirming the reset operation to close the Hammer Editor.**

Hopefully, these additional steps resolve any errors you encounter. To find out, launch the Hammer Editor again by double-clicking its name in the Source SDK window. If you see the error again, you need to proceed with the next section. If, on the other hand, the error is fixed and everything is fine, continue your investigation of the Hammer Editor by skipping to the section, "Exploring the Interface," later in this chapter.

If when you relaunch the Hammer Editor application you see a window asking if you want to run the application anyway, click the Run Anyway button to continue. The next section will outline the manual configuration steps to follow to ensure Hammer Editor runs correctly.

An error will be displayed if you have not yet played the game type you are attempting to edit. In this case, make sure you have played *Half-Life 2: Deathmatch* at least once or an error will occur.

## Manually configuring Hammer Editor

Another window pops up, alerting you that this is the first time the program is running and also asking you whether you want to view the new user's guide to set up Hammer Editor. Click Yes to view it. However, you don't have to view this now because the required setup steps are outlined later in this book. If you choose to not read the new user's guide from Valve, click No.

Now, you're in front of the Configure Hammer window. This window establishes the required settings to make everything run. To configure Hammer Editor manually, you follow exactly the steps listed to avoid any errors.

The Configure Hammer window, as shown in Figure 4-7, has tabs at the top that allow for a specific set of options to be established. The tabs are

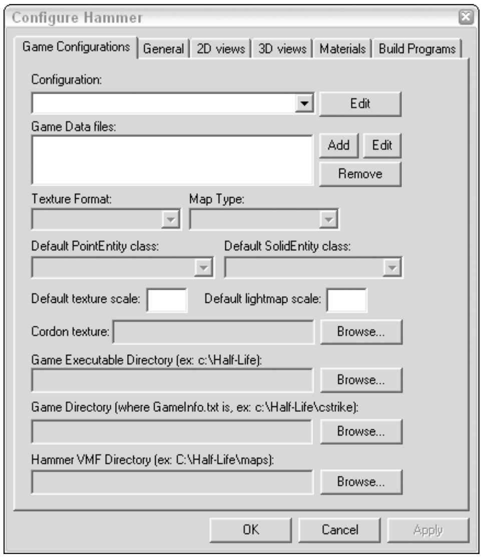


**Figure 4-7:** The Configure Hammer window sets up the options required for the editor.

- **Game Configurations:** The configurations for the game define the name of your configuration as well as the location of the files associated with the particular game type that you're modding. Because you can mod for single-player, multiplayer, and other game types separately, you must first define in this tab where the available game options are found.

- **General:** Here you can set up user preferred options. These options can be adjusted at any time, so for now, use the default settings in this tab.
- **2D Views:** This tab offers further user preferred options. Leave these options with their default values.
- **3D Views:** This tab offers further user preferred options. Again, leave these options with their default values.
- **Materials:** This tab, used to adjust the options of the textures and materials used in the editor, is currently unavailable to you. It becomes available after the first setup is completed.
- **Build Programs:** Here you define the locations of each program used in the level-building process. Your level starts as a simplified map file and must then be turned into a level to be played in the game.

Although these settings can all be adjusted later from within Hammer Editor, a select few settings must be set prior to the first time you launch the application. I point out these settings in detail.

### Adding configurations

The first thing you need to do is create a name for your configuration. Do so by following these steps:

1. **Select the Game Configurations tab from within the Configure Hammer window.**
2. **Click the Edit button to the right of the Configuration field.**

   This opens a new window, as shown in Figure 4-8.

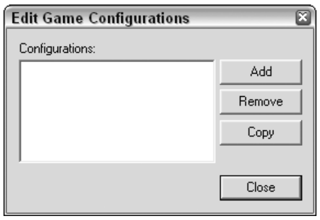


**Figure 4-8:** You can create multiple configurations for Hammer Editor.

3. **In the Edit Game Configurations window, click the Add button to add a new configuration name.**
4. **In the resulting text box, enter** HL2: Deathmatch **and then click Close to continue with the other settings.**

Because you're going to be working on levels for the deathmatch game type, this is an appropriate name to start with.

5. **Click the Add button to the right of the Game Data Files text box.**

   The game data file defines the different options available within the game type you're editing. For example, if you're playing *Counter Strike Source,* you might want to define an area within the level where players can purchase weapons, ammo, and armor. This is done with the game data file. (However, this option is not available when playing in a deathmatch level.)

   When you click Add, a window pops up that allows you to select a game data file (`.fgd` file). This window opens automatically to the folder

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\sourcesdk\bin
   ```

   where *`username`* is your Steam user name, and *`Valve`* is optional based on your installation.

6. **Because you're making a deathmatch level, select the `hl2mp.fdg` file located in this folder and then click Open.**

   Other optional game data files are

   - *Counter Strike Source maps:* `cstrike.fgd`
   - *Day of Defeat:* `dod.fgd`
   - *Half-Life 2 single-player:* `halflife2.fgd`

   Based on the configuration name and game data file that was entered, only the last three fields in this tab remain to be filled in. The others automatically fill in with the following data:

   - *Texture Format: Material (Half-Life 2):* This defines the material files that are used, which in turn defines the texture available when building your maps. The only other option would be for *Team Fortress Classic* maps.

     *Team Fortress Classic* is a game type from the original *Half-Life* game. This game type allows you to select a character to play that has specific characteristics, such as a medic who can heal other players or a demolition man who can plant different types of explosives. The different characters combine to make up a team, and in *Team Fortress Classic,* two teams battle it out for dominance in a game.

   - *Map Type: Half-Life 2:* This defines the map files that are built. The only other option is for *Team Fortress* maps.
   - *Default PointEntity class: info_player_deathmatch:* This defines the default point entity available in *Half-Life 2* when you select which one to insert into your map. In most cases, *info_player_deathmatch* is the entity selected by default. (Entities are defined further in Chapter 3.)

- *Default SolidEntity class: func_detail:* This defines the default solid entity available in *Half-Life 2* when you select which one to insert into your map. In most cases, *func_detail* is the entity selected by default.
- *Default Texture scale: 0.25:* Texture scaling must be set to 0.25; otherwise, the textures won't look right when you compile the map and play it in the game. This scale is actually the equivalent of 1 pixel to 1 unit within Hammer Editor. I discuss this in greater detail later in the book.
- *Default Light map scale: 16:* The light map scale must be set to 16; otherwise, you might end up with errors while compiling your map. This scale represents how defined the lighting is on each wall created in Hammer Editor.
- *Cordon texture: tools\toolsskybox:* This defines the material or texture used when defining a cordon volume in the editor. A *cordon volume* is an area of your map that you boxed out of the map that you're working in. It allows you to section a portion of the map for testing but to ignore the rest. In most cases, *tools\toolsskybox* is the entity selected by default.

The next field to be defined in your Game Configurations tab is the Game Executable Directory. This directory is where the game's `.exe` file is found.

7. **Click the Browse button to open a window that allows you to select a directory on your hard drive. Then, within this new window, select the directory**

```
C:\Program Files\Valve\Steam\SteamApps\username\half-life 2 deathmatch
```

**where** `username` **is your Steam user name, and** `Valve` **is optional based on your installation.**

The other possible game type selections are

- *Counter Strike Source*

```
C:\Program Files\Valve\Steam\SteamApps\username\counter-strike source
```

- *Day of Defeat Source*

```
C:\Program Files\Valve\Steam\SteamApps\username\day of defeat source
```

- *Half-Life 2 single-player*

```
C:\Program Files\Valve\Steam\SteamApps\username\half-life 2
```

For the next field, the Game Directory field, you need to define where the `GameInfo.txt` file is located for the game type with which you are working.

8. **For the deathmatch game type, click the Browse button to the right of the text area and select the directory**

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\
             half-life 2 deathmatch\hl2mp
   ```

   **where** *username* **is your Steam user name, and** *Valve* **is optional based on your installation.**

   Other possible options are

   - *Counter Strike Source*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\
               counter-strike source\cstrike
     ```

   - *Day of Defeat Source*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\
               day of defeat source\hl2mp
     ```

   - *Half-Life 2 single-player*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\half-life 2\hl2
     ```

   The last field on this tab is the Hammer VMF Directory. This is where Hammer Editor saves the map data files required for compiling your map into a level for playing in the game.

9. **Click Browse to the right of this field and select the directory**

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\
             sourcesdk_content\hl2mp\mapsrc
   ```

   **for the deathmatch game type, where** *username* **is your Steam user name, and** *Valve* **is optional based on your installation.**

   Other options are

   - *Counter Strike Source*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\
               sourecesdk_content\cstrike\mapsrc
     ```

   - *Day of Defeat Source*

     ```
     C:\Program Files\Valve\Steam\SteamApps\
               username\ sourecesdk_content\dod\mapsrc
     ```

   - *Half-Life 2 single-player*

     ```
     C:\Program Files\Valve\Steam\SteamApps\
               username\ sourecesdk_content \hl2\mapsrc
     ```

### Specifying programs

Now skip over to the Build Programs tab. Here, you identify each of the separate programs that will be used to turn your map file (that you created in the editor) into a level that can be played in the game. The build process is explained in greater detail in Chapter 3, but for now, just enter the values outlined here into the fields.

1. **Select your new configuration from the drop-down list if you haven't already.**
2. **The game executable is the file that launches your game type. Because this is a deathmatch game type, click Browse and locate the deathmatch executable.**

   It is found in the location

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\
             half-life 2 deathmatch\hl2.exe
   ```

   where `username` is your Steam user name, and `Valve` is optional based on your installation.

   Other possible options are

   - *Counter Strike Source*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\
               counter-strike source\counter-strike source\hl2.exe
     ```

   - *Day of Defeat Source*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\
               day of defeat source\hl2.exe
     ```

   - *Half-Life 2 single-player*

     ```
     C:\Program Files\Valve\Steam\SteamApps\username\half-life 2\hl2.exe
     ```

3. **The BSP executable is the first of three steps in building your level. Click the Browse button and locate as well as select the file**

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\bin\vbsp.exe
   ```

   **where `username` is your Steam user name, and `Valve` is optional based on your installation.**

   The BSP is explained in detail in Chapter 8.

4. **Step two of the building process is the VIS executable. Click the Browse button and then locate a select the file**

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\bin\vvis.exe
   ```

**where *username* is your Steam user name, and *Valve* is optional based on your installation.**

5. **The third step of the building process is the RAD executable. Click Browse and locate as well as select the file**

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\bin\vrad.exe
   ```

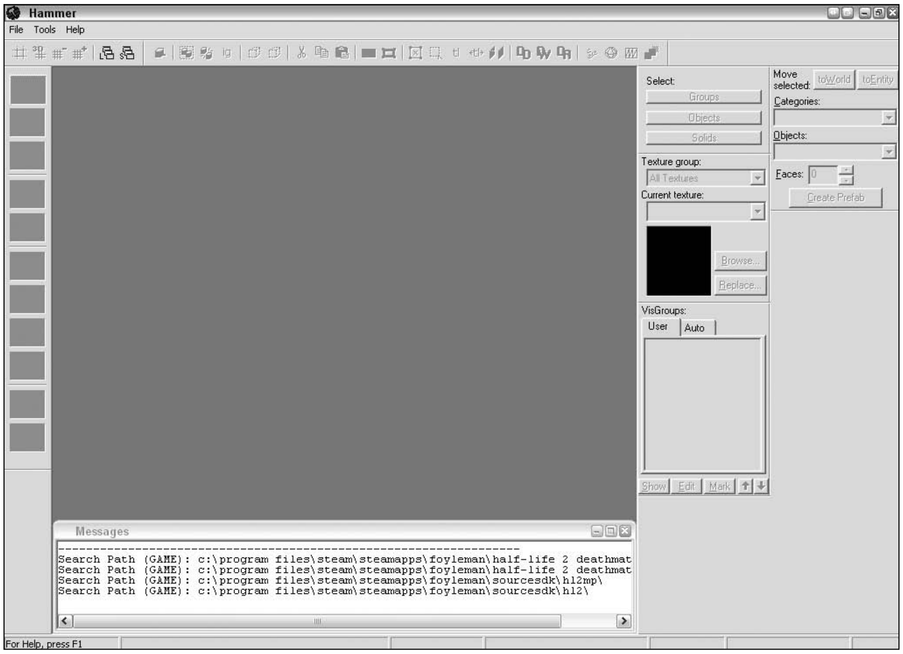
   **where *username* is your Steam user name, and *Valve* is optional based on your installation.**

6. **Finally, the compiled level needs to be saved where you expect the game to find it. Click the Browse button beside Place Compiled Maps in This Directory Before Running The Game field. Then locate and select the directory**

   ```
   C:\Program Files\Valve\Steam\SteamApps\username\
             half-life 2 deathmatch\hl2\maps
   ```

   **where *username* is your Steam user name, and *Valve* is optional based on your installation.**

That's it for the configuration of Hammer Editor. Click OK at the bottom of the window, and the editor loads, as in Figure 4-9.



```
------------------------------------------------------------------
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2mp\
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2\
```

**Figure 4-9:** The Hammer Editor when it first opens.

## Exploring the Interface

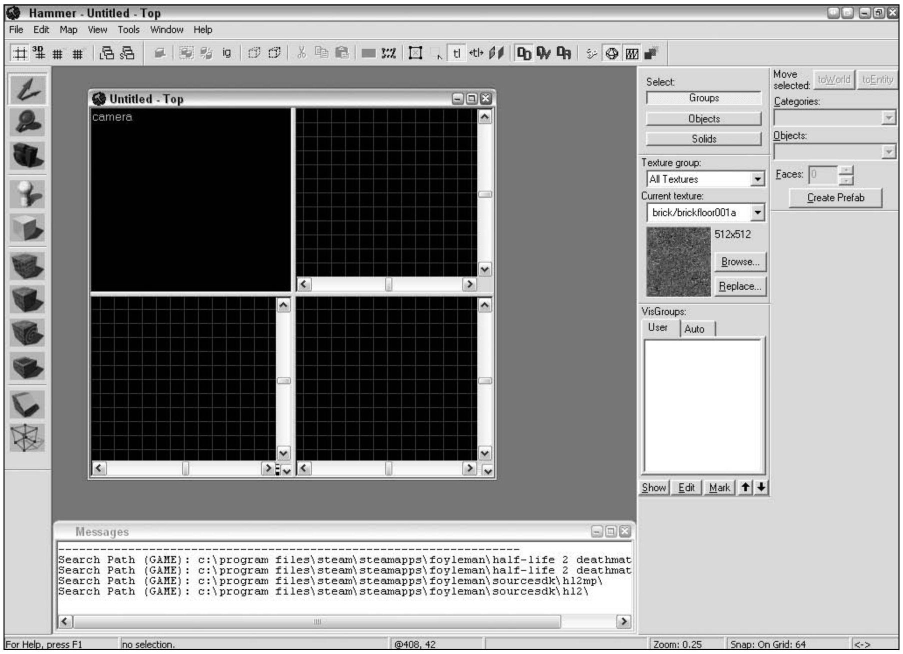
The editor might look somewhat intimidating when you first see it. It has multiple columns with different tools and settings. Unfamiliar words and a string of buttons are across the top of the editor, and you might not recognize them. Don't let this worry you. By the end of this chapter, you should feel much more comfortable with what's onscreen. You'll also be ready to plunge into the world of mapping. Chapter 5 shows you how to construct your first map.

To more easily figure out what's what within the editor, first start a new map. Right now, most of the options in the editor are either hidden or unavailable because you don't have any open maps to work with. Starting a new map activates these buttons so you can see them.

Choose File⇨New. This opens a new map, creates a new window, and enables all the tools in the editor. It looks like Figure 4-10.

TIP

It's possible that your editor displays more or less buttons and options. Valve regularly updates the programs you have installed. These updates are applied automatically. Although your editor may look slightly different, the option I cover in this book will generally apply to all versions regardless of the version of Hammer Editor that you are running.



```
-----------------------------------------------------------------
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2mp\
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2\
```

For Help, press F1 | no selection. | @408, 42 | Zoom: 0.25 | Snap: On Grid: 64 | <->

**Figure 4-10:** Creating a new map opens a new window and activates the buttons.

## Looking in 2D and 3D

Begin by going over the four windows now open in the middle of the screen. You see a single window that's split into four quadrants. One quadrant is the camera viewport; each of the other three quadrants represents a single-dimensional view in the three-dimensional world. You can resize this window as a whole within the editor by clicking and dragging any of the windows edges or corners. You can also move the dividers that split the windows by clicking and dragging them. This allows you more control over your work environment. However, for the time being, just leave everything as is while I explain each of the four views in detail.

TIP

If you move the dividers between the viewports and you want to reset them, press Ctrl+A. This automatically resets the sizes of the four views so that they're all equal.

### Camera viewport

The quadrant located in the top-left corner is the *camera viewport.* This is your 3D viewport. (See Figure 4-10.) The purpose of the viewport is to present you with a visual representation of your map. Right now it's empty; but later, when you start working on your map, this window depicts what you are working on as if you were in the game. The camera viewport also makes it a lot easier to select portions of the map that you're working on. I go into this further in Chapter 5.

When you roll your mouse over this viewport, the title bar of the window changes to `Untitled - Wireframe`. This means that this viewport is displaying content in wireframe mode. Wireframe is just one of several viewing options. The following is a list of the viewing options:

- **3D Wireframe:** When drawing the 3D content within this window, each object is represented as a line drawing rather than as a solid object. (See Figure 4-11.)

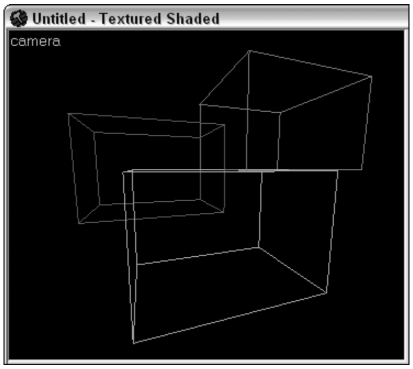


**Figure 4-11:** The 3D Wireframe viewing option of the camera window.

- **3D Filled Polygons:** Each object is filled in with color so that the object looks solid. (See Figure 4-12.)

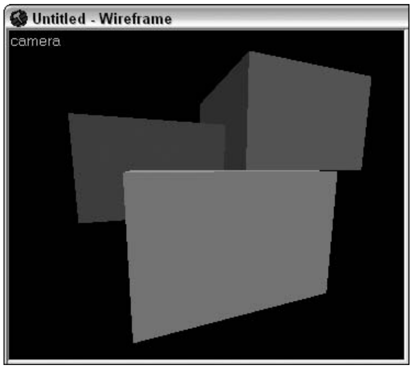


**Figure 4-12:** The 3D Filled Polygons viewing option of the camera window.

- **3D Textured Polygons:** Each object is displayed with its assigned texture similar to how it appears in the game. (See Figure 4-13.)

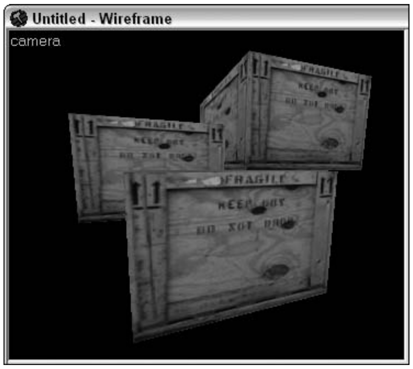


**Figure 4-13:** The 3D Textured Polygons viewing option of the camera window.

- **3D Shaded Textured Polygons:** This is very much like the previous setting except that shading is applied as light hits the textures. In this case, you won't see any difference at all because there isn't anything there to see. This view option is much more apparent later when your map is being constructed.

These modes are better understood when you start making a map and can see the difference as applied to your map. For now, know that the different 3D viewing options are available.

### 2D viewports

The other three viewports that are attached to the camera viewport are the 2D viewports. When you roll your mouse over each of these viewports, the window's title bar shows the viewport name. The top-right view is the Top view, the bottom-left view is the Front view, and the bottom-right view is the Right view. These three views together make up your 3D workspace in the editor.

Although you can change each of these viewports to any of the above-mentioned views, you should leave them at their default settings for now.

TIP

The View menu option in the editor refers to each of these three viewports in an X, Y, and Z format rather than Top, Front, and Right. The equivalent of each view is as follows:

- **Top:** 2D XY
- **Front:** 2D XZ
- **Right:** 2D YZ

## Pressing buttons and working menus

The buttons at the top of the window contain the commands most often used while working on your map. I don't describe all the buttons in this book. Actually, my goal is that by the end of this book, you won't even need any of the buttons. It's more efficient to know the keyboard shortcuts to all the commands in the editor than to rely on buttons and menu options. But before I go over the essential shortcuts in Chapter 5, let me introduce you to these buttons.

Menus are also available at the top of the screen. Here you can find all the editor options available. If you don't have the keyboard shortcut memorized or you don't see the button for the operation you want to perform, you can find it in this menu.

The buttons in the toolbars along the top of the Hammer Editor interface are described in Table 4-1.

**Table 4-1 Tools at the Top of the Hammer Editor Interface**

| *Tool Icon* | *Name* | *Function* |
|---|---|---|
| | Toggle Grid | Toggles the 2D grid on and off so that grid lines show up in the 2D views. |
| | Toggle 3D Grid | Toggles the 3D grid on and off so that grid lines show up on selected objects in the camera viewport. |
| | Smaller Grid (left bracket) | Decreases the size of your grid. |
| | Larger Grid (right bracket) | Increases the size of your grid. |
| | Load Window State | Loads your last saved independent window state. |
| | Save Window State | Saves your current independent window state. |
| | Carve with Selected Objects | When chosen, the selected object's shape is carved out of any overlapping objects. |
| | Group Selected Objects | Places all selected objects into a group so it's easier to move them together. |
| | Ungroup Selected Groups | Ungroups the selected groups. |
| | Toggle Group Ignore | Normally, when you select a single item in a group, the entire group is selected. This tool ignores groups so that individual objects can be selected. |
| | Hide Selected Objects | Removes from view all selected objects. This does not remove them from the map. |
| | Hide Unselected Objects | Shows all hidden objects. |

| *Tool Icon* | *Name* | *Function* |
|---|---|---|
| ✂ | Cut, Copy, Paste | These three buttons cut, copy, and paste (respectively) the selected objects in your map. |
| ▨ | Toggle Cordon State | Turns on and off the cordon state. |
| ⬚ | Edit Cordon Bounds | Selects or edits the cordon area. A cordon area partitions off a selected portion of your map so you can play-test this area without regard to the rest of the map. It's a means of quickly testing small areas of your map. |
| ⊠ | Toggle Select-by-Handles | With this enabled, the objects in your map can be selected only by the center x rather than by clicking anywhere within the object. |
| ⬚ | Toggle Auto Selection | This option allows for easier selection of multiple objects in your map. With it, you can drag a selection box around a group of items in your map to select them. Without it, you must press Enter after dragging the selection box in order to make your selection. |
| t\| | Texture Lock | Allows you to move an object in your map without disturbing the applied texture. |
| ‹t\|› | Scaling Texture Lock | When you scale an object's size with this option turned on, the texture scales with it. |
| ▱ | Toggle Face Alignment | Toggles the automatic alignment of textures on objects. When turned on, the texture is aligned to the object. When off, the texture is aligned to the map's world coordinates. |
| ▣ | Displacement Mask Solid | Turns on or off the display of the non-displacement sides for a brush that contains displacement surfaces. |

*(continued)*

**Table 4-1 *(continued)***

| *Tool Icon* | *Name* | *Function* |
|---|---|---|
| | Displacement Mask Walkable | Highlights a surface in yellow if it is too steep for the player to walk on. Note: This works only in the 3D Textured views. |
| | Displacement Mask Alpha | Turns on or off the display of the vertices that are collapsed when two displacements of different resolutions are connected via a Sew command, or a brush that contains displacement surfaces. |
| | Run Map! [F9] | Loads and runs your map in the game. F9 is the shortcut for this action. |
| | Toggle Helpers | Helpers are words that assist in identifying selected entities. When Toggle Helpers is turned on and you select one or more entities in your map, you can zoom in to your map and see the name of the entity. |
| | Toggle Models in 2D | Here you can show or hide models in the 2D windows. |
| | Toggle Model Fade Preview | Turns on or off the model fade preview in the 3d view window. |

When you click the Load Window State button, the four viewports in the editor separate into four windows. This allows you to customize differently each window in the editor, but it also means that you have to work with four windows instead of one. After you click this button and separate your windows, you can return the viewports back to their previous state by choosing Tools➪Options and deselecting the Use Independent Window Configurations option on the General tab. Afterward, you must close and restart Hammer Editor for your settings to take effect.

The buttons that run down the left side of the editor are for adding and modifying objects in your map. With these, you can add brushes, entities, apply textures, and so on. A breakdown of what these buttons can do for you is shown in Table 4-2.

**Table 4-2 Tools along the Side of the Hammer Editor Interface**

| *Tool Icon* | *Name* | *Function* |
|---|---|---|
| | Selection Tool | This tool allows you to select an object in your map. Press and hold Ctrl while selecting multiple objects. |
| | Magnify | With this tool, left-click to zoom into a specific area of your map. Right-click to zoom out. |
| | Camera | This tool allows you to place and modify cameras in your map. These cameras help you to take a close look at what you're working on. |
| | Entity Tool | This tool allows you to place entities in the map. |
| | Block Tool | Select this tool to add new brushes. |
| | Toggle Texture Application | To apply or modify a texture to a brush, select this tool. |
| | Apply Current Texture | If you already have a texture selected, this tool applies it. |
| | Apply Decals | Use this to apply decals to objects. Decals are like textures and can be applied over the top of other textures. |
| | Apply Overlays | This is very similar to the Apply Decals tool except that overlays have more options in the editor. Chapter 5 dives into the subject of textures and decals in greater detail. |
| | Clipping Tool | With this tool, you can cut brushes to create various shapes other than the common cube. |
| | Vertex Tool | To further manipulate brushes to the desired shape, use this tool. It allows you to move the corner points around in the editor. |

A group of four tools is also on the right side of the editor. (Refer to Figure 4-10.) These groups are portable in that you can freely move them and arrange them however you desire, which is something I recommend. After I define each of these groups, I go over the steps for arranging them.

- **Select Modes:** Use this group to control how map objects are selected. This can help if you want to make sure that you don't select the wrong thing when working on a complicated map.
- **Textures:** This group is a shortcut to selecting textures. The other method is to select your brush, select the Toggle Texture Application button on the left, select a texture, and then apply it. Instead, select a texture in the Texture group on the right and then use the Apply Current Texture button on the left to apply it to a brush.
- **Filter Control:** With Filter Control, you can simply select or deselect groups and objects to hide or show them in your map. When working on large, complicated maps, this comes in handy.
- **New Objects:** This group helps when placing entities, prefabs, and brushes. For instance, when placing brushes, you can choose to place a square, cylinder, spike, or other shaped brush. You can also define if a selected brush or group of brushes should be separated into world brushes or grouped as a single entity.

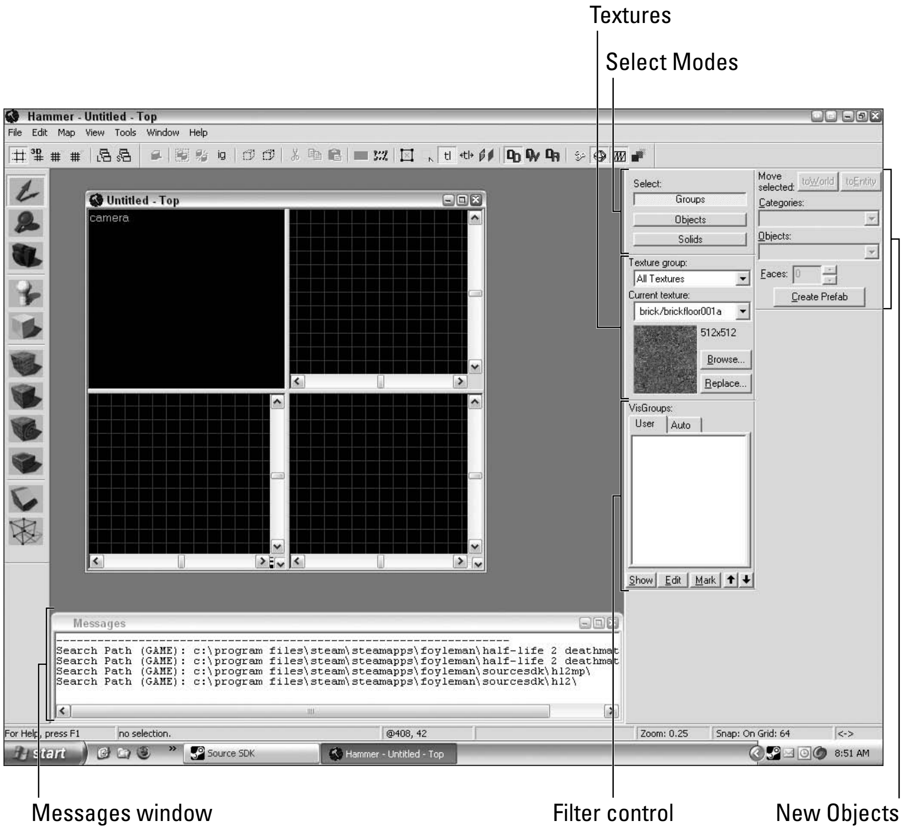
As for arranging these windows, I recommend condensing them into a single column (if they aren't already). How they're arranged when you first load the editor is dependent on your screen resolution. If your resolution is set to 1024 × 768, you have two columns of tools taking up valuable workspace, as shown in Figure 4-14.

To condense these grouped tools into a single column, click and drag the upper-left corner of the group and then move it where you want. In this case, start with the Select Modes group. Left-click the upper-left corner of this group and then drag it under the New Objects group.

Continue with the remaining two groups in the left column, moving them to the right column. The result looks like Figure 4-15. Yes, I know that the group at the bottom of the column is cut off and slightly out of view, but as long as you can still access all functions, you won't have any problems. You have more room to work on your map, which is a huge benefit.

## Reading messages

The Messages window found near the bottom of the editor, as shown in Figure 4-14, is there to help you spot errors in your map or in the loading of the editor. If a problem occurs while loading the configurations files, it's listed in this window.



```
------------------------------------------------------------------
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2mp\
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2\
```

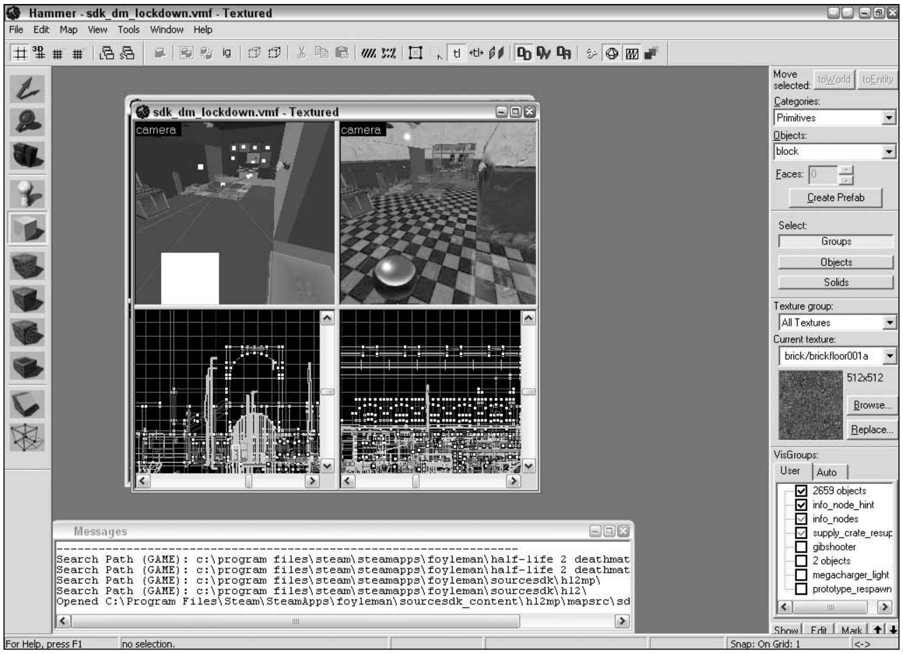
**Figure 4-14:** Two columns of tools can take up a lot of room in the editor.

You're going to find that this window isn't used very much. Rather, it takes up valuable workspace for your map. I recommend that you close this window. If at any time you want to see the Messages window, reopen it either by choosing Window➪Messages or by pressing Alt+F3.

To close this window, choose Window➪Messages, and it disappears.

Notice that I stretched it out across the bottom of the editor and shrank it in height. I did this to further increase my work area for the map while still being able to notice any important messages that might come up.

One last thing before you're done — maximize your map. The work area for your map, which contains the four viewports, is fairly small. This can make it difficult to work on your map because you have to constantly move to place and modify your map objects.



```
------------------------------------------------------------------
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\half-life 2 deathmat
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2mp\
Search Path (GAME): c:\program files\steam\steamapps\foyleman\sourcesdk\hl2\
Opened C:\Program Files\Steam\SteamApps\foyleman\sourcesdk_content\hl2mp\mapsrc\sd
```

**Figure 4-15:** Arrange your work area so that you can see most of your map.

To resolve this size issue, maximize your Mapping window. In the upper-right corner of the window, you have the standard window functions that you find in all the windows on your computer — the Minimize, Maximize, and Close buttons. Clicking the second button from the end maximizes your window to fill in all of the available space in your editor.

Click the Maximize button, and you have a nice working environment with lots of space to make your map just as I have in Figure 4-15.

## Using Shortcuts for Success

You should get to know the keyboard shortcuts for the editor. No quicker and easier way exists to work on your map other than when you're using keyboard shortcuts. You might find it difficult to remember them at first, but trust me when I tell you that it's worth the effort. The more you use them, the better you remember them.

If you forget a keyboard shortcut or want to look for one you don't know yet, you can often find it listed next to the operating command in the editor. For example, if the operating command is a button (like the Clipping tool), hover your mouse over the button, and the shortcut displays as a ToolTip for you. If the operating command is in the main menu, it might be listed next to the command name.

If you read through the upcoming chapters, important shortcuts are introduced. If you're looking for other references, flip over to the Cheat Sheet found at the beginning of this book.

## Troubleshooting Issues

In a perfect world, there would be no problems with the editor. You could load it up and start working regardless of the computer you're using. However, with so many different computers in the world, you can't possibly avoid all possible problems that might arise.

Some problems are more common than others. Throughout this book, I point out common errors and methods to fix them when appropriate. If you're experiencing a problem that isn't covered, I recommend that you search the Internet for a solution. It's quite possible that you're not alone and that the issue has been documented and solved already.

Chapter 5

# Creating Your First Game Map

### In This Chapter

- Selecting the texture for your walls
- Zooming and moving the 2D window
- Using brushes
- Changing your room from solid to hollow
- Keeping a seal on your room

It's time to jump right in and start creating your first game map. Your goal is to create a very simple room — just a small area with four walls, a ceiling, and a floor. This might not sound like much, but from this, you will be able to create great things.

Whenever I'm starting a map or testing out ideas, this is where I start. I create a small room with the basic necessities for playing in the game. Then I build my additions onto this small map. The additions could be an entire city with streets and skyscrapers that stretches on for blocks, or it could be something I want to invent and test, like a fancy door, before adding it into my final piece.

All great things start small, and this chapter is where you begin. In no time, you'll be building onto the small space you create and making something fantastic.

## Selecting a Texture

Before you begin creating the walls and parts of your room, I recommend that you select a base texture to apply to the things you draw. *Textures* are really just images that are painted onto the surfaces of what you see within the game. Just like the walls in your house, start with a *primer* (your base texture) and then go back and apply the *final coat* (what the player sees). I explain more about the base texture in the section, "Deciding on a base texture."

In order to access the available textures, you must first have a map open in the Hammer Editor. If the editor is closed, open it now so that you can start a new map. Choose File➪New, and you're ready to start on your custom level.

## Deciding on a base texture

After the editor is loaded and ready to go, you need to select an initial texture — the base texture — from which to build. While you're creating things within the editor, the selected texture is automatically applied to it. You can change the selected texture at any time, and you can even go back later and apply different textures. When your map grows in size and complexity, though, so does the tediousness of going back and replacing all the textures.

I start with the Nodraw texture, which is a good base texture because it's solid and doesn't take up additional memory in the game. Because the Nodraw texture makes solid walls, your map is self-contained. A map needs to be self-contained so that it can't leak, as I explain in further detail later in the section, "Leaking Is Not an Option." And because it's ignored by the game as a texture, the Nodraw texture doesn't take up any additional computer memory or processing power.

Even though the Nodraw texture is ignored by the game, you can still see it in the editor. When applied to a brush face, that brush face isn't drawn in the 3D world. This makes it perfect for places that can't be seen by the player, such as the areas behind walls, between brushes that are butted against each other (see the upcoming section, "Drawing the First Brush," for further explanation), and so on.

Of course, choosing a texture, such as brick or concrete, for these hidden areas would work as well. However, if you don't plan on using those textures elsewhere in the map, the game simply wastes memory by loading them in the game. And, when it comes to games, you don't want to waste anything because you want the player to have the best possible experience when playing your level.

## Filtering textures

To select the Nodraw texture, you must browse the available textures, find it, and then select it. From the Texture Group option on the right of the editor (see Figure 5-1), click the Browse button to open the list of textures.

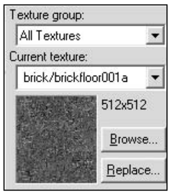
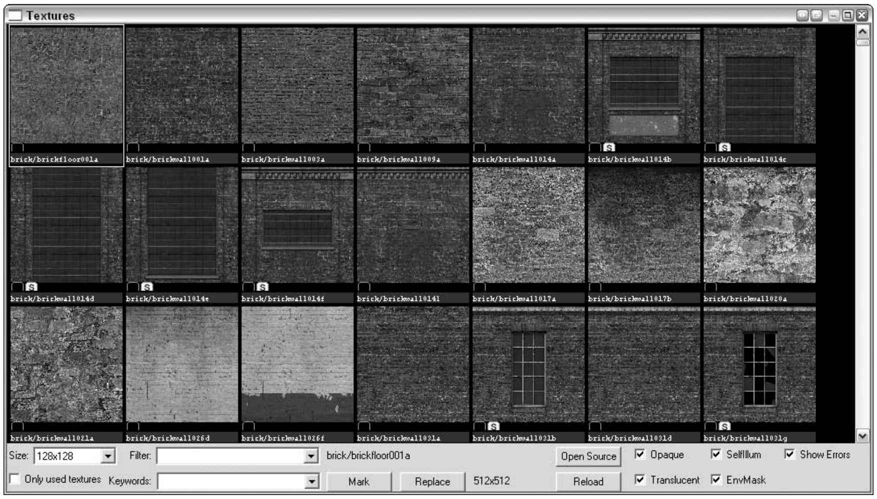

Texture group: All Textures
Current texture: brick/brickfloor001a
512x512

**Figure 5-1:** The Texture Group allows for quick texture selection.

After you click the texture's Browse option, a new window pops up with all the available textures for your map, as shown in Figure 5-2. Seeing all these choices is exciting until you realize that you're looking for only one specific texture in a list of many.



Size: 128x128
Filter:
brick/brickfloor001a
☑ Opaque
☑ SelfIllum
☑ Show Errors
☐ Only used textures
Keywords:
512x512
☑ Translucent
☑ EnvMask

**Figure 5-2:** The Textures window offers a detailed selection of textures.

Thankfully, a group of options that can simplify finding the right texture is at the bottom of this window. Here is a breakdown of the options in this window:

- **Texture Preview window:** Preview the textures in the main window. For each texture, you can find a square sample of the texture along with the texture's name, which should help you quickly scan and locate the texture of your choice. Double-clicking any of these textures automatically selects them for use in the editor.
- **Size:** Adjust the display size of the textures with the Size feature. Selecting a larger size prevents you from seeing as many textures in the area above.
- **Only Used Textures:** Select this option to filter out those textures not currently used in your map. Because you currently have no textures in your map, selecting this option would result in hiding all the textures from view. However, this is a great feature later when you're looking to reuse a texture already found in your map.
- **Filter:** This text box is a great search function. For example, typing **brick** into this box hides any texture that doesn't have *brick* in the title. You can then see all the textures that contain the word *brick.* You'll find that this feature gets a lot of use.
- **Keywords:** This option works very much like the Filter option. The difference is that the words you enter search through each texture's keywords instead of the titles. When *Half-Life 2* textures are created, keywords can

be separately defined. These words can be anything that the author deems suitable. Because you can't be certain of what turns up with keywords, this searching function might not be as useful as the Filter option.

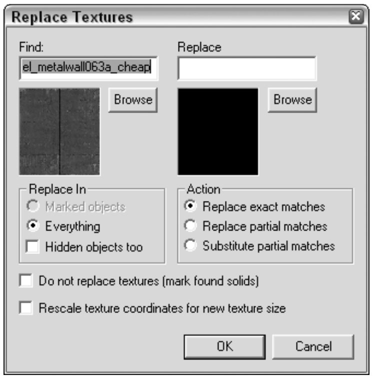
- **Mark:** Select a texture and then click the Mark button to select all the brushes or brush faces in your map with that selected texture. With this button, you can find where you used the selected texture in your map. You can also use this button to select and globally adjust all applied versions of the same texture throughout your map.
- **Replace:** Click the Replace button to open the Replace Textures window, as shown in Figure 5-3. With this window, you can find and replace any texture in your map. If you start your map with grass on the ground of some of your map's areas and later decide that you want to use dirt, this window makes replacing all the grass with dirt quick and easy.
- **Number × Number:** When a texture is selected, two numbers are displayed here that represent the texture's width and height. — number combinations, such as 64 × 64, 128 × 128, 256 × 256, and 512 × 512.
- **Open Source:** Click the Open Source button after selecting a texture to open the text file that defines that texture.

  TIP

  The textures in *Half-Life 2* are more than just images. They're also text files, which are saved with a VMT (Valve Material Texture) extension. In Chapter 15, I discuss textures in more depth.
- **Reload:** Click the Reload button to refresh the list of textures displayed.
- **Opaque:** Select this option to show only those textures that are fully opaque and don't have any transparency.
- **SelfIllum:** Select this option to show only those textures that are defined to give off light in the game.
- **Show Errors:** Select this option to show only those textures that have errors in their VMT file.
- **Translucent:** Select this option to show only those textures that have some form of transparency.
- **EnvMask:** Select this option to show only those textures that have an environmental mask defined. This is similar to the Specular Map created in Chapter 13.

To find and select the Nodraw texture, type "nodraw" into the filter text box and wait. The window then locates and displays all the textures that contain this search term. After this search is complete, you have only one texture to select — the tools/toolsnodraw texture (see Figure 5-4).

To select the Nodraw texture, double-click it. This closes the Textures window and displays the Nodraw texture in your Texture Group window on the right, as shown in Figure 5-5. You're now ready to start building your map.



Replace Textures

Find: el_metalwall063a_cheap

Replace:

Replace In: Marked objects [✓ Everything] Hidden objects too

Action: [✓ Replace exact matches] Replace partial matches Substitute partial matches

Do not replace textures (mark found solids)

Rescale texture coordinates for new texture size

**Figure 5-3:** Find and replace textures in your map quickly.

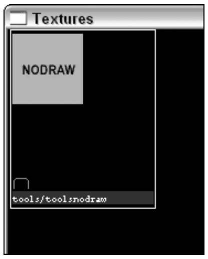
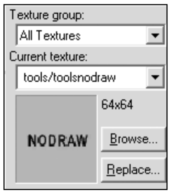


**Figure 5-4:** The Nodraw texture is yellow with *Nodraw* printed in it.



Texture group: All Textures

Current texture: tools/toolsnodraw

64x64

**Figure 5-5:** The selected texture displays in the Texture Group window.

## Drawing the First Brush

The world inside 3D games is made up of building blocks called *brushes.* If you think of this world as being a house, every wall, floor, and roof is made up of one or more of these brushes.

With your texture selected, you're ready to start creating your game world by drawing your first brush. To create a brush, click and drag a rectangular shape into the editor. Clicking and dragging a brush is *drawing.* I show you how to do this in the following steps:

1. **Press Ctrl+B or select the Block Tool icon on the left of the editor.**
2. **To draw your first brush, begin in any one of the three 2D windows.**

   In this example, I start in the top window, which is located in the top-right of the editor.

   If you don't remember which window is which, drag your mouse over the windows and keep your eye on the title bar of the editor. When you drag the mouse over each window, the window's name appears in the title bar. Or, drag your mouse over the upper-left corner of each viewport to see the name of the window appear in that corner.
3. **Click anywhere near the middle of this window and then drag the mouse.**

   When you drag the mouse, a square with a dashed, yellow outline appears. On the top and left sides of the square are numbers, as shown in Figure 5-6. The numbers represent the size of the selection area, which isn't important just yet.
4. **After you create a medium-sized square on the grid, release the mouse button.**

   You aren't concerned with the size of the brush right now because you'll resize it shortly. When you do release the mouse button, the square marking the selection area turns from yellow to white.
5. **Press Enter to create the brush.**

   With your selections are complete, pressing Enter creates the brush within these boundaries. The result is a red square with a white dot in each corner and a small red x in the middle.

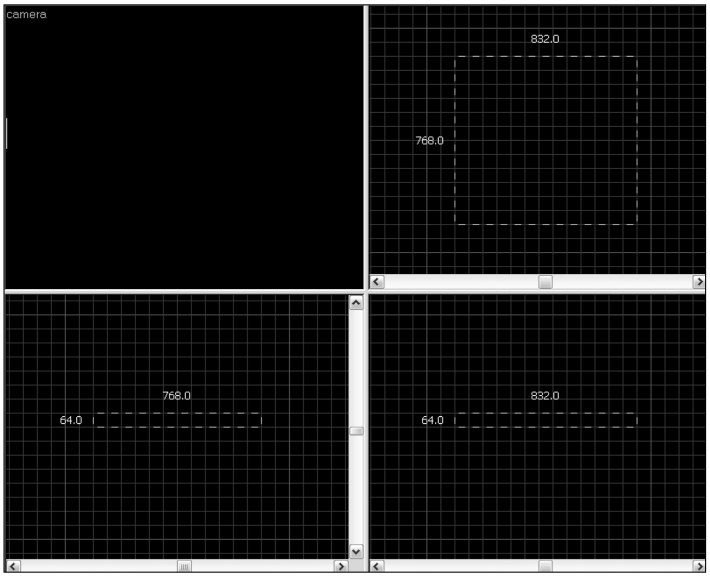


**Figure 5-6:** After you draw your brush, the dimensions are displayed.

There you have it. You just started your first map and created your first brush. You can't yet play it in the game, and it doesn't look too fancy, but it's the start of something that might be played by thousands of gamers when you're done.

## Maneuvering in the Viewports

Before you get back to resizing and building upon your first brush, you should get more familiar with the four windows in the editor. You'll work in these windows most of your mapping life, so understanding how they work is important.

### Zooming and moving the view

Although each of your four viewports can be customized to display anything that you want, I assume that you kept them in their default orientation that I review in Chapter 4.

Moving around within the 2D viewports is fairly simple. There aren't a lot of different movements you can make.

To move left, right, up, and down in the 2D viewport, you have two options. You can slide the scroll bar on the sides of the window in the direction you want to move. Or, you can place your mouse over the viewport and then press your arrow keys to move in the appropriate direction. I usually use the arrow keys because they're more precise.

REMEMBER

Make sure that your mouse cursor is over the top of the window in which you want to move. Otherwise, the window that moves might not be the one that you intended.

Using the arrow keys in the camera viewport actually results in a different kind of movement. Pressing the arrow keys rotates your camera just like when you rotate your head. If you want to shift in the direction of the arrow keys, you must first press Shift. For example, pressing Shift+↑ moves you up instead of causing you to look up.

Perhaps a more familiar way to move around in the camera viewport is via the WASD keys. These are the keys often used to move around in the game, and they've been adapted to work the same within the editor. *W* moves you forward, *A* left, *S* back, and *D* right.

To zoom in and out — or move forward and back — in any of the viewports, use your mouse wheel. Scrolling the wheel is the easiest way to zoom about.

Moving around in each of the viewports might seem tedious. However, a few additional shortcuts make life easier. If you're working in a large map and you need to adjust a brush that you located in one viewport, you can center the view around your selection in the other three viewports by pressing Ctrl+E. You have to locate the brush that you want to edit in only one viewport, and then you can automatically put it in view within all the other viewports.

As for the camera viewport, although a few different view options are available, I recommend changing the view option to 3D Textured Polygon. The camera viewport will become an important means of selecting different objects in your map, and the ability to see exactly what you're selecting is very important.

To make this change, click anywhere in the camera viewport to make sure that it's the selected viewport that you want changed. Choose View➪3D Textured Polygon. When you do this, your camera viewport might appear to go blank because your camera is currently inside the only brush in your map. Place your cursor over the camera viewport and then scroll the mouse wheel downward until you see your brush appear, as shown in Figure 5-7.

## Counting in the power of two

The numbers in the mapping world revolve around the power of two. All textures for a game are sized with numbers, such as 8, 64, 128, 256, 512, or some combination thereof. When images are scaled from one size to another, they look best when they're scaled by powers of two.

Textures in many games are created with a special format that already contains the scaling information for each texture size. So, a texture that's 512 × 512 might already contain the information for the same texture that's 256 × 256, 128 × 128, and so on.

A similar sizing pattern is found in the way the 2D window is laid out. The default setup places the grid lines 64 units apart. The darker grid lines (if you can see them) are 1024 units apart.

Next, note that each unit in the editor is equal to 1 pixel in a texture. Placing a 64 × 64 texture on a brush face that is 64 units square results in an optimally sized texture-to-brush ratio.

The grid lines in the editor's 2D windows can be adjusted. Pressing [ and ] decreases and increases the grid size. What makes this important is that with grid snap turned on, you can't make brushes or selections equal to any other size than a multiplication of that grid size. If you want to make a wall 8 units thick, you must first decrease the grid size to 8 units or less.

TIP

You can turn off the grid snap option by choosing Map➪Snap to Grid. However, this isn't recommended. It is extremely important that your brushes line up exactly in the editor, or errors result later when you try to compile your map into a playable level.

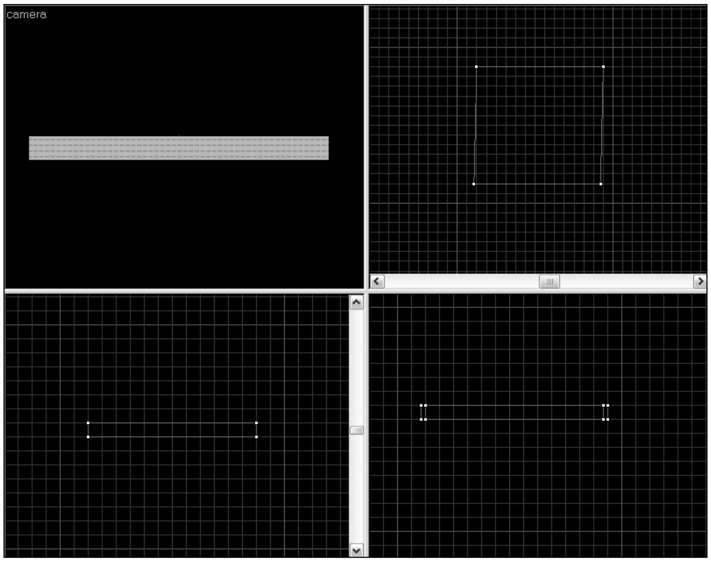


**Figure 5-7:** Change your camera viewport so you can see and select objects in your map.

## Resizing and Moving Brushes

Getting back to your map, you need to finish sizing your medium-size brush that you created in the section, "Drawing the First Brush," earlier in this chapter. You want to size it equal to that of the entire room you're making, which will be 512 units in width, 512 units in depth, and 128 units in height. I choose 512 units square because this creates a room large enough to experiment in without making it too large. A room with a height of 128 units makes the player feel comfortable. You could create a room of nearly any size when constructing your map, but for practice purposes here, you should use the dimensions listed.

If you read the earlier section, "Zooming and moving the view," you already know how to move around in your 2D window, but now you need to understand how to resize and move your brushes in your map.

To resize a brush, you must first select it. Select the Selection tool icon on the left as shown here or press Shift+S. This tool allows you to select and manipulate the objects in your map.

To select your brush, just click it in the camera viewport. This selects your entire brush.

You can also select your brush from any of the 2D viewports by clicking the center x or one of the outer edges of the brush; however, this method can be inaccurate and isn't recommended. If you have a large map with many brushes, it's not as easy to select the precise brush from the 2D window because you might accidentally select the wrong one. If you select your brushes from the camera viewport, making a selection mistake is much more difficult. However, in some cases, you must select your brush from the 2D viewport. One example is when you want to rotate or skew your brush. In Chapter 13, a light fixture is shaped from the 2D viewport.

With your brush selected, you see a group of small, white squares around your brush in the 2D viewports (see Figure 5-7). These are handles that you can click and drag to resize your brush. If you click a corner handle, it resizes both sides of that brush; the other handles affect only one side.

If you need to move the brush, simply click and drag anywhere within the brush from within any of the 2D viewports. You can move your brush around and place it anywhere although right now, your brush probably shouldn't need to be moved.

When resizing your brush, your goal is to create a brush 512 units in width, 512 units in depth, and 128 units in height. First, resize the width and depth. From within the top 2D viewport, stretch each of the four sides of the brush until the numbers around them read 512.0, as shown in Figure 5-7. Remember that for better control of your brush, you can zoom in and out of the window's view as well as move the grid.

Next, resize the height of the brush from either of the two other 2D viewports. Click and drag either the top or bottom handle until the brush height is equal to 128.0, as shown in Figure 5-7.

## Taking Cues

Often people overlook the status bar at the bottom of the editor. It's not prominent within the editor, but take a look at the bottom of the editor now. There you find information about your mapping environment that can be of great value.

Here is a list of what you can discover from this small but resourceful status bar. Each appears from left to right in the status bar:

- **Help Reminder:** The bar starts off reminding you that whenever you need help with the editor, just press F1. After you press F1, the Valve Developer Community page opens in your Web browser where a lot of useful information is found.

- **Selection Information:** Here you see information about what you selected. For example, if you selected a brush, this displays the type of brush and the number of sides. However, if you selected an entity, this displays the name information of the entity.
- **Coordinates:** The numbers displayed here are the current coordinates of your mouse from within one of the 2D viewports. If you're in the top X, Y viewports, the numbers displayed here are the X and Y map coordinates of your mouse in that window.
- **Selection Size:** If you click a brush or create a new selection area in one of the 2D viewports, this information tells you the width, length, and height of your selection as well as its center point in the form of X, Y, and Z coordinates.
- **Zoom Size:** The level of zoom you selected for your current 2D viewport is listed here.
- **Grid Information:** Here you find the current grid information for your 2D viewports. However, note that the grid information is for all 2D viewports because changing the grid value for one affects them all. The information tells you whether grid snap is on or off as well as the size of the grid.

## Hollowing Out the Room

When your brush is properly sized, you need to hollow it out to make room for your player. Right now, it's a solid block. Your goal is to create an empty room with four walls, a floor, and a ceiling.

Make sure your brush is selected first. Then choose Tools➪Make Hollow or press Ctrl+H. A dialog box pops up, asking you, "How thick do you want the wall?" as shown in Figure 5-8.

*Wall thickness* refers to the thickness of the walls after this brush has been hollowed out. If you enter **8**, the hollowed brush's walls will be 8 units thick. However, you also have the option of entering a negative number, which keeps your hollowed area equal to the current size of your brush. The walls would then be created outside of the currently solid brush.

In this case, enter a value of 8. This is a good wall thickness for the game because it makes the walls appear similar in size to the walls in most homes. Then press OK to hollow the brush and continue.

You now have something that looks more like a room and less like a solid block. (See Figure 5-9.) The brush that was once solid now looks more like a room made of six brushes: one for each side, one for the top, and one for the bottom.

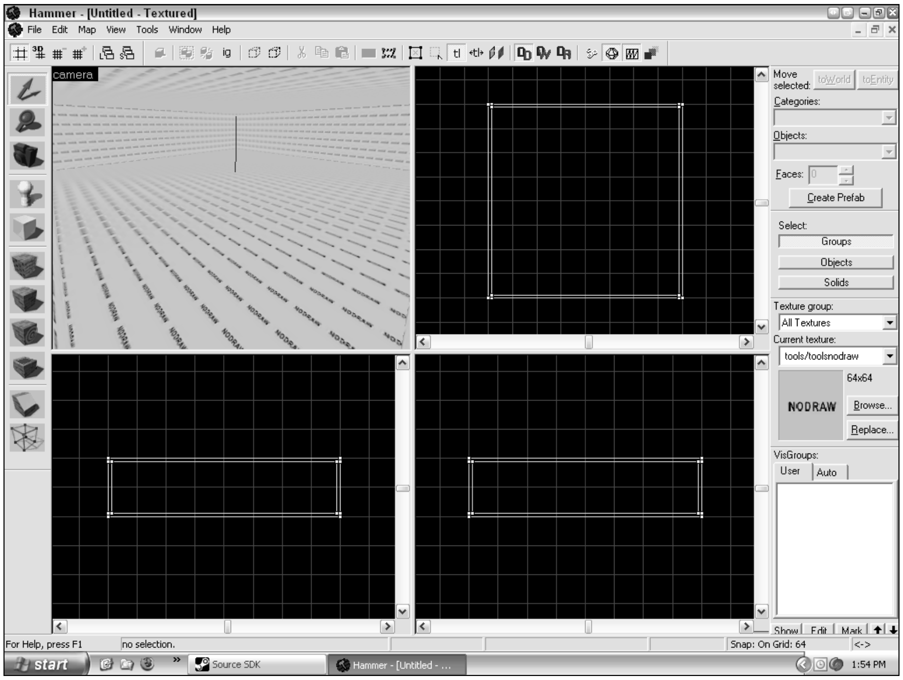


**Figure 5-8:** Enter the resulting brush wall thickness for the hollow function.

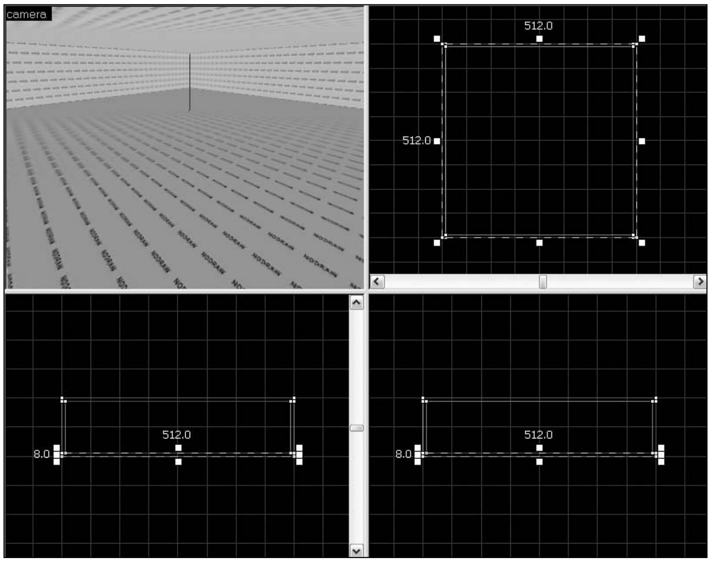


**Figure 5-9:** Hollowing a solid brush results in a structure of multiple brushes.

## Breaking Things Apart

Try selecting just one of the walls in your new room. You'll find that you can't do it without selecting all six walls because all the brushes resulting from the hollowing process are grouped together. In some cases, you might find this grouping useful; however, I find it annoying because I then have to take additional steps later to make changes to individual walls.

To ungroup the brushes that make up your room, follow these steps:

1. **From within the camera viewport, click anywhere in your new room.**

   This selects all size walls, giving them a reddish hue in the camera viewport and a yellow outline in the 2D viewports.

2. **With the group selected, choose Tools➪Ungroup or press Ctrl+U.**

   Seemingly nothing happens, but your brushes did ungroup.

3. **Within the camera viewport, select any wall in your room.**

   Only one wall is selected instead of all the walls in the group.

## Leaking Is Not an Option

Making sure that your map has a good seal is crucial. You'll be adding entities that create light or are interpreted as objects that can create light. If you allow this light to escape into the void outside your map, you create a *leak.*

The game engine doesn't like leaks because they present too much information for it to process. When you take your map and compile the information into a playable level, the process must also calculate how light acts and reacts in the environment. Some textures reflect light like a mirror, and that reflected light can in turn reflect off another surface. This bouncing effect is perhaps the most intensive process for your computer to calculate during the compile process. If you allow the light to go beyond the confines of your playable area, your computer foresees an endless amount of calculating for the light that is bouncing into the void, and the computer reports that a leak has occurred.

If you accidentally moved any of the walls in your room, go back and check for potential leaks. Make sure that no visible gaps fall between your brushes. Also look in the corners for holes; if any exist, adjust the brushes to close them before you continue.

Other possible causes for leaks are:

- Placing an entity outside of your mapping area. All entities should be within the confines of your map.
- Texturing a structural brush or surface that should be keeping the seal on your map from the outer void with a texture that is transparent or non-solid. Such a texture would be a wire fence where you can see through the holes in the fence.
- Not having a good seal on your area portals between leafs. This is part of optimization techniques discussed in Chapter 10. Area portals are used to seal off room from each other.

That's it for the structure of your room, and now is a good time to save your work before you continue.

## Saving Your Map

Finally, it's time to save your map. Save your map often after each important step that you perform in the editor. It would be horrible to lose all your hard work if for some unexpected reason the editor crashed. Sure, the editor isn't suppose to crash, but just as Murphy's Law states, "Whatever can go wrong, will go wrong."

To save the map, you have a couple of options:

- Choose File➪Save or choose File➪Save➪Save As.
- Use the keyboard shortcut, Ctrl+S.

When naming your map, make sure to use only alphanumeric characters and no spaces. Any other characters result in errors when you compile and play your map. Also, since this is a deathmatch game type, it is common practice to start the name of your file with *dm_*. This makes your map easily recognized as a deathmatch map. For example, instead of naming your map `my room.vmf`, use `dm_my_room.vmf`. This saves you a headache later.

Because you're working in Chapter 5 of this book, save it under a name that makes it easy to reference later. For example, name it `dm_chapter5.vmf`.

Chapter 6

# Decorating the Scene

***In This Chapter***

- ▶ Grabbing new textures
- ▶ Covering the Nodraw texture with fresh paint
- ▶ Applying textures to walls, floors, and ceilings

The stage is set, but what about the scene? If you've followed the book to this point, the walls in your `dm_chapter5.vmf` file are just covered in the Nodraw texture. This texture is great for keeping the processing power required by the game to a minimum but is quite unsuitable for your game play. Your player can't see this texture in the game, so you should dress things up with some textures.

*Textures* are really just pictures painted on the sides of the brushes in your map. (A side of a brush is a *face.* Texturing the side of a brush is *painting a brush face.*) These pictures are often designed to look like concrete, brick, metal, wood, or other materials you commonly find in the real world. Most of the time, these pictures are created in small squares that can be stacked on top and next to each other endlessly and saved in an image format that the game can recognize. These are called *tileable images* because the image is applied like tiles on a floor.

Your objective is to place some of the textures onto the brush faces of your map. You can put a concrete texture on the floor, a metal texture on the walls, and a decorative tile on the ceiling. All the textures that you need are available in the editor; they're based on the levels already created for the game. If you see a texture in the game, you can use it in the editor for your map as well.

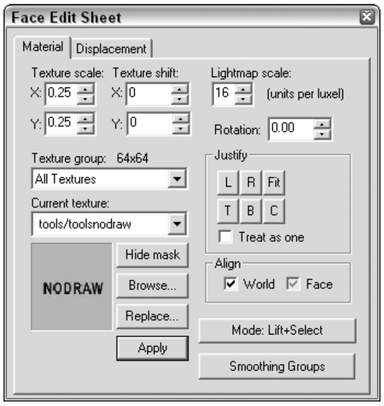
## Selecting Faces on the Wall

Before you select the new texture you want to apply, you must select the brush face of one of your walls. This distinction (the brush *face,* not the brush) is important because you don't want to apply the texture to the entire brush on each of the walls. This defeats the purpose of using the Nodraw texture when constructing the room.

First, adjust your view within the camera viewport so that you can see the inside of your room. From here, you can select the brush faces that require a texture.

Click the Texture Application icon (as shown in the margin) on the left side of the editor. The shortcut for this tool is Shift+A.

After you activate the Texture Application tool, the Face Edit Sheet window comes up, as shown in Figure 6-1.



Texture scale: X: 0.25 Y: 0.25
Texture shift: X: 0 Y: 0
Lightmap scale: 16 (units per luxel)
Rotation: 0.00
Texture group: 64x64
All Textures
Current texture: tools/toolsnodraw
☐ Treat as one
☑ World ☑ Face

**Figure 6-1:** Press Shift+A to access the Texture Application window's feature set.

## Exploring the Face Edit Sheet Window

In the Face Edit Sheet window, you can select additional textures and adjust those that exist in your map. However, perhaps the most important options that become available with this open window are the texture shortcuts for applying, copying, and working with textures in your map.

Move this Face Edit Sheet window to the side so that you can better see the camera viewport. Then let me show you the different options available in this window and what each does for you and your textures.

In the first tab — Material — you have all your texture application options available:

- **Texture Scale, X and Y:** This option allows you to scale the size of the texture applied to your brush. Because a brush face has only two directions, only two axes are represented for scaling — the X and Y axes. Although you can adjust the values by entering them manually or pressing the respective up and down buttons for each axis, you can also place your mouse in the value field and then adjust it by scrolling your mouse wheel. Enter a negative number to invert the texture.

- **Texture Shift, X and Y:** This option allows you to move the texture across the brush face. Because a brush face has only two directions, only two axes are represented for shifting — the X and Y axes. Adjusting the values for shift is done the same way as for the texture scale.
- **Texture Group:** Because no texture groups are available, this selection is limited to All Textures. However, textures that are imported from the original *Half-Life* game WAD files can be referenced here individually as a group.
- **Current Texture:** This is quite simply the texture you have selected for application to a brush face.
- **Hide Mask:** If the red hue that you see on selected brush faces is getting in your way, select Hide Mask to hide this reddish coloring. You can't see which brush face you selected, but you do have an unobstructed view of the texture as it is applied and adjusted on the brush face.
- **Browse:** Click Browse to open the Textures window, which provides you with more texture find and selection options.
- **Replace:** This button opens another dialog box that allows you to find and replace textures throughout your map.
- **Apply:** With a brush face selected, click Apply to paste the selected texture on the selected brush face.
- **Lightmap Scale (units per luxel):** Adjust this value to affect the level of lighting detail on the selected brush face. A lower number increases the quality of light displaying more defined shadows because it applies more lighting detail to fewer units. However, a lower number also requires more processing power from your computer during game play. I recommend that you adjust this value only when needed.
- **Rotation:** The value entered here is the degree of rotation for the texture on the brush face. Positive and negative numbers can be used.
- **Justify:** Within this option area, you're presented with a number of buttons that affect the position and size of the image. The alignment buttons are L (left), R (right), T (top), B (bottom), and C (center). The Fit button is unique because using it fits the single texture to the brush face; therefore, scaling it in size equal to that of the brush face.

  The other option here is the Treat as One check box. When you enable this option and you select multiple brush faces, all the faces are adjusted as if they were one, large brush face. For instance, clicking the Fit button fits the texture to the entire selection of brush faces as if they were one brush face rather than fitting them to each individual brush face.
- **Align:** The align options for textures refer to how the origin of the texture is calculated. A World alignment considers the entire mapping world as a basis for the origin of your texture. A Face alignment considers the selected brush face as being in a world of its own. This matters when trying to line up the position of applied textures.

- **Mode:** This option might appear to be a single button but is actually a drop-down list of options. The modes listed here affect what occurs when you use your mouse in the camera viewport to manipulate textures on brush faces. These options are
  - *Lift + Select:* Clicking a brush face selects the brush face and then sets its texture as your current texture in the Face Edit Sheet.
  - *Lift:* Clicking a brush face selects only its texture (and not the brush face) as your current texture.
  - *Select:* Clicking a brush face selects only that brush face.
  - *Apply (Texture Only):* Clicking a brush face applies the current texture.
  - *Apply (Texture + Values):* Clicking a brush face applies the current texture and any other values entered on the Face Edit Sheet.
  - *Align to View:* Clicking a brush face applies the current texture using your camera as the point of origin. This means that if you're looking at the brush face on an angle from within the Camera viewport, your texture is applied on that same angle.
- **Smoothing Groups:** Selecting this option allows you to assign multiple brush faces to a numbered group for lighting. Then, when compiled and played in the game, the game lights each group of brush faces as if they were a single, continuous brush face. This is helpful if the lighting looks like it is being broken up along a wall made of multiple brush faces.

The second tab in the Face Edit Sheet is Displacement. Its primary use is to modify brush faces, such as those made to look like terrain. With the options listed here, you can raise spots of a brush to form hills, lower spots for valleys and perform a number of other functions to make your terrain look like the bumpy ground you see outside of your window.

## Putting on Some Paint: Applying Textures

Textures are the paint and wallpaper of the virtual world. They are images that you can apply to a brush face making it appear as brick, wood, water or anything else you have in mind.

The steps below will show you how to dress up your empty room. With the textures you are about to apply, your empty, yellow room will start to look like a game level.

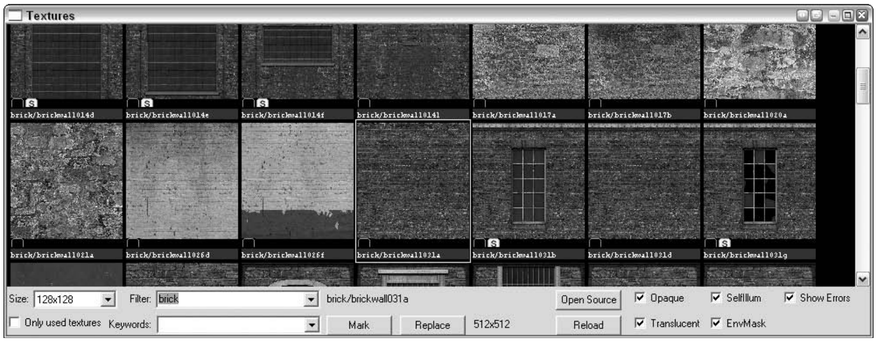
## Adding texture to a wall

When you're ready to apply a new texture to a wall, follow these steps:

1. **Select one of the inner walls of your hollowed room by clicking it in the camera viewport.**
2. **In the Face Edit Sheet, click Browse to open the Textures window.**

   Here you want to locate a good brick texture to apply to the walls of your room. However, when you open the Textures window, you might see only the Nodraw texture. This happens because nodraw is still in your Filter box. Select nodraw and then delete it by pressing Delete. A second or two later, all the textures will show up again.

   You could look through all the textures to find just the right one. But to make things easier, back in the Filter box, you can enter the word **brick** to display only those textures with *brick* in the title. In the third row of textures, you find a good texture to start with: titled **brick/brick-wall031a** (as shown in Figure 6-2). If you have trouble finding it, type the name into the Filter box.
3. **Double-click the brick/brickwall031a texture to simultaneously select it and close the Textures window.**



Size: 128x128 Filter: brick brick/brickwall031a 512x512
[✓] Opaque [✓] SelfIllum [✓] Show Errors
[✓] Translucent [✓] EnvMask
[ ] Only used textures Keywords:

**Figure 6-2:** Filter your textures to find what you're looking for easily.

4. **With your texture selected, right-click one of the four walls in your room.**

   This applies the texture you select.

   If you accidentally left-click the wall, you replace your selected brick texture with the Nodraw texture. You have to go back and reselect the brick texture for application.

5. **Continue to place the brick texture on the remaining walls in the room until all four are covered.**

   You have to move your camera around in the camera viewport to do this. Just use the WASD (*W* moves you forward, *A* left, *S* back, and *D* right) and the arrow keys to get around, as I explain in Chapter 5. The result looks like Figure 6-3.

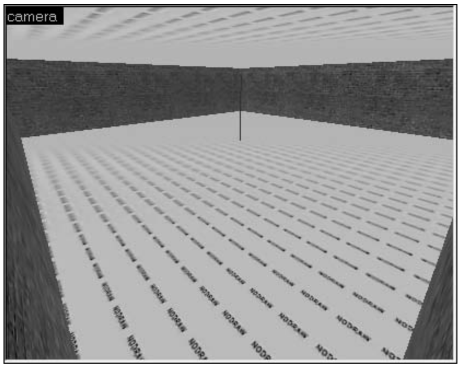


**Figure 6-3:** Apply the brick texture to the four walls in your room.

## Adding texture to the floor

After you apply a texture to the wall, you can move on to other brush faces. Click Browse within the Face Edit Sheet window. It's time to look for a texture for your floor. The following steps show you how:

1. **In the Textures window, replace *brick* in the Filter box with *concrete*.**

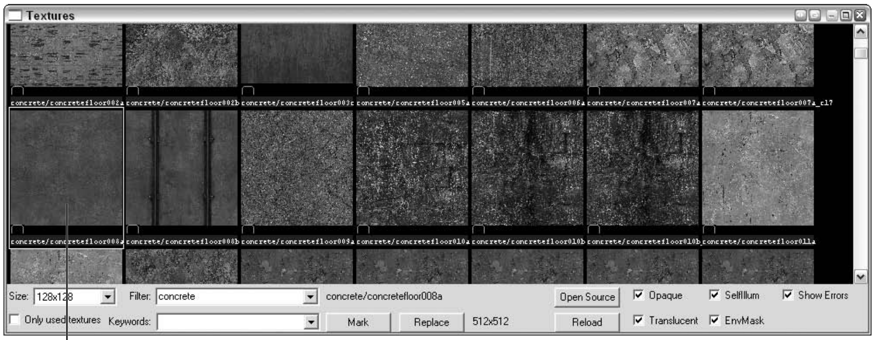
   A room with brick walls looks best with a concrete floor.

2. **Scroll down using the scroll bar on the right to the third row and then select the first texture listed, titled concrete/concretefloor008a (as shown in Figure 6-4).**

   If you have trouble locating it, type the full name of the texture into the Filter box, and you'll find it. Double-click the texture to select it and then close the Textures window.

3. **When your concrete texture is selected as the current texture, right-click the floor in your room to apply it.**

   The result is a nicely textured floor, as shown in Figure 6-5.


Textures
concrete/concretefloor002a
concrete/concretefloor002b
concrete/concretefloor003c
concrete/concretefloor005a
concrete/concretefloor006a
concrete/concretefloor007a
concrete/concretefloor007a_c17
concrete/concretefloor008a
concrete/concretefloor008b
concrete/concretefloor009a
concrete/concretefloor010a
concrete/concretefloor010b
concrete/concretefloor010b_
concrete/concretefloor011a
Open Source
Mark
Replace
Reload
concretefloor008a

Size: 128x128 Filter: concrete concrete/concretefloor008a ✓ Opaque ✓ SelfIllum ✓ Show Errors
☐ Only used textures Keywords: 512x512 ✓ Translucent ✓ EnvMask

**Figure 6-4:** Select a concrete texture for the floor of your room.

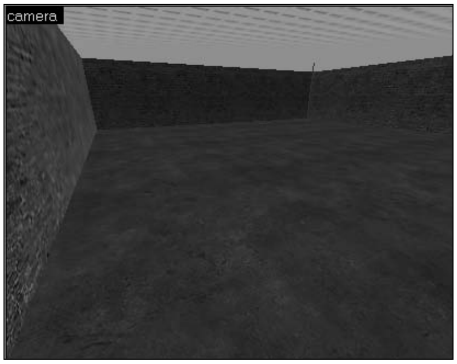


**Figure 6-5:** Right-click to apply the concrete texture to your floor.

## Adding texture to the ceiling

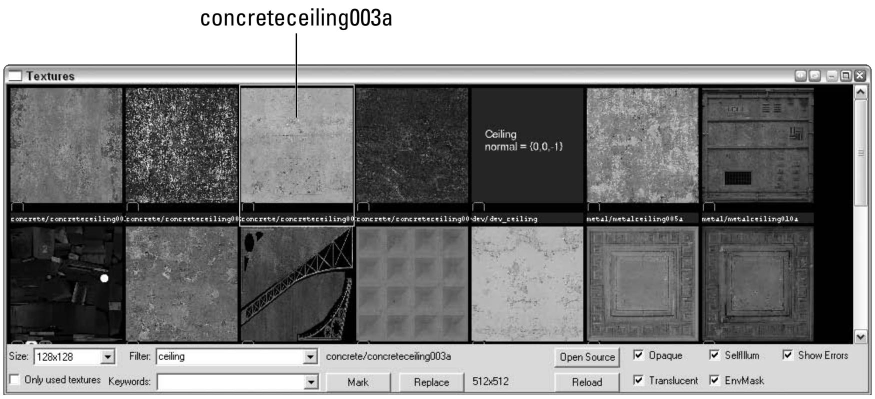
After the floor is done, you're left with the ceiling to texture. The following steps show you how:

1. **Click Browse to open the Textures window and then replace whatever is in the Filter box with *ceiling*.**

   This lists all textures with *ceiling* in the title.

2. **To fit the theme of the room that you establish, select the texture concrete/concreteceiling003a.**

   Look for the third texture in the top row, as in Figure 6-6. Double-click this texture to select it and close the Textures window.



Size: 128x128 Filter: ceiling concrete/concreteceiling003a [✓ Opaque] [✓ SelfIllum] [✓ Show Errors]
Only used textures Keywords: 512x512 [✓ Translucent] [✓ EnvMask]

**Figure 6-6:** Select the new ceiling texture for the ceiling of your room.

3. **Right-click the ceiling in your room to apply the newly selected texture.**

   Your room is complete now with textures. (See Figure 6-7.)

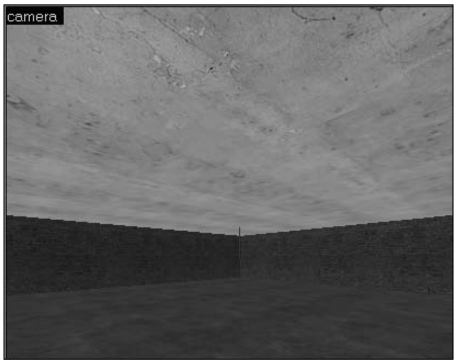


**Figure 6-7:** Right-click to apply the ceiling texture.

4. **Close the Face Edit Sheet window after you're done applying your texture. Then choose File➪Save As and save your map as `dm_chapter6.vmf`.**

   You don't want to lose the work you've done to this point, and I recommend saving your map under progressing names as you go along, such as `dm_mymap1.vmf`, `dm_mymap2.vmf`, and so on. Naming your files in this way can give you some security so that a map error halfway through its development doesn't ruin the rest of your work. If an error occurs that you can't fix, you can always go back to a previous version of the file.

Chapter 7

# Adding Lights and a Player

***In This Chapter***

▶ Adding light to the room

▶ Coloring a light from the Object Properties window

▶ Creating a player spawn point

▶ Adjusting spawn point properties

After you create a room for your player as was done in Chapter 6, you need to add two important things: light and a player spawn point.

You need light to see. Without the light, everything is black. Sometimes you want the player to rely on his or her flashlight or other resources to see where he or she is headed, but generally you want to provide some source of light.

Next you need to define where your player *spawns* (starts) in the level. Without the spawn point, the game doesn't know where to put the player when the game starts. Although, later you'll want to add multiple spawn points for many players, for now, you need just one so you can spawn into your level for testing.

## Lighting the Way

Adding light to a map is quite important and really simple. Without the light, the player can't see in the game. Although it's true that sometimes you want a dark area in the level — it can provide a place to hide something like a *pickup* (such as a weapon, armor or health pack), an enemy, or the player — you probably don't want the entire map completely dark. If the player can't see where he or she is going, he or she becomes frustrated. So, you should provide at least one dim light within your level.

To add light to your level, you must add an entity to your map. *Entities* are active or interactive elements of a game such as a light, a door, or a pickup. Here's how to add one to your map:

1. **Activate the Entity Tool by clicking its icon on the left or by pressing Shift+E.**

   The Entity Tool icon looks like a lightbulb. Selecting this tool allows you to place entities in your map.

2. **From within the New Objects group, choose Light from the Objects drop-down list.**

   The New Objects group is on the right side of the editor window, as shown in Figure 7-1. The Categories box shows the default selection, Entities, and in the Objects list are the entities available for you to place in your map. The entity name that you place to create light is called *light.*

3. **Click any 2D viewport in the location where you want the entity placed.**

   You don't have to place the 2D viewport in an exact location. You can move it later. However, try and place the light close to where you want it. For now, place the light in the upper-right corner of the room near the ceiling, also shown in Figure 7-1.

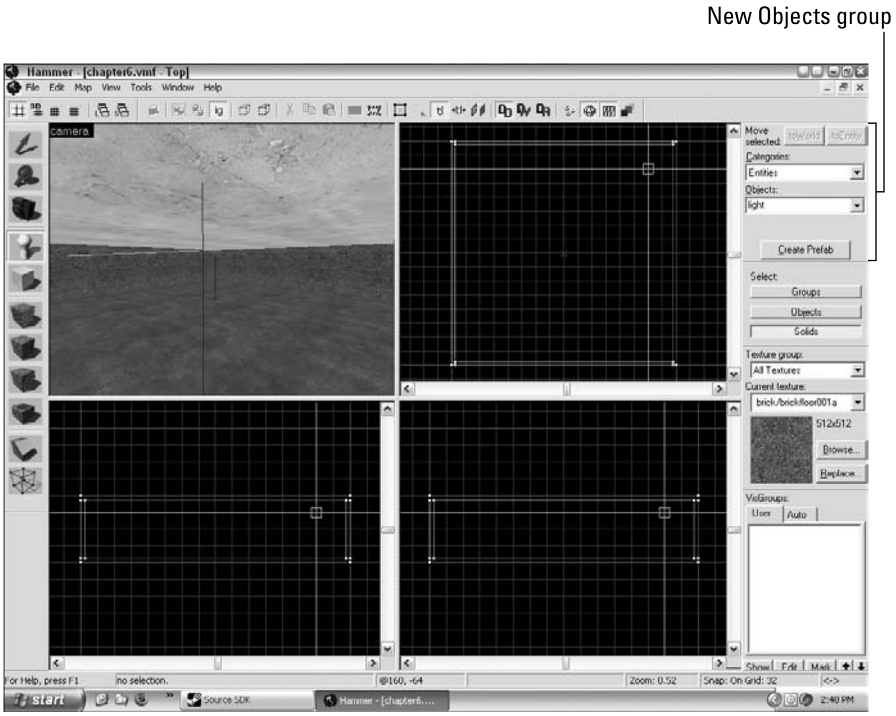


**Figure 7-1:** The *light* entity creates a point of light in your map.

You might find that you can't place the light exactly where you want it because the entity placement keeps snapping to the grid intersections. Press [ to decrease the grid spread. This allows you to more accurately place the light.

4. **Press Enter to insert the light entity into this position.**

   After you press enter, you see a small red square in the 2D viewports. In the camera viewport, the light looks like a lightbulb, as shown in Figure 7-2.

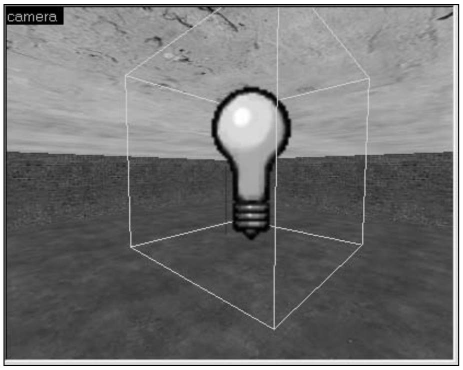


**Figure 7-2:** The light looks like a lightbulb in the camera viewport.

If you read to this point, the light is placed and ready to light your map. Don't worry about this lightbulb image showing in your map and being seen by players in the game. The light entity is a *point entity* and therefore, isn't a direct interactive element in the game, but rather an active element that produces the action of providing light from a single point. The player can't see point entities directly.

## Positioning the entity

Position the light entity in a way that makes sense to the player. It wouldn't make a lot of sense if the light came from the center of the floor or a bottom corner of the room (that is, unless you put something else there that looks like it should create this light). Instead, the player is going to expect the light to come from the ceiling as if from a light fixture.

To move your new entity in the map, switch from the Entity Tool to the Selection Tool. The Entity Tool allows you only to place entities in the map and not to move them around.

To activate the Selection Tool, press Ctrl+S.

From within any 2D viewport, click and drag the light entity toward a corner in the room and place it similarly to the entity shown in Figure 7-3. Also, you can move the light toward the ceiling the same way.

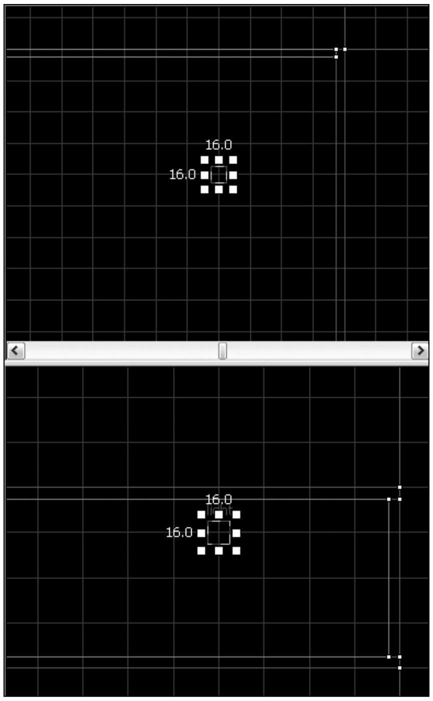


**Figure 7-3:** Move the light to the corner of the room after selecting the Selection Tool.

For now, leave this single light in the corner so you can see the difference of light and dark within the game when you test the compiled map.

## Adding a bit of color

The light entity in the camera viewport is white because that is the default light color in any map. However, sometimes you might want to choose a different color depending on the environment you're trying to simulate. A change of color can make a huge difference in the look of your level.

As an example, try changing the color of the light to red so that you can see the difference it provides in the game. Here's how:

1. **With the light entity selected, press Alt+Enter to open the Object Properties dialog box.**

   The Object Properties dialog box allows you to adjust any property of the entity you select (see Figure 7-4). Right now, you're interested in the Brightness property, which also determines the color of the light.

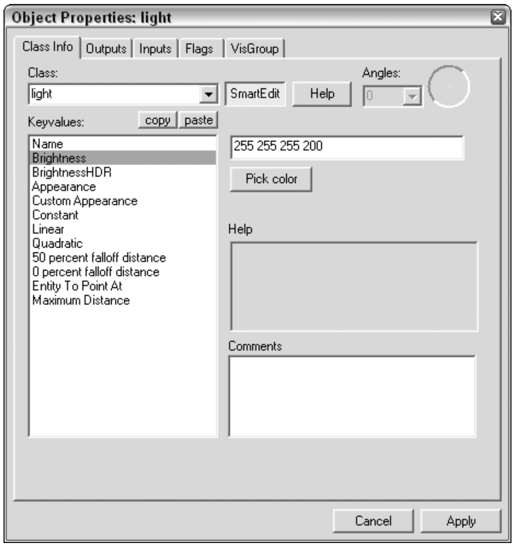


**Figure 7-4:** The Color Picker window allows you to select or type in a color value for your light.

2. **Choose Brightness from the list of Keyvalues.**

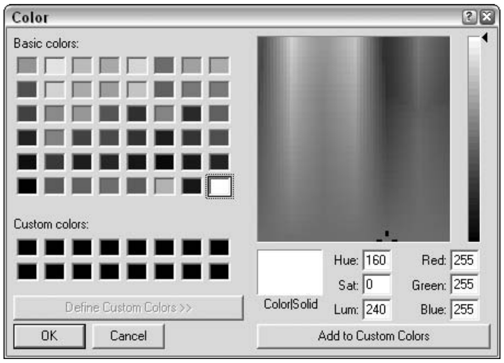
   Brightness is the second option, right under Name, in the list. Making this selection also changes some of the options on the right.

3. **Click Pick Color to select a new color for your light entity.**

   Pick Color opens a new dialog box with your color selection options, as shown in Figure 7-5. You have the option of selecting a color or typing in the specific color values to obtain what you're looking for.

4. **Select the color red and click OK.**

   Selecting your color automatically enters the appropriate values into the properties value box located just above the Pick Color button. Yours reads 255 0 0 200. The first three digits are the RGB color value that you select, and the last digit is the brightness level, which is defaulted to 200.



Hue: 160 Sat: 0 Lum: 240 Red: 255 Green: 255 Blue: 255

**Figure 7-5:** The color the light emits within the game is represented by the color of the entity in the camera viewport.

5. **Click the Apply button and then the Cancel button to apply the property changes and close the window.**

   The once-white light entity in the camera viewport turns red so that you can see from within the editor what color the light is, as shown in Figure 7-6.

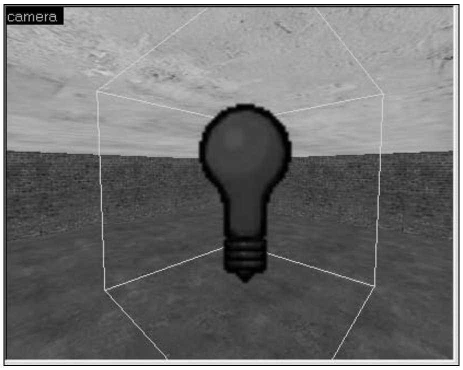


**Figure 7-5:** The color the light emits within the game is represented by the color of the entity in the camera viewport.

That's enough with the light in this map. It's time to move on to creating the player spawn point.

## Adding a Place to Start

The *player spawn point* is the location where the player starts within the game. A player spawn point is also an entity. You can define the location as well as the direction the player is facing when he or she spawns. Here's how:

1. **Press Shift+E to activate the Entity Tool.**

   The Entity Tool allows you to place new entities.

2. **Choose the info_player_deathmatch entity from the Objects list in the Objects group on the right.**

   Before placing your entity, you need to choose the entity that you want to place from the Object group. In this case, this entity is theinfo_player_ deathmatch entity. This is the entity that is used for player spawning by the *Half-Life 2: Deathmatch* game type.

3. **Click in any 2D viewport to place your new entity.**

   In the corner opposite the light, click and place your new entity within any 2D viewport.

4. **Press Enter to insert your new entity.**

   After you insert the info_player_deathmatch entity, you see what looks like a model similar to Gordon from the game, as shown in Figure 7-7. It's kind of funny to see, but this is a good representation of a player spawn point.

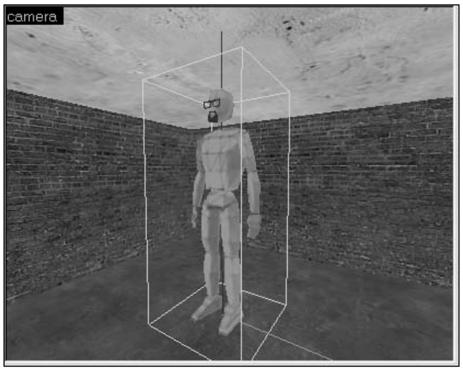


**Figure 7-7:** The info_player_ deathmatch entity looks like Gordon in the editor.

TIP

When you zoom in closer to view the entity in any 2D viewport, you also notice that the entity name is displayed. This is another helpful tool when working on your maps.

## Positioning and Providing Some Direction

Looking at the placement of the entity, it might not be where you want it in the map. In Figure 7-8, you can see the entity isn't standing on the ground, and it's also facing to the side. These things need corrected.

Switch from the Entity Tool to the Selection Tool by pressing Shift+S. With this tool selected, you can move the player entity so that it is standing on the ground. In my case, I had to lower the grid size to 8 units before I could get the entity placed properly on the floor instead of inside it or floating above of it.

If the entity is positioned in the room, it's a good idea to adjust the direction it's facing. A better option is if the player was facing the center of the room than when they spawned one of the walls.

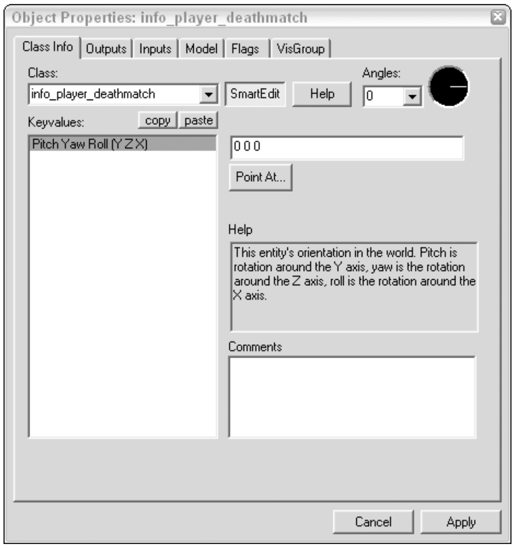
To adjust the angle at which the entity is facing, you have two options:

- Adjust the entity's angle from the Object Properties dialog box.
- Adjust the rotation of the entity from within the 2D viewport.

## Adjusting the rotation from the Object Properties window

The first way to adjust the angle the entity is facing is from the Object Properties window. Here's how:

1. **With the entity selected, press Alt+Enter to open the Object Properties dialog box (see Figure 7-8).**



Class: info_player_deathmatch

Angles: 0

Keyvalues: Pitch Yaw Roll (Y Z X)

0 0 0

Help

This entity's orientation in the world. Pitch is rotation around the Y axis, yaw is the rotation around the Z axis, roll is the rotation around the X axis.

Comments

**Figure 7-8:** The Object Properties dialog box for this entity allows you to face the player in a specific direction.

This dialog box, for the info_player_deathmatch entity, has only one property adjustment, which is for the Pitch Yaw Roll of the player.

In the upper-right corner of this dialog box, you see a black circle. Within this circle you will either see a white line or a white dot. This circle works a lot like a compass. The dot represents the center of the black circle while the line represents the direction your object is facing as from the center of the circle.

2. **Click and drag within the circle and adjust the radius line so it is pointing in the direction that you want the entity to face, as I did in Figure 7-8.**

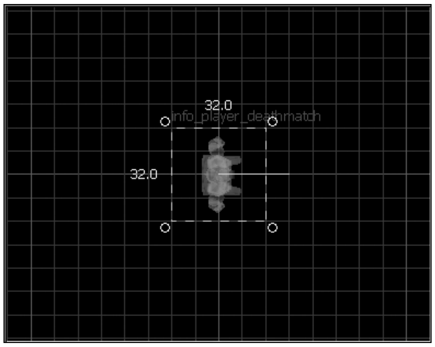


**Figure 7-9:** Two clicks of your object in the 2D viewport activate the Rotation option.

This black circle automatically adjusts the Yaw of the entity. You can see the degree of rotation represented as 0 45 0 in the text box below the circle. The three digits represent the Pitch, Yaw, and Roll respectively.

3. **After the angle is set to your liking, click Apply to apply the change and then click Cancel to close the window.**

## Adjusting the rotation from within the 2D viewport

Perhaps the easiest way to adjust the direction your entity is facing is from right within the 2D viewport. Clicking on the entity more than once offers you additional manipulation options.

- The first click selects the object and size the object in your map.
- The second click offers you rotation adjustment.
- The third click lets you skew the object.
- The fourth click brings you back to selecting and sizing the object.

Now, change the direction of your player spawn point so that it is facing the center of the room.

1. **If you selected your player spawn point already, click the entity from within the Top (X/Y) viewport.**

   You must rotate your entity from within this viewport. Any other viewport rotates your entity the wrong way.

   When you select for rotation, you see four circles around the entity, one on each corner (see Figure 7-9). This means the entity is ready for rotation.

2. **Click and drag any of these circles to rotate the entity. Rotate the entity until it's facing the direction you want. Then release the mouse button and you're done.**

   The result is a spawn entity that causes the player to spawn facing the center of the room. (See Figure 7-10.)

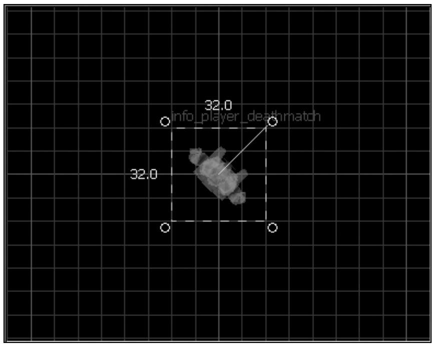


**Figure 7-10:** Rotate the spawn entity so that the player is facing the center of the room.

All right, this is another good point to save your map. Choose File➪Save As and then save your map as `dm_chapter7.vmf`.

# Chapter 8
# Putting the Pieces Together

*In This Chapter*

- ▶ Understanding the compile process of the map
- ▶ Turning your map into a playable level
- ▶ Playing your custom level in the game

If you read the preceding chapters in this book, you know how to construct all the pieces of your map. You create a basic room for your player to start the game. You dress things by placing textures on all the walls. You add some colorful light to the area and give a player a place to spawn. These are all the elements you need to create your map. Before you can play your map in the game, though, you need to turn it into a level.

There is an important difference between maps and levels. The *map* is where you put together all the pieces to make up the world for your game. However, a level is what the player can load and play in the game. A *level* is a map file that has been read, converted, and saved into a file or group of files that is read by the game and interpreted as the playable field.

## Leveling the Playing Field

A buddy of mine who now works for a popular game developer once said to me, "It's easier to drive a car when you know how it works." From that point on, I looked at a game engine the same way a mechanic looks at the engine under the hood of a car. I study the mechanics of the map files, their conversion process, and the result as seen in the game. When something goes wrong with a map I'm building, I now know where to start looking for the problem — or at the very least, I know where not to look because I know what's going on under the hood.

The compile process takes a map file and turns it into a playable level for a game. This process, as it turns out, happens to be very similar for most of the games on the market today. Each of the three stages in the compile process does its part to turn the map file into a playable part of the game. In the following sections, I explain these three steps as they occur during the compile process.

## Building the BSP

The first process the map goes through is the Valve Binary Space Partitioning (BSP or VBSP) process. Here, the map file is read and picked apart, creating only the basic compiled level, excluding lights. Each brush face is located in the map file and then checked in relation to how it interacts with the other brush faces in the map. If it touches another brush face, that brush face is split where they touch.

Take a look at Figure 8-1. Here you look down on a simple map from the top. It's a rectangular room with two walls placed within. This is what the map looks like from above before it goes through the BSP process.

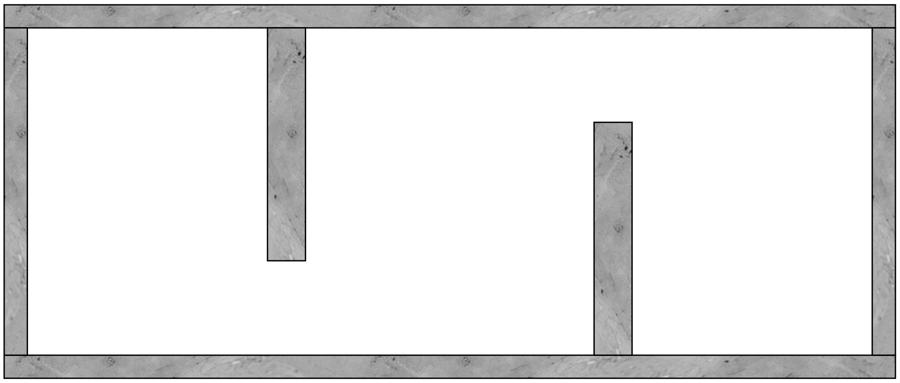


**Figure 8-1:** A simple map as seen from the top.

Now look at Figure 8-2. This illustration shows what happens after the BSP process. Each dashed line represents where the floor and ceiling of the room is split as a result of where other brush faces — such as the walls within the map — touch. The split starts where the walls touch the floor and continues either to the end of the brush or to the next split. The simple map is divided by the editor when the map is turned into a playable level.

The areas that are created by these splits are *BSP leaves.* These leaves are different than those you create in the next phase of compiling, the VIS leaves.

## Seeing what can be seen

The second process of compiling the map is the Visibility (VIS or VVIS for *Valve VIS*) process. During the VIS process, the visibility of each player is determined to assist the game in maximizing the rendering of what you see. The map is checked to define what the player can and cannot see from any point within the level by dividing the map into smaller blocks. These smaller blocks go by many names for different games, but Valve Software refers to them as *visleafs* (or VIS leafs) or just *leaves,* in *Half-Life 2.* I define them as a combination of the splitting created in the BSP process defined in the preceding section, predefined algorithms, hint brushes, and portal brushes, which I explore further in Chapter 10.

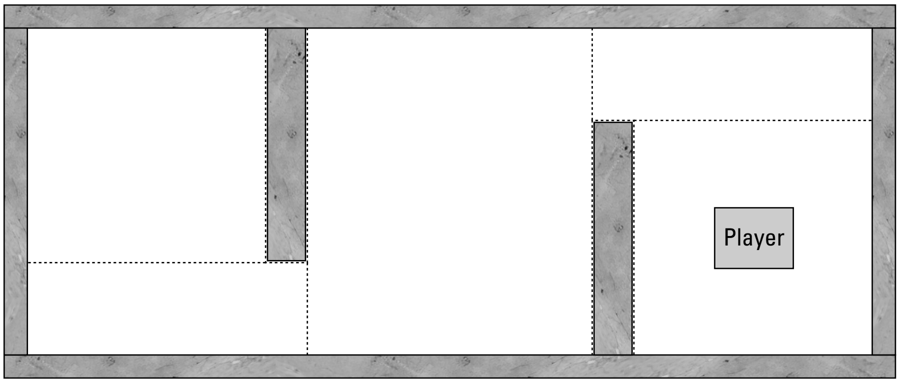


**Figure 8-2:** A map divided into areas by the editor.

When this VIS leaf data is read during the game, it determines which VIS leaves can be seen by the player, and it three-dimensionally re-creates only those sectioned-off areas for the player during the game. This re-creation is *rendering.* The computer renders the visuals within the level by determining which leaves can "see" each other.

In Figure 8-3, you can see a possible scenario of how a map might be divided into VIS leaves. Each division in the sample creates what the game defines as an *area,* which the game uses for optimization. When the player looks across the splitting in Leaf 1, he or she can see Leaves 2 and 3. Therefore, everything contained within these areas is re-created and drawn within the game because it's assumed that if Leaf 1 can see these areas, the player can, too. Because they can't be seen by Leaf 1, the other areas in the level aren't drawn in the game at this time. Leaf 2 can't see Leaf 5; therefore, when the player steps into Leaf 2, all areas except Leaf 5 could be drawn. Only when the player steps into Leaf 3 does the game re-create the entire level because only that leaf can see all other areas. The places at which these leaves split from one another are *portals* because they enable you to view one leaf from another.

This is important because you can prevent problems that can occur with the game. For example, if the game tries to render the entire level all at once, the computer has too much information to handle. This could result in the game running ridiculously slowly or crashing. You'll often notice this slowdown in the form of lag on some custom maps. You can prevent this, however, by either properly positioning your brushes during the design of your map or by adding special Hint or Portal brushes that can help you force the splitting where you want the splitting. Splitting is discussed in detail in Chapter 10.

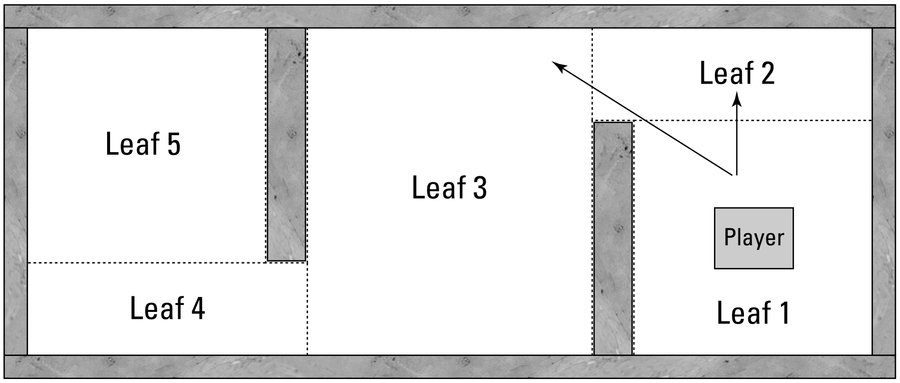


**Figure 8-3:** A map divided into VIS leaves.

### Lighting the scene

The third and final process of compiling the map is the lighting process. This process is called VRAD (for Valve Radiance) or Flare by some games, but the meaning is the same. This stage is when the light is properly added to the map, and it's often the most time consuming of the three stages. Light comes from the sky, light entities, and some other entities as well. Other map objects, such as walls, might cast shadows, but that doesn't mean the light stops at those objects. In many cases, light bounces off objects, like the sunlight bouncing off a pond or a piece of paper. The calculations go on for a default value of five bounces, which is the effect of light bouncing off five surfaces. Considering the number of surfaces that could be in a map and the amount of open space the light travels through, it's understandable why this process can take so long.

### Playing with the results

Although the BSP process is actually the first of the three stages in compiling your map, as a whole, these stages are together called the *BSP process.* The result of compiling a map is a single `.bsp` file that loads into your game as a playable level. So, if you're compiling a map file named `dm_chapter8.vmf`, the result is a new file, `dm_chapter8.bsp`.

## Processing the Pieces

Before your changes can take effect, you must compile your custom map. If you don't have your last saved map file open in the editor, open it now. If you're following along with the book, the map file you need is `dm_chapter7.vmf`. When the editor is running, you can open your map file from the menu by choosing File➪Open and selecting your `dm_chapter7.vmf` file.

Because you're in Chapter 8 of this book, resave this map as `dm_chapter8.vmf`.

The compile process is really quite simple. All the necessary settings were set up during the configuration process when you installed SDK (Software Developer Kit). All that remains for you to do is launch the compile process and click OK.

To launch the compile process, press F9. This is the shortcut for Run Map, which you can also find by choosing File➪Run Map. After the Run Map launches, the Run Map dialog box opens, offering you access to more options. As shown in Figure 8-4, the Normal compiling options are

- **Run BSP:** With three options, you want to make sure you select Normal. Because this is the first time you're compiling your map, you must run the BSP process and include everything, not just entities. You could recompile a map to update the entity information, but I recommend that you always run the BSP at the Normal setting, regardless.
- **Run VIS:** You should run the VIS process at Normal. Without it, your map will barely run in the game, if it runs at all. This is because the previous BSP process only roughed out your map into a level and left it without any optimization whatsoever, like the VIS process does.

  If you're simply trying to compile your map to confirm texture placement and basic layout, the Fast option is useful. This option quickly builds your BSP but leaves out the higher-level optimization performed during the Normal VIS process. However, this option also provides you with a false reading of frame rates, lag issues, and other things that make this compile option less useful.

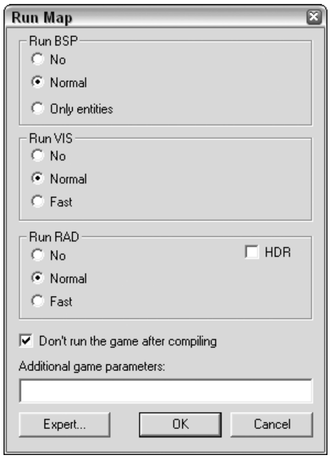
  Don't ever release your map to the public when you compile it using this Fast VIS option. This is like handing out a half-finished product, which results in poor game play.
- **Run RAD:** Keep this option set to Normal as well. Without this option, your map has no lighting effects. Yes, you could see, but only because the default light setting in *Half-Life 2* is Full Brightness. However, your map has no shadows or other lighting effects that make the finished level interesting.

  You also notice an HDR (High Dynamic Range) check box here. Selecting this option enables the HDR lighting, which allows more colors to show up in your lighting. When enabled, the player's settings are taken into consideration. The darkest area in the level is displayed at the darkest level of light and color that the player's computer can possibly display. The same occurs for the brightest levels. HDR often makes more contrast between colors and levels of light, and it can make a game look more vibrant.

WARNING!

The drawback of enabling the HDR option is that it significantly adds to the compile time. At first, while your map is still small, you aren't going to see a big difference between the different levels of compiling, such as enabling HDR or running at Fast VIS. However, as you map gets larger, the time it takes to compile your map into a playable level significantly increases. For instance, I have friends that have waited days for their very complex level to compile.

- **Don't Run the Game After Compiling:** Select this box now. This disables the option of having the compiler automatically open the game and run your level after the compile process is complete. For now, select this option so that the level doesn't load up automatically. The first time you compile your map, I want to show you the compile console, which is hidden when the game is launched.
- **Additional Game Parameters:** This text box near the bottom of the window allows you to enter special commands that are run during the compile process. Usually, you aren't going to need this. This is for advanced processing of your map.


Run Map

Run BSP: No [✓ Normal] Only entities

Run VIS: No [✓ Normal] Fast

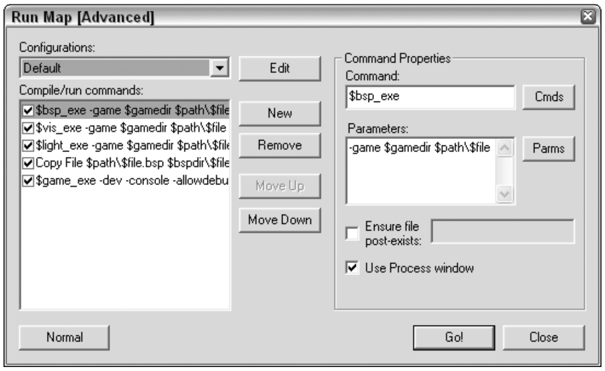
Run RAD: No [✓ Normal] Fast; HDR [ ]

[✓] Don't run the game after compiling

Additional game parameters:

**Figure 8-4:** The Normal compile options are most often used to make your map playable.

Press the Expert button at the bottom of the Run Map dialog box to open the Expert Run Map window. In Figure 8-5, the Expert Run Map window offers you a more complex list of compile options that you can select from and change. Usually, you don't need these additional options when compiling your map, but they're nice to have if you want more control over how your map is compiled.



Configurations: Default

Compile/run commands:

```
☑ $bsp_exe -game $gamedir $path\$file
☑ $vis_exe -game $gamedir $path\$file
☑ $light_exe -game $gamedir $path\$file
☑ Copy File $path\$file.bsp $bspdir\$file
☑ $game_exe -dev -console -allowdebu
```

Command Properties

Command: $bsp_exe

Parameters: -game $gamedir $path\$file

☐ Ensure file post-exists:

☑ Use Process window

**Figure 8-5:** The Expert compile options offer you more control.

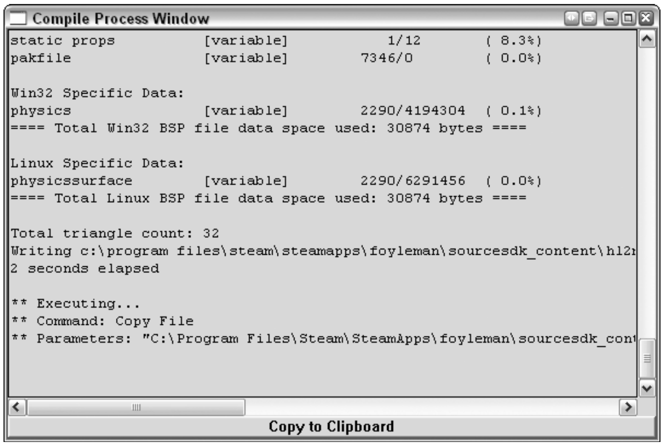
- **Configurations:** The Configurations drop-down list provides you with a list of preset options for compiling your map. You can select one for compiling your map, edit one, or create a new one based on all the other details specified in this window.
- **Compile/Run Command**: The options you see here mimic the Compile process, which I describe earlier in this chapter. First, you must run your BSP, VIS, and RAD to turn your map into a level. Then, as in the Default configuration, the compiled map is copied to the proper game directory on your hard drive and then run in the game.

  The buttons on the right allow you to change the order and details of each command, but leave that for when you have a better understanding of how things work. Making any changes now could result in errors.
- **Command:** For each command on the left side of this window, a primary command starts it. Usually, this is an executable file.
- **Parameters:** For each command, parameters must be passed on. These parameters tell the executed command what to do and how to do it.
- **Ensure File Post-Exists:** Select this check box and enter a filename if you wish to be notified that the specified file doesn't exist after the compile process is complete. You could specify your compiled BSP file here so that everything stops if no BSP file is created.
- **Use Process Window:** This is a very important option to select. This provides you with a window during the compile process that displays all the details of the process when they occur. More importantly, this window alerts you to errors that might exist in your map.

Click Normal to go back to the Normal Run Map window. Make sure that you select the Don't Run the Game After Compiling option. Then click OK to compile your map.

## Listening to the console

After you initiate the compile process, a new window comes up in the editor. This window, the Compile Process window, has a lot of text in it (see Figure 8-6). While the map is compiling, all the commands and results are printed in this window. This way, if any errors come up during the process, you can see what they are. If you select the Don't Run the Game After Compiling option in the previous window before compiling the map, this Compile Process window is covered by the game window, and you can't see whether there were any errors until you close the game window.



```
static props          [variable]          1/12       ( 8.3%)
pakfile               [variable]        7346/0       ( 0.0%)

Win32 Specific Data:
physics               [variable]        2290/4194304  ( 0.1%)
==== Total Win32 BSP file data space used: 30874 bytes ====

Linux Specific Data:
physicssurface        [variable]        2290/6291456  ( 0.0%)
==== Total Linux BSP file data space used: 30874 bytes ====

Total triangle count: 32
Writing c:\program files\steam\steamapps\foyleman\sourcesdk_content\hl2r
2 seconds elapsed

** Executing...
** Command: Copy File
** Parameters: "C:\Program Files\Steam\SteamApps\foyleman\sourcesdk_con
```

**Figure 8-6:** The Compile Process window displays the progress of the compile process.

The information printed in the Compile Process window should follow the same outline as the compile process that is described earlier in this chapter. Each of the compiling commands, BSP, VIS, and RAD are executed using the specified parameters.

The first process to run is VBSP. The output to the Compile Process window looks similar to this:

```
** Executing...
** Command: "c:\program files\steam\steamapps\foyleman\sourcesdk\bin\vbsp.exe"
** Parameters: -game "c:\program files\steam\steamapps\foyleman\half-life 2
               deathmatch\hl2mp" "C:\Program Files\Steam\SteamApps\foyleman\
               sourcesdk_content\hl2mp\mapsrc\dm_chapter8"

Valve Software - vbsp.exe (Jan  2 2006)
2 threads
materialPath: c:\program files\steam\steamapps\foyleman\half-life 2
               deathmatch\hl2mp\materials
```

```
Loading C:\Program
              Files\Steam\SteamApps\foyleman\sourcesdk_content\hl2mp\mapsrc\
              dm_chapter8.vmf
fixing up env_cubemap materials on brush sides...
ProcessBlock_Thread: 0...1...2...3...4...5...6...7...8...9...10 (0)
ProcessBlock_Thread: 0...1...2...3...4...5...6...7...8...9...10 (0)
Processing areas...done (0)
Building Faces...done (0)
FixTjuncs...
PruneNodes...
WriteBSP...
done (0)
writing C:\Program
              Files\Steam\SteamApps\foyleman\sourcesdk_content\hl2mp\mapsrc\
              dm_chapter8.prt...done (0)
Creating default cubemaps for env_cubemap using skybox materials:
skybox/sky_day01_01*.vmt
Run buildcubemaps in the engine to get the correct cube maps.
```

In the preceding example, the VBSP command is run. First, details about the command are output if you defined them within the Expert Run Map window. Then, the command is executed.

When the VBSP executable is run, the results are output to the window. This is helpful for two reasons:

- **You're notified of any errors or warnings during this process.** Without this information, you couldn't know if something was wrong with your map or what it was that was wrong.
- **You can see the progress of the compile process and not fear that your computer has frozen.** Some larger maps can take very long to compile, and you might begin to wonder whether your map is still compiling. With this window, you know.

The string of numbers, from 1–10, is displayed when executing more processor intensive operations. When the process runs, these numbers are displayed incrementally to let you know that it is still working.

TIP

Some information that shows up might look like errors but doesn't actually need your attention. When running this example map through the compile process, I experienced a potential error:

```
"No such variable '$hdrbasetexture' for material
          'skybox/sky_day01_01rt'."
```

If asterisks preceded this line of text, something requires your attention. However, if no asterisks appear, as in this example, the line is printed as information rather than a warning or an error.

```
After the BSP process, comes the VIS process. As VVIS is executed, similar lines
               of text will be output to the window as displayed here:
** Executing...
** Command: "c:\program files\steam\steamapps\foyleman\sourcesdk\bin\vvis.exe"
** Parameters: -game "c:\program files\steam\steamapps\foyleman\half-life 2
               deathmatch\hl2mp" "C:\Program
               Files\Steam\SteamApps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               dm_chapter8"
Valve Software - vvis.exe (Jan  2 2006)
2 threads
reading c:\program
               files\steam\steamapps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               dm_chapter8.bsp
reading c:\program
               files\steam\steamapps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               dm_chapter8.prt
   4 portalclusters
   4 numportals
BasePortalVis:       0...1...2...3...4...5...6...7...8...9...10 (0)
PortalFlow:          0...1...2...3...4...5...6...7...8...9...10 (0)
Optimized: 0 visible clusters (0.00%)
Total clusters visible: 16
Average clusters visible: 4
Building PAS...
Average clusters audible: 4
visdatasize:44  compressed from 64
writing c:\program
               files\steam\steamapps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               dm_chapter8.bsp
0 seconds elapsed
```

VVIS follows similar steps as VBSP when outputting to the window. The process is defined at the top, the process is executed below, and lines of text display the progress of the process while it occurs.

The third setup of the compile process is VRAD.

```
** Executing...
** Command: "c:\program files\steam\steamapps\foyleman\sourcesdk\bin\vrad.exe"
** Parameters:  -game "c:\program files\steam\steamapps\foyleman\half-life 2
               deathmatch\hl2mp" "C:\Program
               Files\Steam\SteamApps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               dm_chapter8"

Valve Software - vrad.exe SSE (Jan 16 2006)
----- Radiosity Simulator ----
2 threads
[Reading texlights from 'lights.rad']
[45 texlights parsed from 'lights.rad']
```

```
Loading c:\program
               files\steam\steamapps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               dm_chapter8.bsp
16 faces
4960 square feet [714240.00 square inches]
0 displacements
0 square feet [0.00 square inches]
16 patches before subdivision
368 patches after subdivision
1 direct lights
BuildFacelights:     0...1...2...3...4...5...6...7...8...9...10 (0)
BuildVisLeafs:       0...1...2...3...4...5...6...7...8...9...10 (1)
transfers 20392, max 131
transfer lists:   0.2 megs
GatherLight:         0...1...2...3...4...5...6...7...8...9...10 (0)
              Bounce #1 added RGB(1855, 0, 0)
GatherLight:         0...1...2...3...4...5...6...7...8...9...10 (0)
              Bounce #2 added RGB(338, 0, 0)
GatherLight:         0...1...2...3...4...5...6...7...8...9...10 (0)
              Bounce #3 added RGB(83, 0, 0)
GatherLight:         0...1...2...3...4...5...6...7...8...9...10 (1)
              Bounce #4 added RGB(16, 0, 0)
GatherLight:         0...1...2...3...4...5...6...7...8...9...10 (0)
              Bounce #5 added RGB(4, 0, 0)
GatherLight:         0...1...2...3...4...5...6...7...8...9...10 (0)
              Bounce #6 added RGB(1, 0, 0)
Build Patch/Sample Hash Table(s).....Done<0.0004 sec>
FinalLightFace:      0...1...2...3...4...5...6...7...8...9...10 (0)
FinalLightFace Done
Ready to Finish
0 of 0 (0% of) surface lights went in leaf ambient cubes.
ComputePerLeafAmbientLighting: 0...1...2...3...4...5...6...7...8...9...10
```

The VRAD is a far more complex process as represented by the amount of text that is output in the window. However, the output also provides you with much more information about your map.

First, your computer determines how much area your light can cover before the light dissipates. Like the lamp in your room, when you turn it on, your entire house doesn't light up. The light can only reach so far, and this is considered in the game. This is represented in the output as *BuildFaceLights* and *BuildVisLeafs.*

The lighting process then has to consider how light can start from a single source, such as that lamp in your room, and spread from there. The light starts from one point and then hits a surface such as a wall, a floor, or anything else in its way. The light then bounces off that surface and continues its path. Consider how light can be reflected off a white piece of paper. Well, light can reflect off anything really. Some surfaces simply reflect more light than others.

This light reflection process is calculated for six bounces. You can see this in the text output earlier as:

```
GatherLight:
               0...1...2...3...4...5...6...7...8...9...10 (0)
               Bounce #1 added RGB(1, 0, 0)
```

TIP

The VRAD step of the compile process usually takes the most time to complete. Consider all the lights that shine and reflect in your map. The light starts from a single point and shines in every direction. The light reflects off all the different surfaces in your map. This takes a lot of time for your computer to calculate.

After the light is calculated, the details about your map are printed to the window.

```
Object names       Objects/Maxobjs  Memory / Maxmem  Fullness
------------       ---------------  ---------------  --------
models                   1/1024          48/49152    ( 0.1%)
brushes                  6/8192          72/98304    ( 0.1%)
brushsides              36/65536        288/524288   ( 0.1%)
planes                  40/65536        800/1310720  ( 0.1%)
vertexes                35/65536        420/786432   ( 0.1%)
nodes                   27/65536        864/2097152  ( 0.0%)
texinfos                 5/12288        360/884736   ( 0.0%)
texdata                  4/2048         128/65536    ( 0.2%)
dispinfos                0/0              0/0        ( 0.0%)
disp_verts               0/0              0/0        ( 0.0%)
disp_tris                0/0              0/0        ( 0.0%)
disp_lmsamples           0/0              0/0        ( 0.0%)
faces                   16/65536        896/3670016  ( 0.0%)
origfaces                6/65536        336/3670016  ( 0.0%)
leaves                  29/65536        928/2097152  ( 0.0%)
leaffaces               16/65536         32/131072   ( 0.0%)
leafbrushes             16/65536         32/131072   ( 0.0%)
areas                    2/256           16/2048     ( 0.8%)
surfedges               88/512000       352/2048000  ( 0.0%)
edges                   57/256000       228/1024000  ( 0.0%)
LDR worldlights          1/8192          88/720896   ( 0.0%)
HDR worldlights          0/8192           0/720896   ( 0.0%)
waterstrips              0/32768          0/327680   ( 0.0%)
waterverts               0/65536          0/786432   ( 0.0%)
waterindices             0/65536          0/131072   ( 0.0%)
cubemapsamples           0/1024           0/16384    ( 0.0%)
overlays                 0/512            0/180224   ( 0.0%)
LDR lightdata         [variable]      14208/0        ( 0.0%)
HDR lightdata         [variable]          0/0        ( 0.0%)
visdata               [variable]         44/16777216 ( 0.0%)
entdata               [variable]        400/393216   ( 0.1%)
LDR leaf ambient        29/65536        696/1572864  ( 0.0%)
HDR leaf ambient         0/65536          0/1572864  ( 0.0%)
occluders                0/0              0/0        ( 0.0%)
```

```
occluder polygons        0/0              0/0        ( 0.0%)
occluder vert ind        0/0              0/0        ( 0.0%)
detail props          [variable]          1/12       ( 8.3%)
static props          [variable]          1/12       ( 8.3%)
pakfile               [variable]       7346/0        ( 0.0%)

Win32 Specific Data:
physics               [variable]       2290/4194304  ( 0.1%)
==== Total Win32 BSP file data space used: 30874 bytes ====

Linux Specific Data:
physicssurface        [variable]       2290/6291456  ( 0.0%)
==== Total Linux BSP file data space used: 30874 bytes ====

Total triangle count: 32
Writing c:\program
               files\steam\steamapps\foyleman\sourcesdk_content\hl2mp\mapsrc\
               chapter8.bsp
2 seconds elapsed
```

You can see that from the detail above that 1 model is in your map, 6 brushes that make up the single room, 36 brush sides, and so on. What you can also see here is the maximum number of each that is allowed in a single map. If you try and place more than 1,024 models in your map, it results in an error during the compile process.

Finally, the last few lines in this window display the last command to be run. This is the command that copies your compiled map into the directory that the game can locate. When you go to play your new map, the game can find and then load it.

```
** Executing...
** Command: Copy File
** Parameters: "C:\Program
               Files\Steam\SteamApps\foyleman\sourcesdk_content\hl2mp\mapsrc\dm_c
               hapter8.bsp" "c:\program files\steam\steamapps\foyleman\half-life
               2 deathmatch\hl2mp\maps\dm_chapter8.bsp"
```

## Spotting an error

Spotting errors isn’t always easy, so you need to read through the text in the Compile Process window carefully. Eventually, you’ll become familiar with the text here and you can skim quickly through it to recognize problems. However, if you haven’t seen errors displayed before, you might not spot them right away.

For example, I created an error in my map on purpose, as shown in Figure 8-7. I moved one of the walls away from the rest of the room and created a leak.

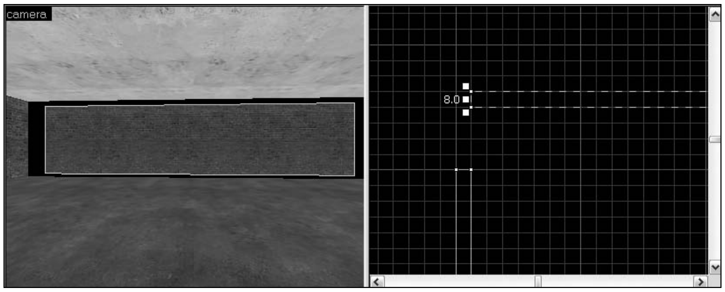


**Figure 8-7:** Don't leave any gaps between your walls that might leak out of your mapping area.

Chapter 5 shows that you need to be careful not to leave any gaps between the walls. If you leave a gap between walls, you might find an error similar to the one in the following text:

```
**** leaked ****
Entity info_player_deathmatch (-128.00 -128.00 8.00) leaked!
```

Even though the compile process doesn't seem to stop when this error occurs, it is a very serious error that must be fixed. The VVIS process actually doesn't run when there is a leak, and you'll most likely experience bad frame rates that feel like lag in larger maps.

If you do see this text in the Compile Process window, you need to locate the leak and seal it. Sometimes when working on very large maps, it's nearly impossible to find a leak with any guidance. Well, not to worry, because the editor actually shows you the way.

To find your leak, close the Compile Process window. Choose Map➪Load Pointfile. A small window comes up. Just make sure that the pointfile you're loading has the same name as your map, such as `dm_chapter8.lin`. Then click Yes to continue.

TIP

If the pointfile that you're prompted to load isn't the same name as your map, click No in the dialog box prompting you to load the pointfile. Save your map again, recompile your map, and then load the pointfile again. The correct pointfile is then loaded.

The pointfile contains information about your map and the brushes within it. This file is generated during the compile process and used to construct your BSP. When an error occurs, such as a leak, an additional file is created that is referenced by your pointfile. It contains the coordinates of your leak, which can then be referenced in the Hammer editor.

When you load your pointfile into the editor, you can find a red line. I made this red line a little more obvious in Figure 8-8.

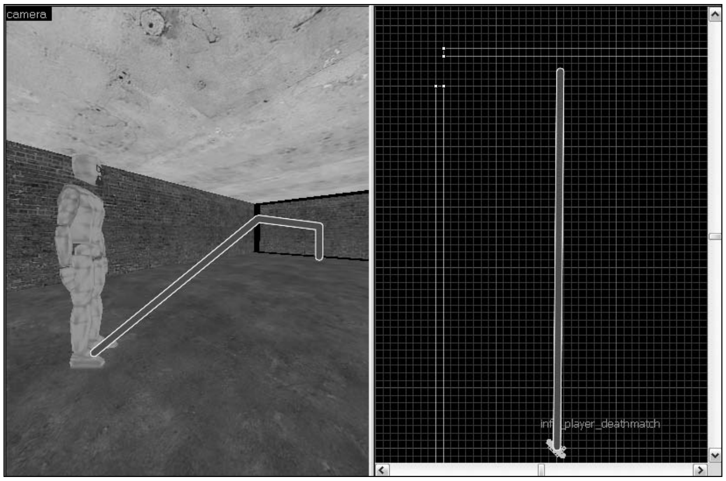


**Figure 8-8:** A red line shows you where your leak is located.

The red line starts from one of your entities. In the case of Figure 8-9, the line starts from the player spawn point. Then, the line goes through your map and stops where it exits through your leak.

TIP

If you're having trouble finding the red line that points to your leak, try zooming out in your 2D viewports. You have a better chance of locating the red line if you can see more of your map.

After you locate your leak, you can fix it.

## Playing the Result

Compile your map again, but this time, you deselect the option that stops your map from running. Press F9 to open the Run Map window. Make sure that all your compiling commands are set to Normal. Deselect the Don't Run the Game After Compiling option and then press OK.

Your map runs through the compile process as it did before. However, when the compiling is complete, the game loads automatically. When the game loads, it proceeds to automatically load your new level and then spawn your player into this level.

WARNING!

Make sure that you have loaded the Half-Life 2: Deathmatch game type at least once before compiling and playing your new level. If you do not, the game will crash in lieu of loading your level.

As with all deathmatch game types for *Half-Life 2,* you must first click OK to play in the game. Then you can feel free to run around your new map. You can see how one corner is very dark and that another is very red because of the red light that was placed there, as shown in Figure 8-9.

When you're done exploring, simply press Esc to access the game options. Then click Quit to end and close the game. The game window closes, and the editor appears. This way, you can continue to improve your custom map.

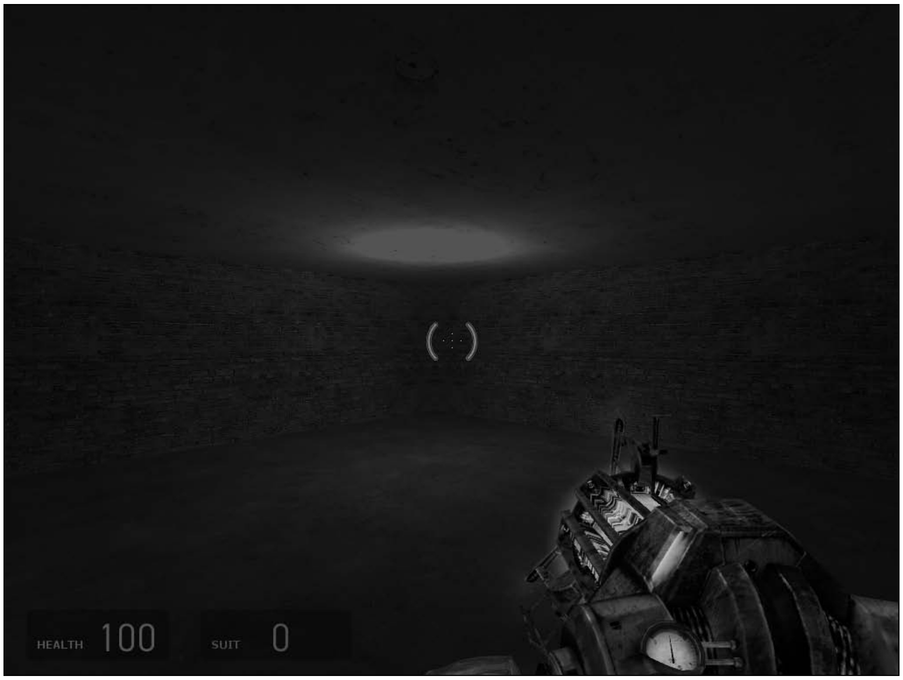


**Figure 8-9:** Your new map is now playable.

# Part III
# Expanding on Your Creation

"Are you using that 'clone' tool again?!"

## *In this part . . .*

You have the knowledge to make your own map, but now it’s time to become an artist. Take that one-room level and turn it into something you can take online and play with your friends. Step outdoors (at least virtually) for the first time, and I will show you the door that leads to a bigger, better level that plays with ease for either one or many players.

# Chapter 9

# Expanding Your Map with Additions

### In This Chapter

- ▶ Duplicating selections
- ▶ Joining rooms with halls
- ▶ Adding doors
- ▶ Texturing and lighting the hall
- ▶ Play-testing the new map

If you've read the previous chapters, you know how to put together a simple room — really just a box with a light where you can play. In this chapter, I help you expand on this creation and make it more interesting. I show you how to add another room to your map, placed diagonally from your current structure. Then you connect the two with an L-shaped hallway.

If you don't already have the map from Chapter 8 loaded in the editor, do so now by choosing File➪Open and opening `dm_chapter8.vmf`. I show you how to build your additions onto this already constructed map.

## Making Copies

You could go through the same steps that I outline in Chapter 5 to create your second room, but instead, I show you another option. Because the goal is to create a room just like your first room, you can duplicate the existing structure. Then you can move your copy diagonally in your 2D window to create the second room.

1. **Select all the brushes and objects that make up your first room.**

   Do this from the camera viewport. Move around the camera viewport by using the arrow keys. Select the walls, ceiling, and floor by Ctrl+clicking from within the camera viewport, as shown in Figure 9-1. Also select the light that is in this room. However, don't select the player spawn point; you don't want to make a duplicate of this entity yet. Later, you'll add additional spawn points to the map for multiple players.

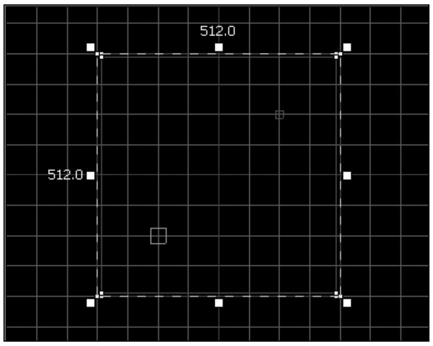


**Figure 9-1:** Select everything in your map except for the player spawn point.

2. **Zoom out so you can see more of your map and then adjust your grid size to 64 units.**

   The window is probably zoomed in on this one room. By zooming out, you can better move the copy to another location. Use the mouse wheel to zoom out farther.

   Adjust your grid size to 64 units by pressing [ or ] to lower or raise the grid size. This makes it easier for you to move your copy and place it more accurately. In the next step, you place your copy 64 units away from the original room on a diagonal. Because your selection snaps to the grid when you move it, adjusting the grid makes it that much easier to move into place.

3. **Shift+click and drag your selection to create a copy of your selection. Drag it up and to the left diagonally.**

   Continue dragging until there is one 64 unit block between your copies, as shown in Figure 9-2.

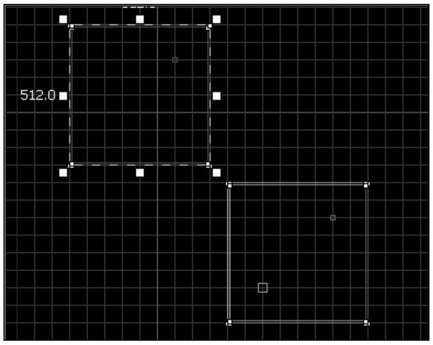


**Figure 9-2:** Drag your copy up and to the left of the original to create two separate rooms.

REMEMBER

Clicking outside your selection results in making a new selection rather than moving your current selection. If you accidentally deselect everything, try pressing Ctrl+Z to undo the last selection, which causes you to deselect your room.

4. **With your duplicate room in place, press Esc to deselect everything.**

Now you have two identical rooms. To make your new room different, change the color of the light. The light in the original room is red, so, for this example, make the light in the new room blue. Follow the steps in Chapter 7 to adjust the light color in this room.

This is a good time to save your map. Save your map as `dm_chapter9.vmf` by choosing File➪Save As, entering the new name, and clicking Save. As I mention in Chapter 2, save your maps in stages. If you find that one of the steps you take during the building process creates an error within the map, going back to a good copy is easier than finding the source of the problem one brush at a time or worse, starting completely over.

## Joining Rooms

You now have two rooms in your map but no way to move between them as a player. To solve this, you need to add a hallway connecting the two rooms. In this section, I show you how to create an L-shaped hall below your new room and to the left of your original room. The halls connect to the center of a wall on each of your rooms. Here's how:

1. **After making sure that everything is deselected, select the Nodraw texture.**

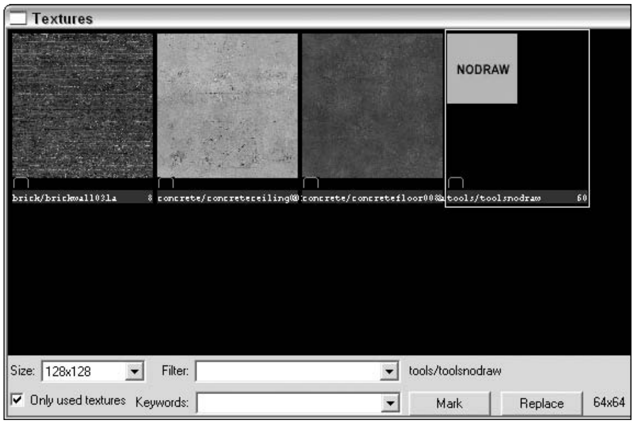
   This is the texture you use to build all your structures before applying decoration. Click Browse from the Textures group on the right side of the editor to open the Textures window.

2. **Because this texture was used before, select the check box on the bottom of this window to display Only Used Textures, as shown in Figure 9-3.**

   This filters your view to only those textures that are used in your map. The Nodraw texture surrounds the outside of your rooms and therefore would be part of this list.

3. **Double-click the Nodraw texture to select it.**

   The window closes.



Size: 128x128 Filter: tools/toolsnodraw
☑ Only used textures Keywords: 64x64

**Figure 9-3:** In the Textures group, select the Only Used Textures option.

## Drawing your halls

How wide should the hall be? How tall? Determining the width and height of structures, such as hallways, is something that comes with experience. You don't have a set number that you must use when constructing different areas of a map, but you do have ideal numbers. The hall you're creating is 128 units wide, and the height is equal to that of your rooms. A width of 128 units makes your player feel comfortable and also allows multiple players to pass each other in the same hall without having to squeeze through.

1. **Select the block tool either from the left side of the editor or by pressing Shift+B.**

   With this tool, you can draw the brushes in your map that make up your hall.

2. **From within the Top X/Y viewport, draw out a solid brush for your first hallway.**

   Use the same method that you use when creating a room by drawing a solid brush and hollowing it. (See Chapter 5 for more on how to do this.) You should end up with a brush that looks similar to Figure 9-4 in length and width. It measures 128 × 384 units, but the height of the brush is currently too small.

3. **From one of the other 2D view ports, adjust the height of the brush so that it is equal to that of the two rooms you're connecting.**

   The height is 128 units, like in Figure 9-4.

4. **With your hallway started, press Enter to create the new brush.**

   You have half of the hallway structure created and are ready to create the second.

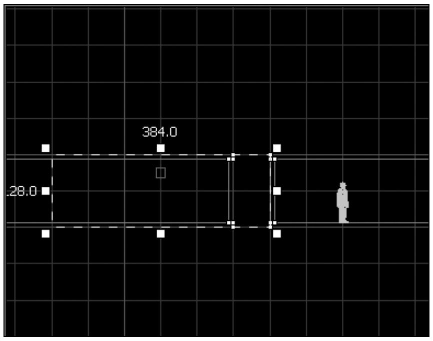


**Figure 9-4:** Adjust the height of your halls to equal that of the adjoining rooms.

5. **Draw a new brush between the second room and the previous hallway brush to connect the two, as shown in Figure 9-5.**

   You draw this brush from the Top X/Y viewport. The brush is sized at about 256 ×128 units, and the height is the same as the previous brush, 128 units.

6. **With the second hallway brush drawn, press Enter to create the brush.**

   Your hallway shape is laid out with two solid blocks, like that in Figure 9-6. However, solid brushes aren't going to do you or your player much good. You need to hollow them.

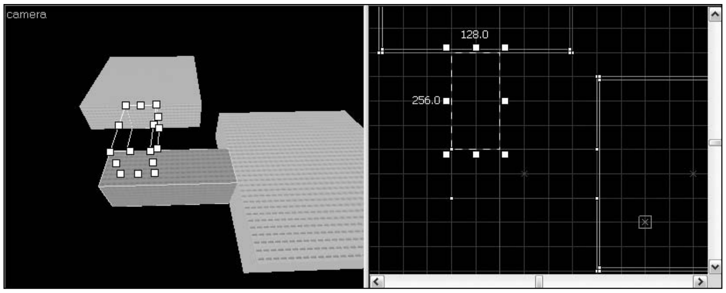


**Figure 9-5:** Create the second hallway brush from the Top 2D view port.

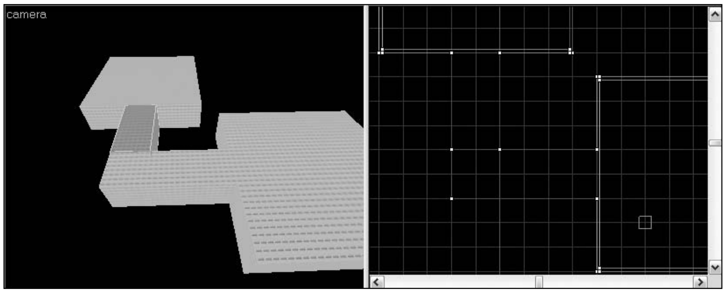


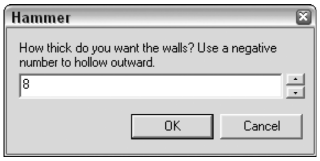
**Figure 9-6:** Press Enter to create the brush, and your hallway begins to take shape.

## Making room to play

It's time to hollow these two brushes, one at a time. You should hollow each of these brushes separately so that you don't create errors with the map structure. Here's how:

1. **With one brush selected, press Ctrl+H to bring up the Hammer window where you can enter your wall thickness (see Figure 9-7).**

   If you hollowed the original room in your map, you entered a wall thickness of 8, which represents 8 units. Enter **8** again and then click OK to continue. The result looks like Figure 9-8, where your brush is hollowed and looks more like a room of its own.


Hammer

How thick do you want the walls? Use a negative number to hollow outward.

8

**Figure 9-7:** Press Ctrl+H to hollow a selected brush.

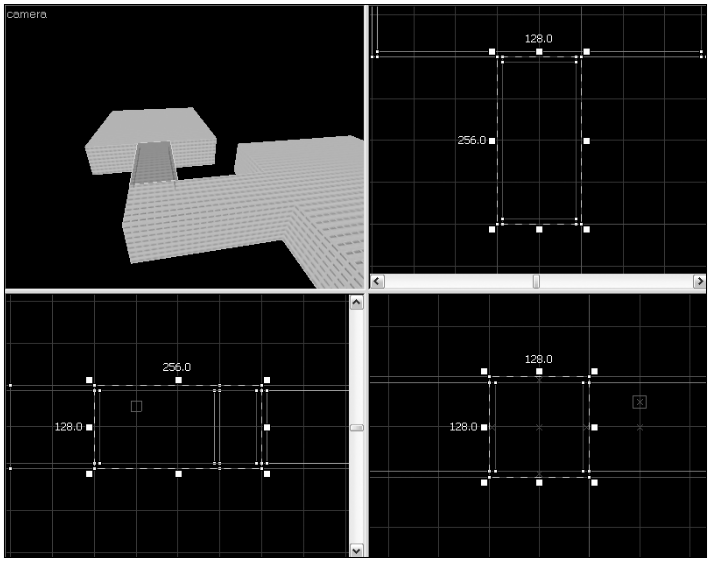


**Figure 9-8:** Your hollowed brush with a wall thickness of 8 units.

2. **For the second hallway brush, select it, press Ctrl+H, enter a value of 8 units for the wall thickness, and click OK.**

   You end up with two hollowed hallway brushes, like that of Figure 9-9. However, you have no way to get into these halls from either room. You solve that problem in the next step.

3. **Select one of the brushes that make up your halls.**

   When you do, the entire hall section is selected rather than just the single brush. This same thing occurs if you hollow the original room in your map because these brushes are now grouped.

4. **Ungroup your brushes by selecting each of the two hall sections that were hollowed. With the group of brushes selected, press Ctrl+U to ungroup them.**

   Now, when you select one of the brushes, you select only that one because they are all ungrouped. Make sure to ungroup the brushes in both hall sections.

REMEMBER

To start making access ways through these soon-to-be hallways, you need to delete the brushes that you don't need and then adjust the others to fill in any gaps. However, to manipulate each brush that was hollowed, you have to ungroup them.

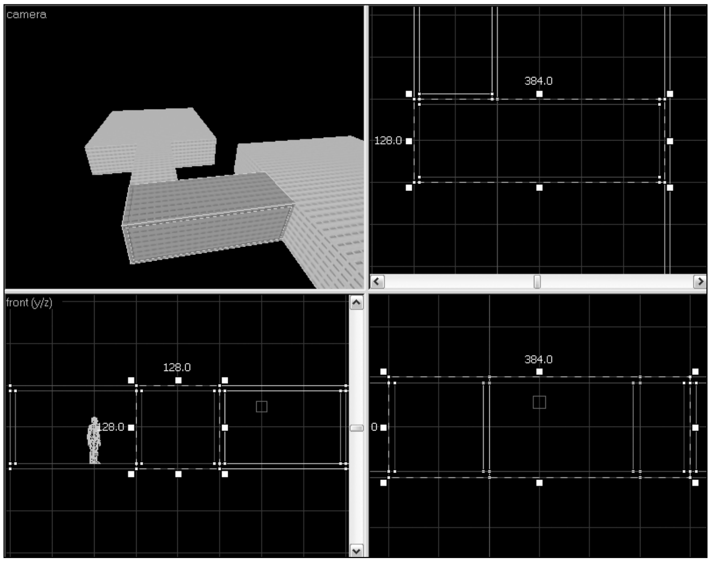


**Figure 9-9:** The second brush is hollowed, and you need to create access to both brushes.

5. **Selectively delete the brushes that you don't need.**

   When the brushes were hollowed, walls were created on all six sides of the hollowed brush. You need only the side walls, floor, and ceiling for your halls. So, select each of the three smaller-sized walls that get in the way. Two of these walls butt against the rooms, and one butts against the other hall section, as depicted in Figure 9-10. After you select the brushes, press Delete to remove them.

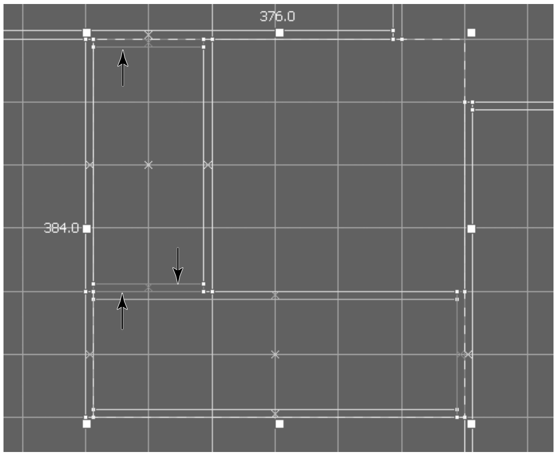


**Figure 9-10:** Remove the unnecessary brushes from your halls.

6. **Deleting those brushes leaves you with a gap between your original room and the new hall brushes. To close this gap, drag the edge of the hallway brushes, thereby stretching them until they touch the room.**

   Figure 9-11 show the gap that must be closed to prevent the leak error, as I describe in Chapter 5.

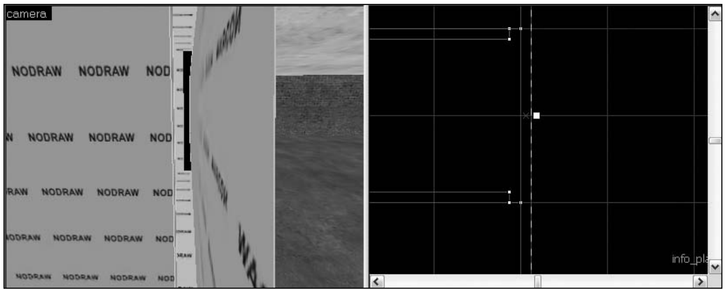


**Figure 9-11:** This gap between your brushes must be closed to prevent errors later.

7. **In the camera viewport, select one of the walls that needs to be stretched in order to close the gap on one side.**
8. **It's easiest to see this gap in the 2D top viewport, so select the middle, white handle that represents the side of your brush (the other two handles represent the corners of your brush) and then drag the handle until it meets with the wall of your room, as shown in Figure 9-12.**

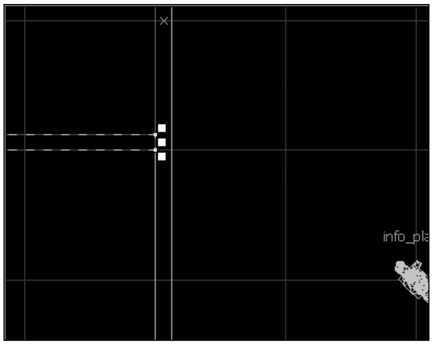


**Figure 9-12:** Select the brush and close the gap by stretching it.

TIP

The little white blocks on the corners and sides of a selected brush are *handles.* You can grab these handles with your mouse and then drag them around. In this case, you're grabbing one of the handles to resize the brush in one direction. You could also grab a corner handle to do this, but that might result in resizing the brush in two directions rather than just one.

WARNING!

Make sure that you grab the handle when you want to resize the brush. If you don't, you move the brush rather than resize it. If you're unsure that you resized the brush rather than moving it, zoom out in your view and make sure that you didn't create a gap on the other side of your selected brush.

9. **Perform the same action in Step 9 on the brush located on the other side of this hall section.**

   This brush also creates a gap that must be closed. Select the brush and drag the edge handle until the wall meets with the other wall, and the gap is closed.

TIP

If you see any other gaps in your map, make sure to close them. Gaps will create errors in your map during the compile process, causing your map not to compile.

All of the gaps are now closed between your brushes, but you still don't have a means of moving the player between the rooms and hall sections. You need to create these entrances. Here's how:

1. **Adjust your view in the viewports so that you can see the inner corner of the halls.**

   Notice that one brush cuts off the player's access between these two hall sections. All you need to do is resize that brush and make an opening. The process is very similar to closing the gaps.

2. **Select the brush that splits the two hall sections from within the camera viewport and then select the middle handle on the left of this brush to resize it.**

3. **Click and drag the brush to create your opening, as shown in Figure 9-13.**

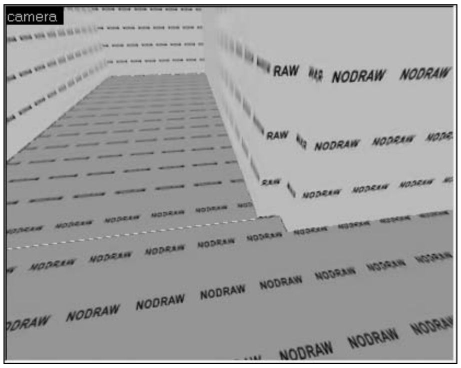


**Figure 9-13:** Resize the brush that is separating the two hall sections to create an opening.

Look closer at Figure 9-18 where you dragged the one wall brush to create an opening between the two hall sections. You see that the two brushes in the hall corner don't meet properly. This happens when your grid size isn't small enough. With grid snap turned on, the edge of the brush you're resizing snaps to the grid lines in your 2D viewports. In Figure 9-14, my grid size is too large to get the two corner brushes to fit properly. The solution is to adjust my grid size.

4. **Lower the grid size by pressing [. Keep pressing [ until the grid size is 8, the same size as your wall thickness.**

   You can see the grid size in the lower-right corner of the editor, as is explored in Chapter 4.

5. **After the grid appears small enough to fix the corner issue, select the brush that you resized to open the hall sections toward each other and adjust the brush so the corner meets cleanly, as in Figure 9-14.**

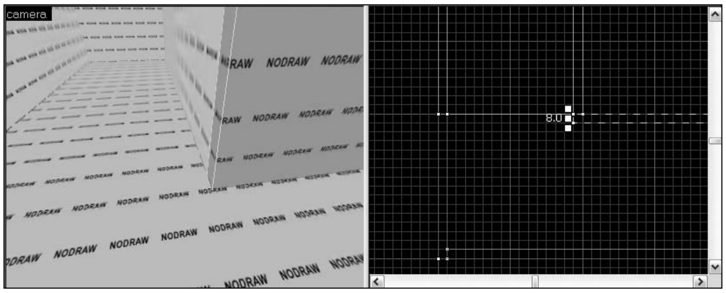


**Figure 9-14:** Adjust your grid size to more precisely size your brushes in the 2D viewports.

## Cutting in some doors

You now have two rooms and a hallway between them. However, you don't have access yet to your new hallway. What you need to do is cut some doorways. I show you two methods to accomplish this task:

- Clipping brushes
- Carving brushes

The next two sections explain both of these techniques.

### Clipping brushes

Move the view in the camera viewport into the first room so that it faces the wall you want to cut. You're going to cut an entrance into the wall of this room.

Instead of stretching and moving brushes, you're going to use another tool: the Clipper. Here's how:

1. **Select the wall you want to edit and then select the Clipping tool by pressing its icon or by pressing Shift+X.**

   To activate the Clipping tool, press the Clipping tool icon, as shown in the margin, on the left side of the editor.

2. **Before you use the Clipper tool, adjust your 2D top viewport so that you're looking at the location on the wall where you want to create your opening.**

   You could create the opening in this wall from any 2D viewport, however; doing this from the top view is easier, as shown in Figure 9-15.

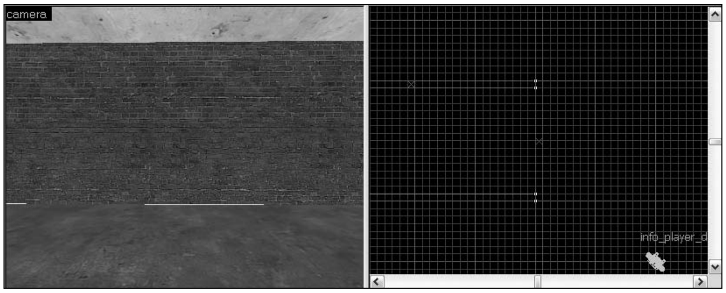


**Figure 9-15:** Adjust your view so that you're looking at the spot you want to clip.

3. **Create your clipping plane by clicking and dragging in the 2D window, thereby creating a line by which to clip.**

   When you click, you see a white handle like the handle used to resize selected brushes. When you drag your mouse, a second handle shows up and is placed where you let go of your mouse. Between the two handles is a teal line, which is your *clip line.* It is on this line, as shown in Figure 9-16, that your selected brush splits.

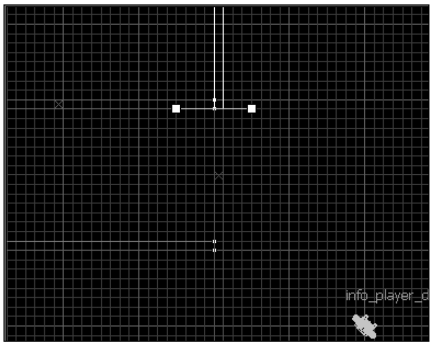


**Figure 9-16:** The white clip line is where your brush is split.

TIP

If you click in the 2D viewport, place the first handle of your Clipper, and forget to drag the mouse to create the second handle — don't worry. You can now click and drag the first handle to create the second handle. To move your clipping plain, click and drag either of the handles at either end of the clipping line.

4. **Press Shift+X until both halves of your brush are outlined in white.**

   Because the clipping tool has multiple functions, only half of the brush you're about to split is outlined in white (see Figure 9-21). The clipping tool can cut off half your brush or it can split your brush in two. The portion of your brush that is outlined in white is the portion that remains in your map after you proceed with the clipping operation.

To cycle through your clipping options, press Shift+X. Each press provides you with one of three clipping options:

- Cut off one side of the brush.
- Cut off the other side of the brush.
- Split the brush in two.

You want the third option, which splits your selected brush into two brushes. When both halves of your brush appear, you selected the correct clipping option.

5. **To split your brush, press Enter.**

   You should now have two brushes where one used to be, as shown in Figure 9-17. All that's left is to resize the brush that is cutting off the entrance into the hall, and you're done.

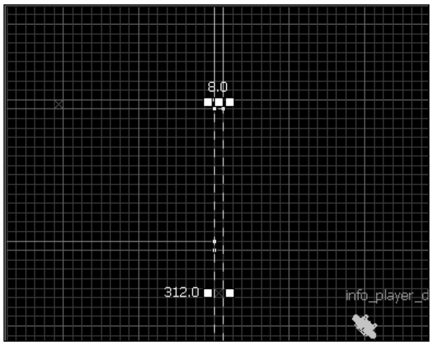


**Figure 9-17:** Press Enter to split the wall in two.

After you finish the preceding steps, switch back to the Selection tool (Shift+S). You now need to select and resize the wall brush that covers the entrance to the hall.

Select the brush that needs to be resized. Then grab the middle handle at the end of the brush that is covering the hall entrance. Click and drag that handle until it no longer blocks the hall, but not so far that it goes beyond the outer wall of the hall and creates a leak. You want a nice corner, like you had on the inside of the halls. The result is a clean looking entrance into the hall, like Figure 9-18.

## Carving brushes

For the entrance in the other room, I show you another method — *carving.* Using this method, you create a brush that is equal in size to the entrance that you want to create. You overlap this brush — with the wall — where you want to create your entrance. Then you carve the shape of this brush out of the wall.

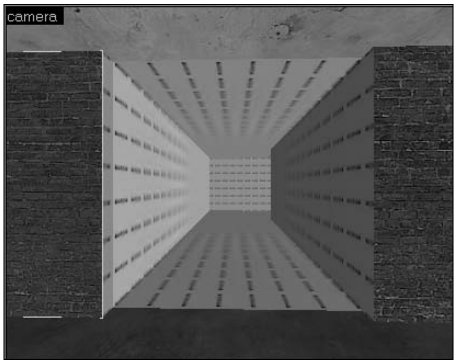


**Figure 9-18:** Resize the wall for a nice entrance into the hall.

WARNING!

Be careful when using the carving tool. Any other brush that your carving brush intersects with is carved. If you're intersecting with the floor, the floor is carved, too. Make sure that you only overlap those brushes that you want to be affected when using this method.

1. **Adjust your viewports so that you can see the area in the other room where you want to create your entrance to the hall.**
2. **Select the Block Creation tool by pressing Shift+B.**

   With this tool, you create the brush that is used to carve your entrance.
3. **Draw a brush equal in width and height to the entrance you want to make.**

   Make the depth a little bit larger so that it encompasses the hole you want to carve, as shown in Figure 9-19. After you draw the brush, press Enter to create it.

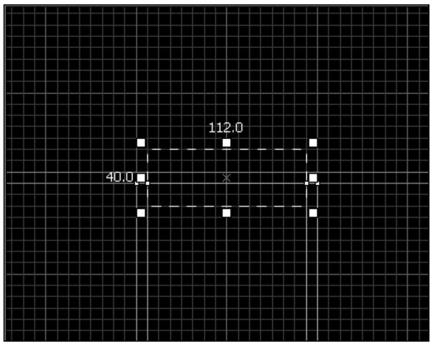


**Figure 9-19:** Create a brush that is the size of the hole you want to carve.

WARNING!

Make certain that the new brush isn't overlapping the floor or ceiling brushes. Also, make sure it isn't overlapping any of the brushes inside the hall. Remember, anything that this brush overlaps is carved by it.

4. **Switch back to the Selection tool by pressing Shift+S and then select the new brush that you want to use for carving.**
5. **When you're ready to carve, press the Carve icon (in the margin on the top of the editor) or press Ctrl+Shift+C.**

   After you carve, you might not notice that it worked right away because the carving brush is still in your map. Press Delete to delete this brush because it is still selected. You now have a perfect entrance into the hall from this room, as shown in Figure 9-20.

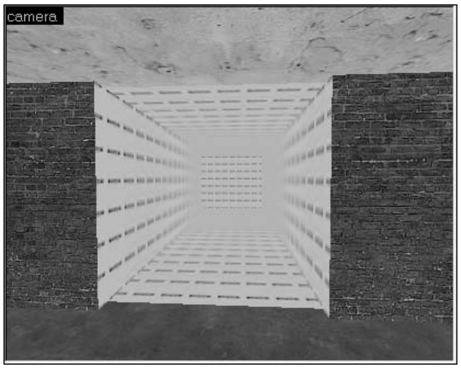


**Figure 9-20:** Carving the wall leaves you with a hole shaped like your carving brush.

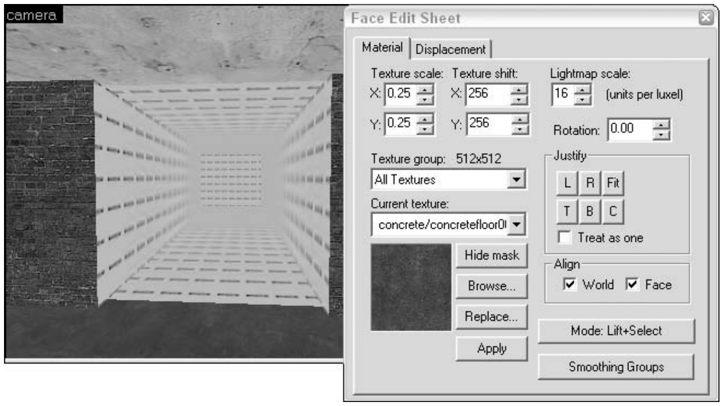
## Retexturing the Hallway Walls

If you've followed to this point, you now need to texture the walls in your hall. The walls are currently textured with the Nodraw texture, which doesn't do you any good in the game. You can apply the same textures that you have in the rooms to the walls in the hall.

1. **Open the Texture Application tool by selecting its icon on the left side of the editor.**

   You can also open the Texture Application tool by pressing Shift+A. This opens the Face Edit Sheet window and makes a number of other texture application options available that are otherwise unavailable. These options and a description of this window are explored in Chapter 6.
2. **Starting with the floors, click the floor of your textured room to select that texture.**

   When you click it, the brush face is selected; more importantly, the texture that is currently applied to that brush face becomes your active texture in the Face Edit Sheet window, as shown in Figure 9-21.



Texture scale: X: 0.25 Y: 0.25
Texture shift: X: 256 Y: 256
Lightmap scale: 16 (units per luxel)
Rotation: 0.00
Texture group: 512x512 All Textures
Current texture: concrete/concretefloor0
[ ] Treat as one
[✓ World] [✓ Face]

**Figure 9-21:** Select a brush face to select its texture when the Texture Application tool is active.

3. **To quickly apply the selected texture, right-click the floor in the hall. Do so with both floor brushes in the hall and then move on to the ceiling brushes.**

4. **Click the ceiling brush to select its texture and then right-click the untextured ceiling brushes in the hall to apply this texture.**

   Repeat this process with the walls until your hall is completely textured.

   Texture all the brush faces inside your map. Don't leave any Nodraw texture showing, or your map won't look right in the game. Don't texture the outside of your rooms and hall. Because the outside won't be seen by the player, it's better to leave those brush faces textured with the Nodraw texture.

5. **After you're done texturing, switch back to the Selection tool by pressing Shift+S.**

   This way, you don't accidentally change any of the textures you placed.

## Lighting the Path

If you've read to this point, your structures are complete. However, when the player enters the hallway, he isn't going to see much because there isn't much light. You need to add some more lights to your map, and I show you a very easy way to make duplicates of the lights already in your map.

Start by adding more light to one of your rooms and its adjoined hallway leg. Here's how:

1. **Select the Selection tool by pressing Shift+S and make sure you have a good view of your entire room in the Top 2D view port.**

   You need a good view of the entire room so that you can see where you're placing your copied light.

2. **Select one of the lights within one of the rooms.**
3. **Shift+click your light and then drag it to another corner in the room.**

   Repeat this process until you have a light in three of the four corners of the room. Leave one corner a little dark so you can hide as a player in the game or something else in it later.

4. **Make one more copy of this light and then place it in the middle of the hall section closest to that room.**

   Using the same light color that you used in the room lights the hallway enough for the player.

   Repeat this process within the other room. The result is a level that has half blue light and half red light. Your 2D Top overview of the map looks something like Figure 9-22.

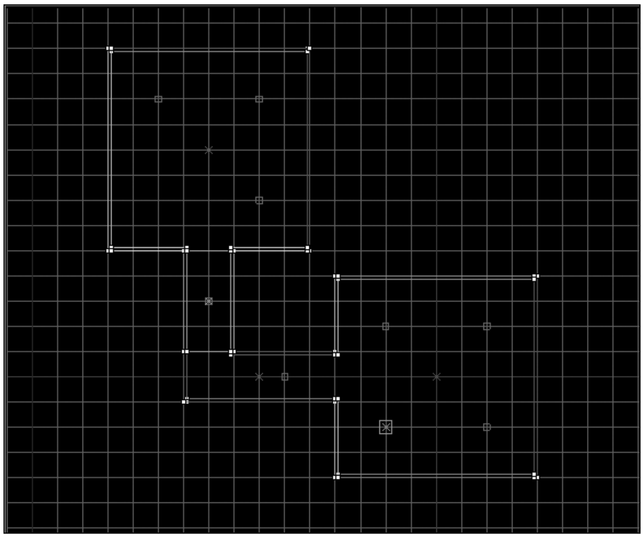


**Figure 9-22:** Place copies of your lights around the map, half red and the other half blue.

## Running in the Halls

When your changes are complete, save your map so you don't lose any of the hard work you just completed. You're ready to test your hard work in the game. Go ahead and compile the map again to create your playable level. To do so, press F9 to bring up the Run Map window. Make sure the three compiling commands are set to Normal. Also, deselect the Don't Run the Game After Compiling option. Click OK to run the compile process and launch your new level in the game.

TIP

You might want to first compile the map with the Don't Run the Game After Compiling option selected. This way, you can look for possible errors in the Compile Process window before running the map. However, if an error is in your map, you'll notice it in the game, when the game doesn't load your map, or after you close the game window and review the Compile Process window in the editor. That message window is still there when you close the game.

When the compile process completes, the game loads automatically and starts your custom level. When the level starts, click OK to enter the game, and have fun exploring. You have two great-looking rooms with matching halls, like you see in Figure 9-23. One half of the map is blue, and the other half is red, thanks to the colored lights.

While looking around, take notice of your textures and brushes. Make sure your textures line up and look good. You don't want to see any seams in the middle of the wall that might make the wall look like it's broken. Make sure all your corners are straight and nothing is sticking out or too far in. Most importantly, feel proud that you just completed this map, and you can now have fun playing in it.

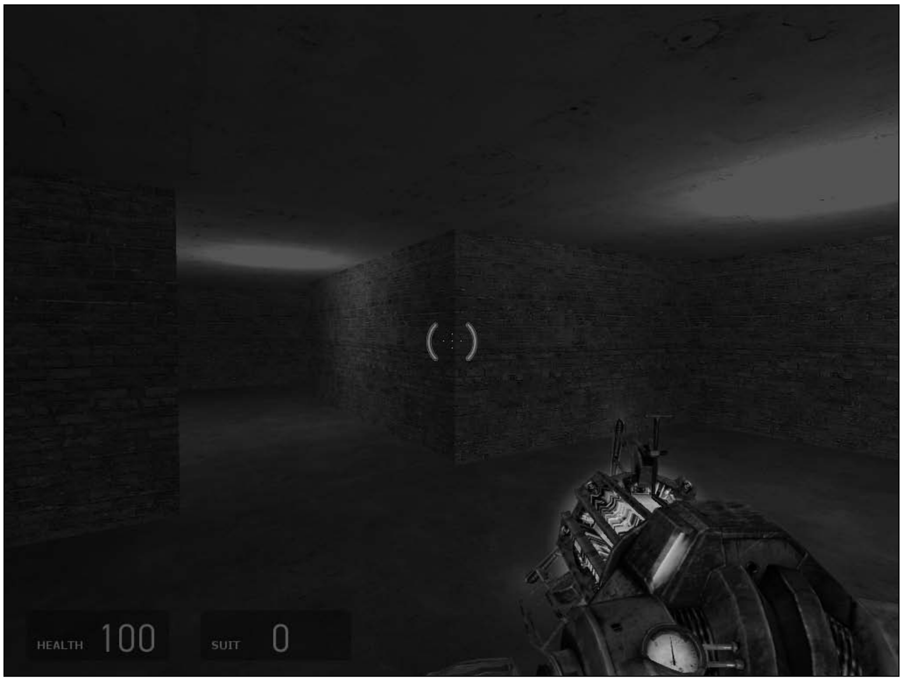


**Figure 9-23:** Your first map is growing into something great.

# Chapter 10

# Building with Optimization in Mind

*In This Chapter*

- ▶ Looking at your level as a developer
- ▶ Mitering corners
- ▶ Dividing the map into areas
- ▶ Viewing the map from a new perspective

If you built your map in Chapter 10, it is pretty small: two rooms and an L-shaped hallway. If your map is four or more times larger, optimization becomes an important concern. When you play-test your new level within a game window that you launch from the editor, you might notice that the game plays in a choppy manner. Player movement isn't smooth at all, and it feels as though the game is lagging. With a poorly optimized map, this will actually happen during a real game even when the editor isn't loaded in the background. Your map might end up unplayable.

When you experience these slow downs or jittery issues, it's your computer's way of asking for help. Your level has too much information, and it can't handle it all. Think of all the game elements it must load, display, and keep track of, including all textures, brush faces, and even other players running around the map. These elements make your computer work very hard.

Optimization techniques come into play at this time. The methods listed in this chapter need to be employed when you're building a map. When you know how to build a map, it's a good time to explore these methods. Afterward, I walk you through some testing methods that help you to find and fix trouble spots in your map.

TIP

No matter how small your map, optimizing it is always a good idea.

## Seeing What the Game Can See

Most problems that occur in maps are caused by allowing the player to see too much of the map at once. In Chapter 8, I explain the compiling process and how the level is rendered within the game based on leaves and portals. In the following sections, I cover methods of displaying onscreen images in the manner that the game has rendered them. Understanding what the game is doing to create your visuals is the best way to understand how to control your visuals with your mapping technique.

### Accessing your commands

Before you can understand the hows of optimizing, you need to know the "whys." When your game lags and your computer struggles to keep up, you can enter a few commands into the console of the game to find out what your computer is working so hard at. These commands show you what is being rendered within the game regardless of where the player is located in your level.

When you can see what is being done in the game, you know what needs improvement. If your player can't physically see around the corner of your hallway but the computer is drawing the room that's over there anyway, something needs to stop this. There is no reason why the game should think the player can see around corners that he or she actually can't see around. Fixing this is one example of optimization. Here's how you do it:

1. **Load the last saved version of your map and resave it under a new name.**

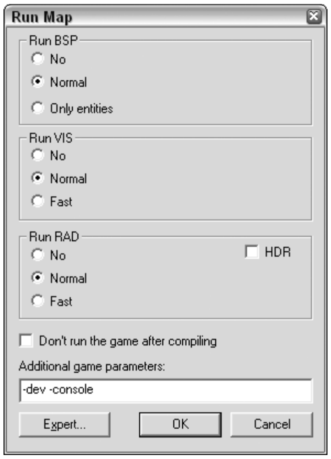
   The last version was `dm_chapter9.vmf`, so open this file and then save it as `dm_chapter10.vmf`.

2. **Open the Run Map window to compile the map but first enter some additional parameters to run your map.**

   I explain the compile process in Chapter 8. However, the Additional Game Parameters text box at the bottom of the Run Map window has not yet been used. Before compiling the map, enter **–dev –console** in this text box, as shown in Figure 10-1. These commands are used when the game launches your map, and they allow you to enter developer commands into the now accessible console.

3. **Click OK to compile and run the game after compiling.**

   The game loads the same as it has before after compiling your map. However, after it's loaded, you see a lot more text information as well as a window — that you never saw before — display briefly. This window is your Console window where you can enter special commands for the game.



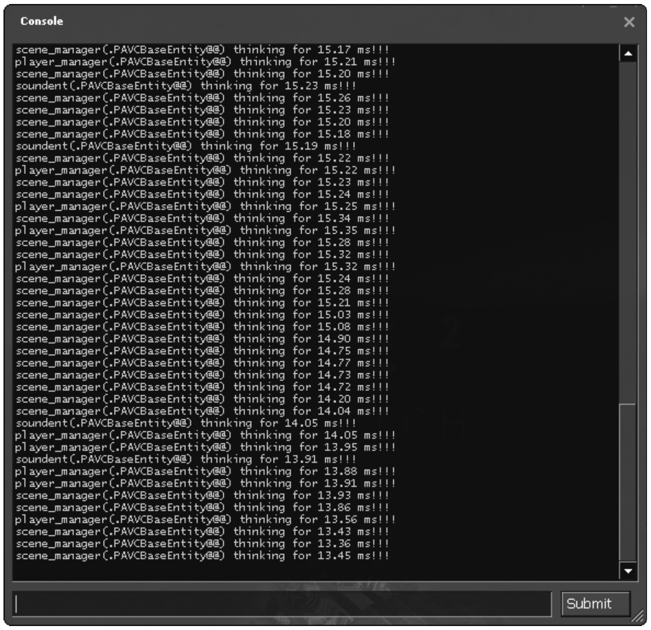
Run BSP: [✓ Normal]
Run VIS: [✓ Normal]
Run RAD: [✓ Normal]

Additional game parameters: -dev -console

**Figure 10-1:** Enter **–dev –console** into the Additional Game Parameters text box.

4. **Click OK to enter the game.**
5. **Open the Console window by pressing the ~ key.**

   Pressing the ~ key opens the Console window in the game, as shown in Figure 10-2. The Console window is now an available option because you entered the special commands before compiling and running the map.
6. **Open the Console again while the map is running and type the command you want to run for testing your map.**



```
scene_manager(.PAVCBaseEntity@@) thinking for 15.17 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 15.21 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.20 ms!!!
soundent(.PAVCBaseEntity@@) thinking for 15.23 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.26 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.23 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.20 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.18 ms!!!
soundent(.PAVCBaseEntity@@) thinking for 15.19 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.22 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 15.22 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.23 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.24 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 15.25 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.34 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 15.35 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.28 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.32 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 15.32 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.24 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.28 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.21 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.03 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 15.08 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.90 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.75 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.77 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.73 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.72 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.20 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 14.04 ms!!!
soundent(.PAVCBaseEntity@@) thinking for 14.05 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 14.05 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 13.95 ms!!!
soundent(.PAVCBaseEntity@@) thinking for 13.91 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 13.88 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 13.91 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 13.93 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 13.86 ms!!!
player_manager(.PAVCBaseEntity@@) thinking for 13.56 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 13.43 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 13.36 ms!!!
scene_manager(.PAVCBaseEntity@@) thinking for 13.45 ms!!!
```

**Figure 10-2:** The game Console window allows you to enter special commands run by the game.

TIP

While writing this book, an update for the Half-Life 2 SDK was released on Steam. After updating, I began running into some minor issues. One of these issues was that when I started a game to test my new level, I would immediately be kicked out of the game. This was because the game's default time and frag limit was set to 0 — a limit that is always met at the start of the game. The solution is to add two more commands to the line in Step 2 above. Add +mp_timelimit 20 +mp_fraglimit 20. This tells your test game that you have 20 minutes or 20 frags (kills) before it should end.

## Outlining your world

Several commands can be run, but only a few of them provide you with information about your level that can help you understand how your level is built, or *rendered,* by the game. This information helps you use important optimization techniques in your map so that it runs better in the game as a level.

All of these commands, if used by a player within the game, could be used as an unfair advantage against another player. With this in mind, the developers of the game have designated these special commands as cheats. In order to use these cheats, you must enable them.

The first command that you enter in the console is **sv_cheats 1**. This enable cheats within the game which will allow you to enter the additional commands outlined below. Type **sv_cheats 1** into the console window and press Enter.

Next, type the command **mat_wireframe 1**. When you're done, press Enter and then close the Console window by pressing the ~ key to see the results. What you see is something like Figure 10-3.

REMEMBER

Don't worry about these settings affecting your game play later. These settings reset to their default values after you shut down and restart the game.

You now see white lines all over your map. These white lines represent the outlines of the polygons that the game renders for the player. The more polygons your player can see, the harder your computer works to render the level on your screen. Your goal is to reduce the total number of polygons rendered in the level so your computer doesn't have to work so hard.

You can gather from this, as shown in Figure 10-3, that the game thinks your player can see a lot more of the level than you can actually see. Should any room that the player can't physically see be rendered? Well, of course it shouldn't. You can fix this through optimization.

Open the Console window again by pressing the ~ key. Enter the same command again, but this time, change the command value to 3. So, enter **mat_wireframe 3**, press Enter, and then close the Console window.

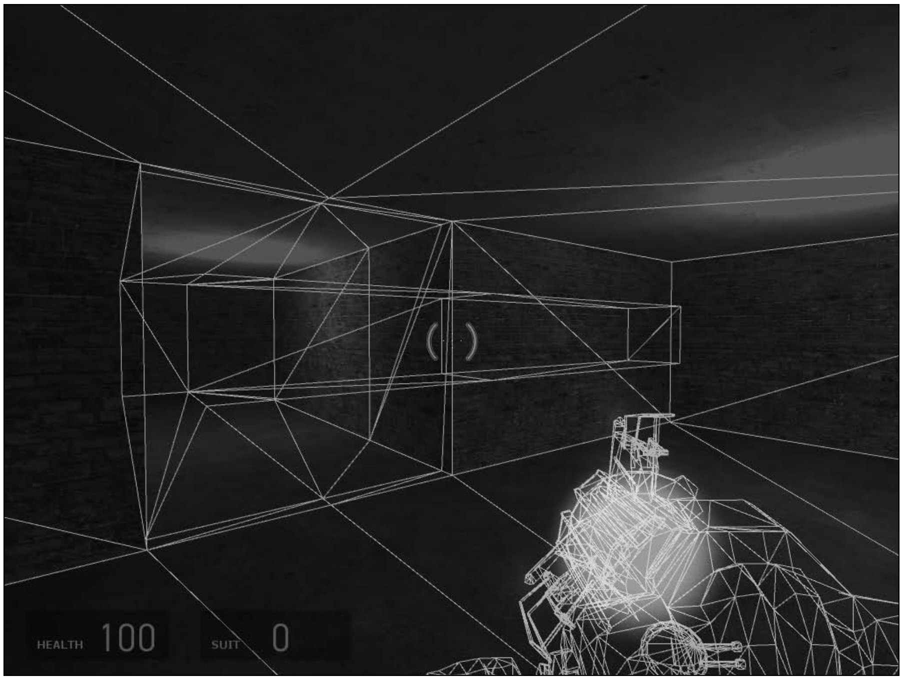


**Figure 10-3:** The game, as shown with `mat_wireframe 1` entered in the Console window.

This command, as well as a few others, can be entered with different values to produce different results. In the case of the command `mat_wireframe`, you have four options:

- `mat_wireframe 0`: This option turns off the wireframe viewing in the game.
- `mat_wireframe 1`: If you've followed the chapter to this point, this is the command you first ran (refer to Figure 10-3). With it, you can see an outline of all the polygons currently being drawn and rendered in the game.
- `mat_wireframe 2`: This option looks similar to `mat_wireframe 1` except that only those polygons that the player can physically see in the game are outlined.
- `mat_wireframe 3`: This option outlines only the visible BSP (Binary Space Partitioning) leaves, similar to VIS (Visibility) leaves that I describe in Chapter 8. The difference between these leaves is that BSP leaves contain the geometry of the map, but VIS leaves contain what the player can see. This command option provides you with enough detail to see how your level is created in the game, without the clutter of the additional lines (see Figure 10-4).

TIP

If you are entering these commands as they are explained above, try this shortcut: Instead of retyping the entire command, press the up arrow to display your last entered command and then simply change the value at the end of the command. Then press Enter to execute the command. It's sometimes easier than retyping the entire command several times in a row.

When you enter **mat_wireframe 3** in the Console window, your view is less complex. The lines in this view exist in the `mat_wireframe 2` view, but there are fewer of them. This less complex view of your level helps you to best optimize your map.

In Chapter 8, the compile process is outlined. The first stage is VBSP, when the map is broken into BSP leafs. You now see this after entering `mat_wireframe 3` in the Console window. To make the game environment look better and appear more realistic, these BSP leafs are then broken further into polygons, which you see after entering `mat_wireframe 1` in the Console window.

The second stage of the compile process is VIS. This process determines which VIS leafs (or leaves) can see each other. The game knows now that any VIS leaf that can be seen from the VIS leaf in which your player is standing needs rendering in the game.

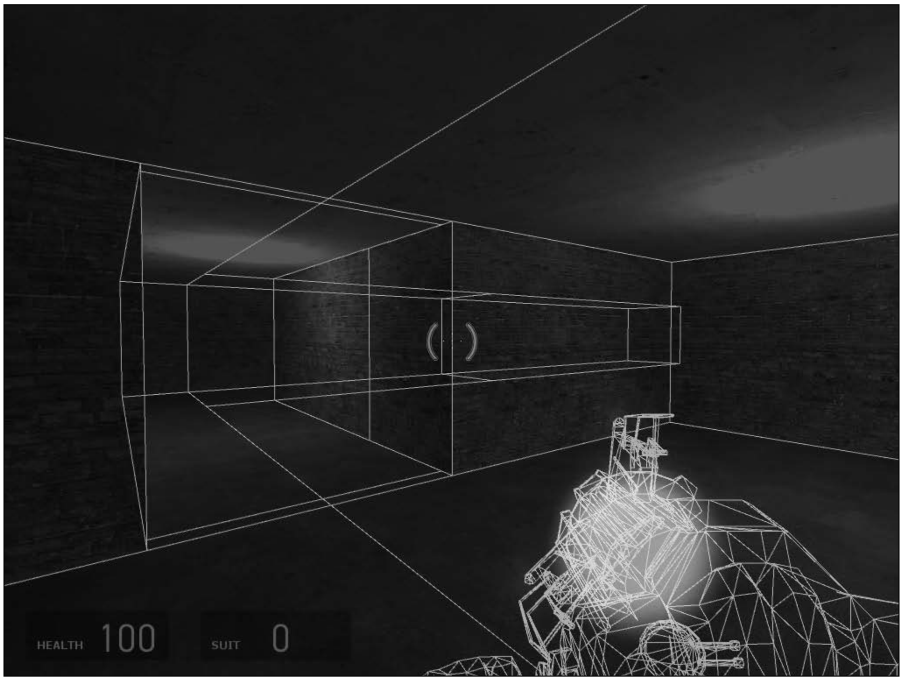


**Figure 10-4:** The game, as shown with `mat_wireframe 3` entered into the console.

I know it's confusing to know which leaf is which. Some BSP leaves handle the geometry of the level, and some VIS leaves handle what the player can and cannot see. However, anytime that I refer to *leaves* in this book, I am referencing VIS leaves. VIS leaves are the important leaves because they refer to what the player can see, which affects what is rendered — rendering puts strain on your computer. I specify when I refer to BSP leafs.

If you take some time to wander around your level with the `mat_wireframe 3` option enabled, you might end up with some questions. Why can I still see so much of the map? Why did the compile process split the map in this manner? I could have split the map better, so how can I have control over the compile process?

I answer the last question throughout the rest of this chapter — and in so doing, your other questions are answered, too.

## Mitering for Mappers

*Mitering* is not just a word for carpenters and woodworkers. Mitering is also an important part of mapping. When carpenters miter the joint between two pieces of wood, they bevel the edges of both pieces at a 45-degree angle to form a 90-degree corner. In Figure 10-5, two corners are shown in the mapping editor. The left corner is not mitered; a player looking at this corner from within the game sees three brush faces because the thin edge of one wall is exposed. The right corner right is mitered; a player looking at this corner sees only two brush faces.

This non-mitered edge creates additional geometry within the game. This edge increases the number of brush faces, resulting in additional polygons and more splits to surrounding surfaces. Figure 10-6 is an example of this additional geometry. Sure, this won't have an impact on a small, two-room level, but if this were a larger map with more corners, problems during game play could arise.

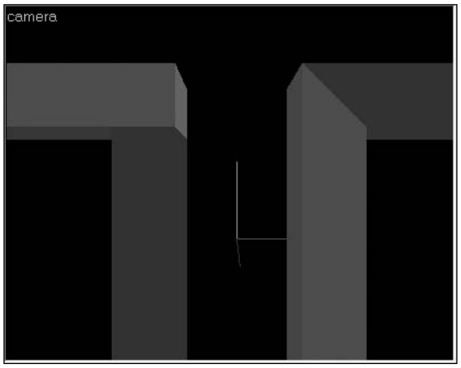


**Figure 10-5:** The corner on the right is optimized through mitering.

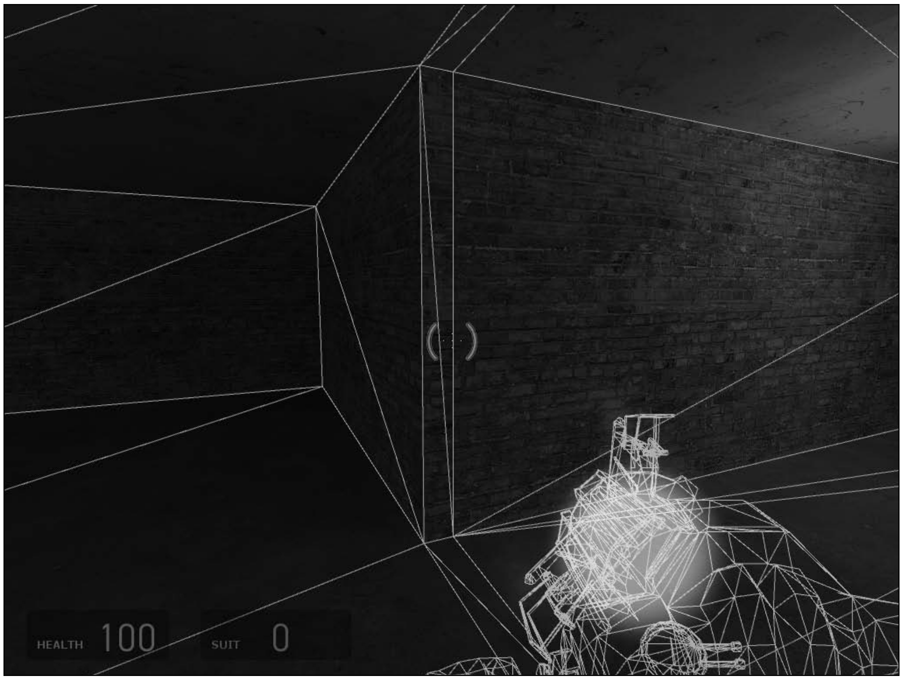


**Figure 10-6:** Failing to miter corners creates additional geometry that the gun is aimed at.

When you create maps, you need to perform all the mitering work to the walls and other brushes as you go along. If you wait until the map is big and complicated, missing some of the corners is easy.

## Fixing those corners

In the editor, go back to your map (`dm_chapter10.vmf`) to fix those corners. In this section, I show you how to miter your corners by moving the edges of your brushes. Because the outside of the map is never seen, you aren't concerned about the outside corners. Instead, five inner corners need your attention. In Figure 10-7, I circled the five corners for your reference.

To understand what a brush edge is, think of what makes a rectangular brush. A rectangle has six sides, and an *edge* is where each side meets another. So, when you're looking at the flat side of a brush, like that of Figure 10-8, the corner at which one side meets another is its edge. These edges can be manipulated to allow you to turn that rectangle into a trapezoid or any other shape.

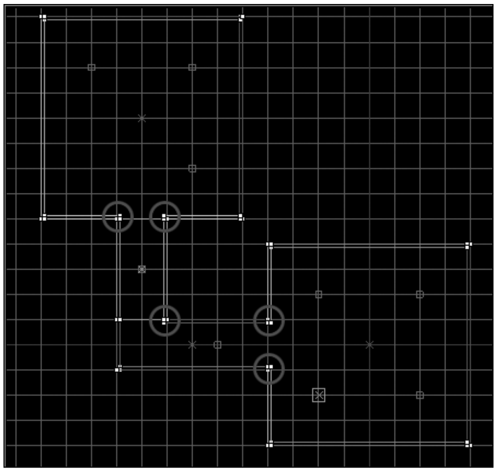


**Figure 10-7:** These five corners can be seen from within the game and should be mitered for optimization.

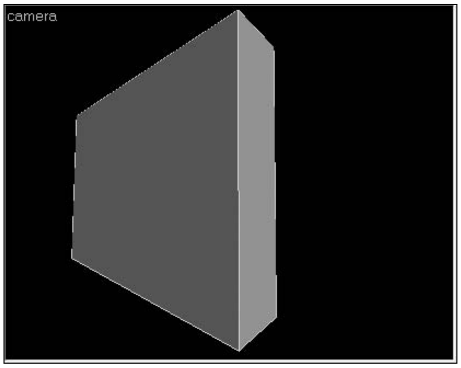


**Figure 10-8:** An *edge* is where two brush faces meet.

To create your mitered corner, follow these steps:

1. **Zoom in close to the corner that you want to miter so you can get a good view.**

   I started with the corner in the middle of the hall and zoomed in close in the 2D Top viewport so that I'm looking down on the corner that needs mitered.

2. **Adjust your grid to 8 units.**

   Your grid size needs to be equal to that of your wall thickness. Because you wall thickness is 8 units, your grid should be 8 as well.

3. **From the camera viewport, select one of the two walls.**

   You want to move one corner of the selected brush to create your 45-degree bevel.

4. **Press Shift+V to activate the Vertex Manipulation tool.**

   This tool can also be activated by pressing its icon located on the left of the editor and shown in this margin. White and yellow handles, representing each corner and edge of the brush, appear in all the viewports.

   Like the Clipping tool I use in Chapter 9, the Vertex Manipulation tool has multiple options that can be accessed with multiple activations of the tool. To cycle through these options, press Shift+V. Each press provides you with one of three options:

   - Manipulate corners and edges.
   - Manipulate corners only represented as white handles.
   - Manipulate edges only represented as yellow handles.

   You want the third option, which avoids possible errors in making your adjustments to the brush. You don't want to accidentally move the top corner without moving the bottom corner of the brush. Moving the entire edge is, therefore, safer in accomplishing this task. When you see only yellow handles on the edges of the selected brush, as shown in Figure 10-9, you selected the correct option for this tool.

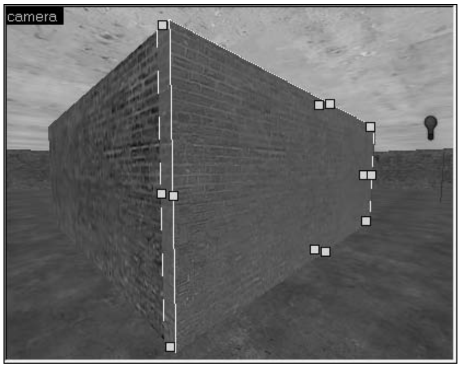


**Figure 10-9:** Activate the Vertex Manipulation tool multiple times for different options.

5. **Click and drag the brush edge that you want to move until you have a 45-degree angle, like the one in Figure 10-10.**

6. **After one wall is complete, select the other wall and do the same for it.**

   When you're done, you have two mitered brushes that meet perfectly with each other and look like Figure 10-11. You effectively reduced the number of polygons and splits within your level.

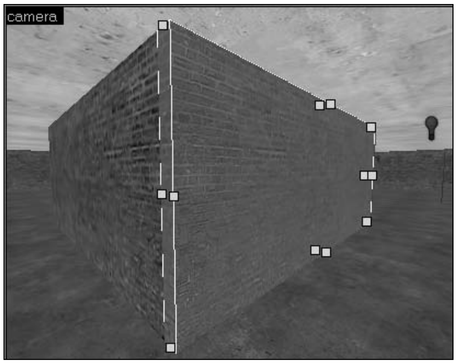


**Figure 10-10:** Drag the brush corner until you have a 45-degree angle.

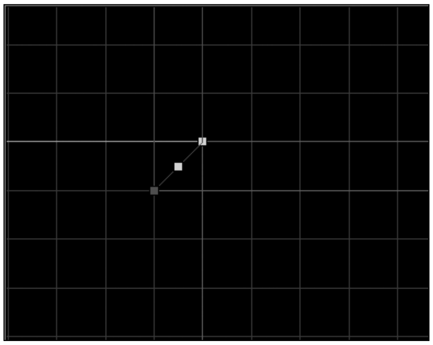


**Figure 10-11:** After you miter your corner, you have fewer visible brush faces.

Continue with this method until all the corners within the playable area of your map are mitered.

However, one more corner might be overlooked — the corner in the middle of the hall. This hallway started out as two separate brushes, and you can still see where they are divided in the floor and ceiling. These brushes also need to be mitered to increase the optimization in the level, and the process is the same as that of the corners.

Adjust the floor and ceiling brushes so they create a mitered edge. The result looks like Figure 10-12.

After you miter the visible corners in your map, think about the corners that you can't see, such as those outside your map. Because the player never sees these areas of the map, you don't need to optimize them. You need to worry about only what the player sees because everything else is simply ignored by the game.

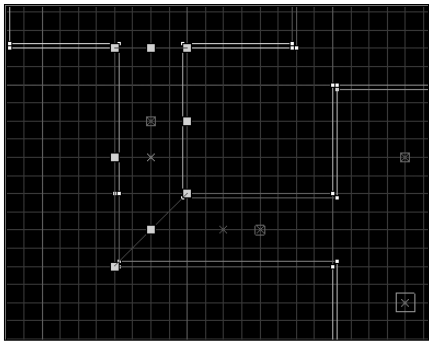


**Figure 10-12:** Miter the floor and ceiling in the hall, too.

## Combining multiple brushes

Corners aren't the only things that can be adjusted to reduce the splitting effect. Sometimes you end up with multiple brushes that make up a single wall. One such case exists in the hall.

If you created the hall from two hollowed brushes, one length of wall would now be composed of two separate brushes (see Figure 10-13). These were two separate blocks before they were hollowed into two hall sections. What you're left with is a single wall made of two brushes. This needs optimized to prevent possible splitting later.

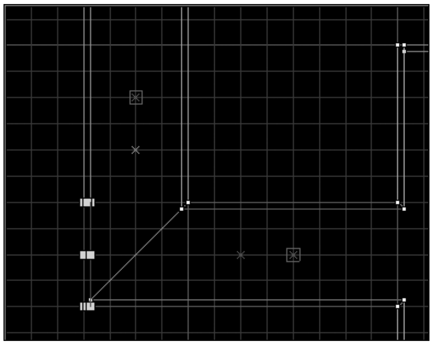


**Figure 10-13:** Walls that don't need split need adjusted.

Select the short length of wall and delete it by pressing Delete. Then select and stretch the longer length to fill the gap left behind by the previous brush. This leaves you with one, long brush that makes the wall that consisted of two separate brushes. Having one, long brush helps with the optimization of your map.

## Automatic optimization

Although the compile process can optimize some brushes for you, don't rely on it. The compile process can automatically reduce the brush face count, regardless of any settings. When the compiler looks at two brush faces that meet, it also looks at the textures applied to them. If the texture values are exactly the same, like with a continuous texture, the compiler merges the two brush faces. However, if you copy and paste the same texture throughout the level (as you do for the map in this book), the compiler automatically merges your brushes for you. Just remember that if you decide to make any changes, such as texturing the hallway differently, you have to go back and test the results in the game because you might need to make some adjustments for better optimization in the editor.

Luckily, when you test your map later with the developer commands (see the earlier section, "Accessing your commands"), you see whether an issue needs your attention. All the splits in the brushes are outlined.

Go back through your map and make sure that all the textures line up properly. Select one floor texture and paste it throughout all the adjoining floors. Do the same for the ceiling textures and wall textures so you know that any brush faces that touch are optimized automatically for you.

The splitting of your brushes based on distance is automated. If you make a map of a huge outdoor area that consists of a really large, single brush for your ground, this could cause difficulty for the game to render. The compile process breaks your map into cubes of 1,024 units to resolve this difficulty. At every 1,024 units starting from the 0, 0, 0 (X, Y, Z) axis point in the center of the mapping world, your map creates a split.

This might not be convenient, however, when you're trying to create a well-optimized map, but it's something that has to be dealt with. This also must be considered while designing and constructing your map.

Take a look at your map from within the 2D Top viewport. Zoom out so you can see all your construction and try to picture where each 1,024 split occurs.

In Figure 10-14, I emphasized with bold lines where these 1,024 splits occur. The first room was created in the middle of the mapping area. Therefore, splitting starts in the middle of this room.

If the room is created away from these 1,024 splits, the only splits made in this room are those that you create with your brushes. Now that you know about this automatic process, it can be prepared ahead of time before the map construction begins. However, because it was not considered before building, you adjust for it now by moving all your brushes and other objects in the map to a better location on the mapping grid.

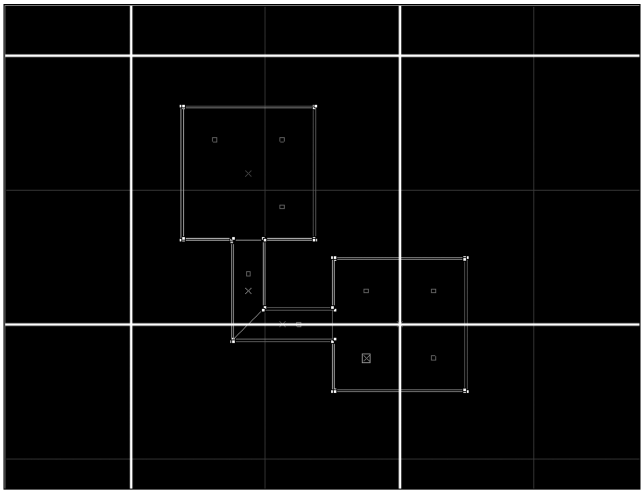


**Figure 10-14:** The compiler automatically splits your map every 1,024 units.

Choose Edit➪Select All to select everything in your map. Then click and drag your selection to a location where the least amount of splitting occurs. You might need to adjust your grid size to place the selection exactly where you want it. I had to lower my grid size to 8 units.

TIP

You may find it difficult to see where these split locations are located within the 2D viewports. In Figure 10-16, I enlarged my grid to 512 units to better see where every 1,024 unit grid line crossed in my map. This is where the 1,024 splits mentioned above will occur. Then, after making a mental note of this location in the Top 2D viewport, I adjusted my grid to 8 units so I could properly position my selection.

I moved my selection down and to the right slightly, as shown in Figure 10-15. Because walls often hide the location of splits, I considered this in my placement. Now, rather than having splitting that runs through the middle of the room, I reduced the splitting for a more optimized map.

After you finish moving the contents of your map and all the other previous steps are complete, test it out in the game. Press F9 to open the Run Map window, and then compile and run the map.

When the level opens in the game, open the Console window by pressing the ~ key. Type **mat_wireframe 3** in the Console window, press Enter, and then close the window and take a look around (see Figure 10-16). You see an improvement compared with your level before the optimization techniques were employed. Fewer splits are in your brushes, and already, less of the map is rendered for the player. However, you can still do more.

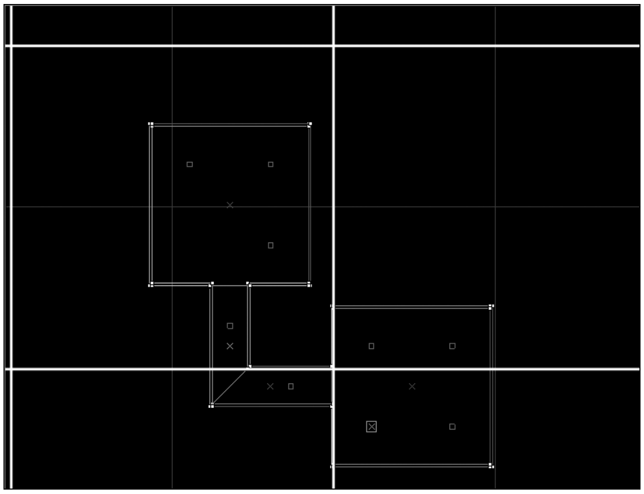


**Figure 10-15:** Adjust your map for more control over where the 1,024 splits occur.

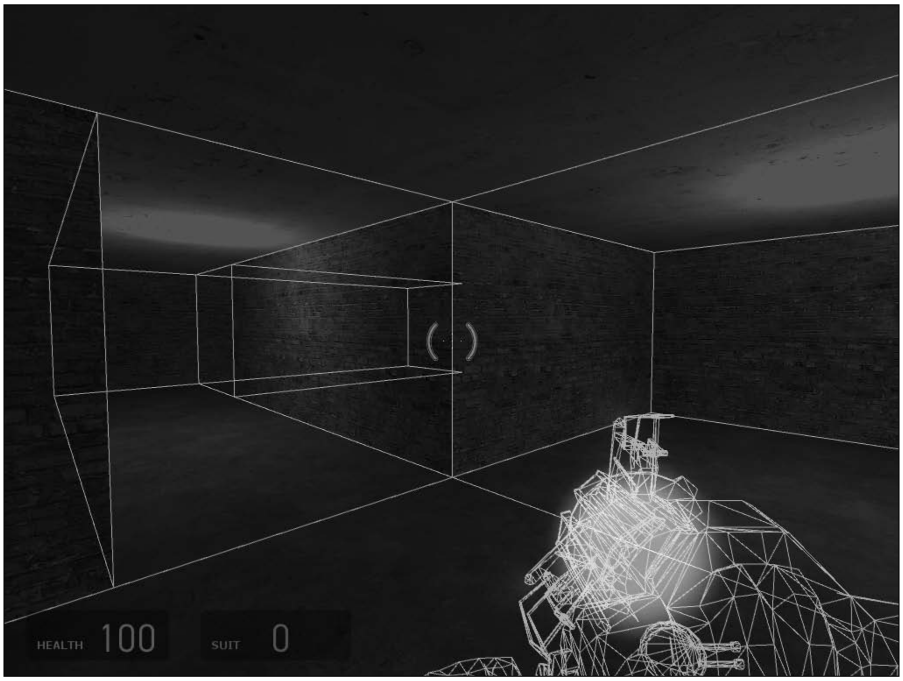


**Figure 10-16:** Less of your map is now rendered with fewer splits.

For now, save the map over the existing file, `dm_chapter10.vmf`.

## Creating portals

Mitering corners reduces polygons, which helps with optimization. However, you still have to solve the issue of too much area being rendered within the game. The problem isn't that bad in a small map, but it could be better. Your computer shouldn't have to work hard at rendering more of the level than your player can actually see. The solution is to add portals.

As I discuss in Chapter 8, *portals* define VIS leaves within the map. This affects what the game thinks the player can and cannot see. You can create a couple of portals in your map through the placement of Hint brushes. In the example map, these Hint brushes split the map into three areas so that the outer two areas — the two large rooms plus some of the attached hall — can't see each other. Because they can't see each other, the game doesn't render the contents of the room you aren't located in.

Before showing you how to create and place these Hint brushes, see what your VIS leaves look like beforehand. The view is similar to what you saw when running the `mat_wireframe` command in the console but different in that the following method helps you see VIS leaves rather than BSP leaves.

### Viewing your leaves in-game

You can view VIS leafs in you level in two different ways:

- One by one in the game by using a new console command
- By running a special compiling command in the editor

To start, let me show you the in-game option.

1. **Press F9 to open the Run Map window, and then click OK to compile and run your map in the game.**
2. **When the game launches with your level, open the Console window, type `mat_leafvis 1`, and then press Enter.**
3. **Close the Console window and take a walk around your map to view the VIS leaves as the player walks through them.**

While you walk around the level, each VIS leaf that you step into becomes outlined in red, like that of Figure 10-17. The leaves might look the same as the BSP leaves that you saw before because they're created automatically using similar algorithms. Soon, I show you how to manage your VIS leaves, which makes the difference between the two leaves apparent.

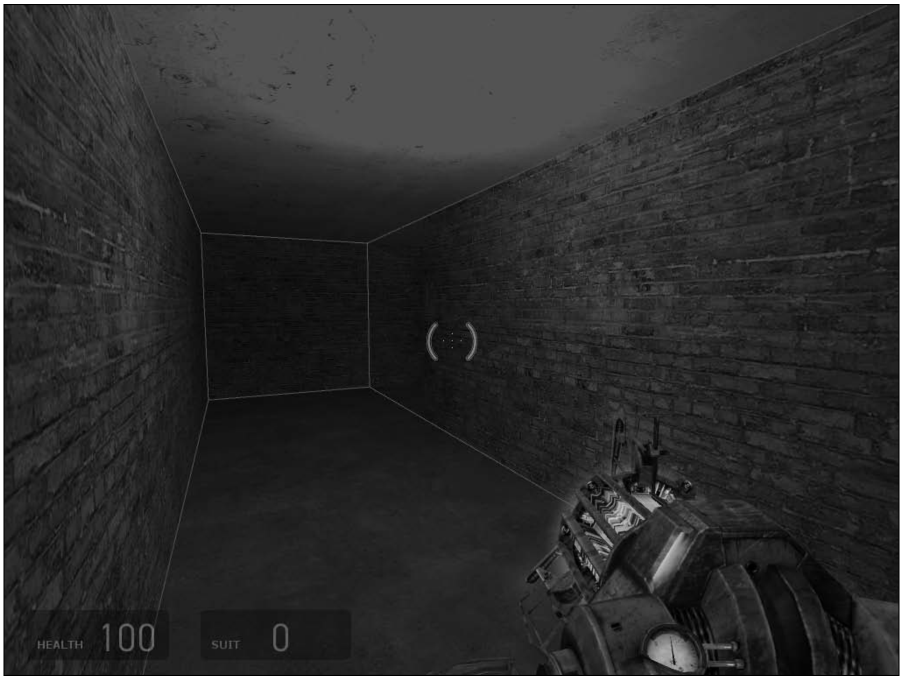


**Figure 10-17:** The console command `mat_ leafvis` outlines the player's current VIS leaf.

## *Viewing your leaves with GLView*

The second option for viewing VIS leaves requires a little additional setup. This option is a separate compiling command that runs within the editor to provide you with a new view of your map. The primary purpose of this new view is to provide you with the visibility information for your level.

1. **Press F9 to open the Run Map window but don't click OK to compile the map just yet.**

   As I mention before, some additional setup is required.

2. **Click Expert to open the advanced compiling options.**
3. **Click Edit to the right of the Configurations drop-down list.**

   This opens a new window, Run Map Configurations, as shown in Figure 10-18. Here, you can manage the different configurations you have available for compiling your map.

4. **Click New to create a new configuration.**

   Another new window opens, prompting you to enter a name for your configuration.

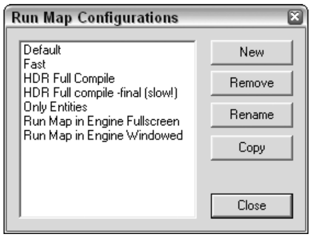


**Figure 10-18:** Manage your compiling configurations here.

5. **Enter GLView and click OK. Then click Close on the Run Map Configurations window.**

   Your new configuration shows up in the Run Map Configurations window. When you see it, you can close this window.

6. **Choose GLView from the Configurations drop-down list.**

   Select your new configuration. After you do, the other options in this window appear blank because nothing has yet been entered in them. Those steps are next.

7. **Click New to the right of the Compile/Run Commands option list.**

   When you click New, a blank entry appears in the Compile/Run Commands list (Figure 10-19). This new entry won't be blank for long.

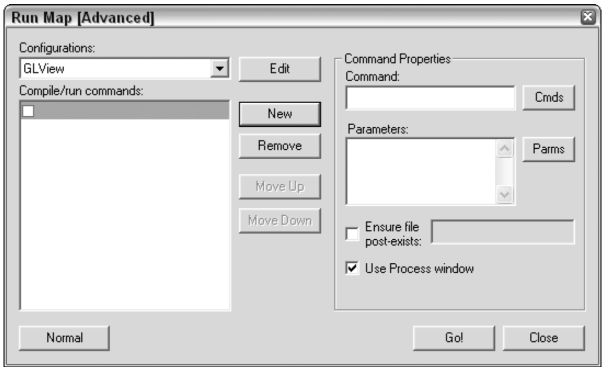


**Figure 10-19:** A blank entry appears when entering a new Compile/Run Command.

8. **Click the Cmds button (on the right side of the window) and choose BSP program from the list.**

   After you choose this, a new command appears in the empty Command box: `$bsp_exe`. This command refers to the first phase of three in the compile process.

9. **Click and place your cursor in the Parameters box and then enter `-glview $path\$file.$ext`.**

   This new parameter tells the compiler to run your map in the BSP compiler with the `-glview` command.

10. **Select the Ensure File Post-Exists option and then enter `$path\$file.gl` in the text box.**

    This checks that the `.gl` file for your compiled map exists before continuing. If the file doesn't exist, using this method to view your VIS leaves doesn't work.

11. **Click New to the right of the Compile/Run Commands option list to create another new command.**

    This configuration requires two commands.

12. **Click the Cmds button (on the right side of the window) and choose Executable from the list.**

    This pops up a file browser window where you can select a specific file to be run. Browse to

    ```
    C:\Program Files\Valve\Steam\SteamApps\username\sourcesdk\bin
    ```

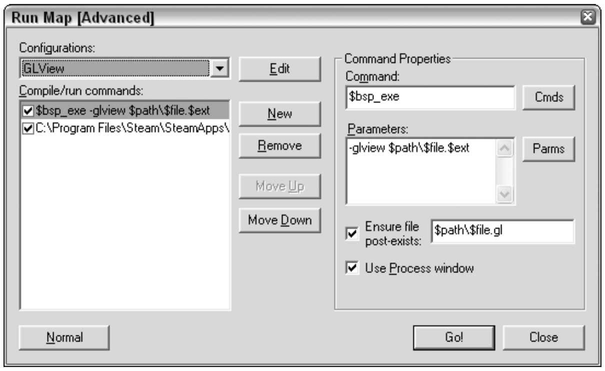
    where *`username`* is your Steam username, and *`Valve`* is optional based on your installation.

13. **Select `glview.exe` and click Open to insert the selected file as your command.**

14. **Click and place your cursor in the Parameters box and enter `-portals $path\$file.gl`.**

    This new parameter tells the compiler to run your map in the glview program with the `-portals` command. This is why the .gl file must exist as specified in the previous compiling command.

15. **Select the check box next to each of your new compile/run commands to enable them (see Figure 10-20).**


Run Map [Advanced]

Configurations: GLView

Compile/run commands:
- ☑ $bsp_exe -glview $path\$file.$ext
- ☑ C:\Program Files\Steam\SteamApps\

Command Properties
Command: $bsp_exe
Parameters: -glview $path\$file.$ext
☑ Ensure file post-exists: $path\$file.gl
☑ Use Process window

**Figure 10-20:** Your new configuration is set and ready to compile your map.

16. **Click the Go! button to compile your map using your new configuration.**

    When you click Go!, your map compiles, and the process appears to be the same. However, when the compiling is done, the game doesn't load for you. This time, the GLView program does load and run your level, and it provides you with some new, valuable information about your map.

In Figure 10-21, the GLView program displays a map in a very basic form. You're looking at the hall from within the first room. All the walls in the map are colored in shades of gray and with some white outlines. You're looking for these white outlines because they define where your VIS leaves exist in your map.

Move around your map by using either the arrow keys or the WASD keys. With your mouse, you can pitch your view to look up and down. Take notice of where each portal is located. If you want, you can even move your camera out of and over the top of your map, as I have in the overhead shot in Figure 10-22.

In Chapter 8, I describe the process of how the game is rendered for the player and how only the leaves that can see each other are drawn. These lessons can now be applied to your map, using the numbers for each VIS leaf in Figure 10-24.

If the player is in Leaf 2, both hallways and all their contents are rendered in the game. I specify the contents of the hall because there could be other objects (like weapons, ammo, or other players) in this leaf that would also be rendered regardless of whether you could actually see them.

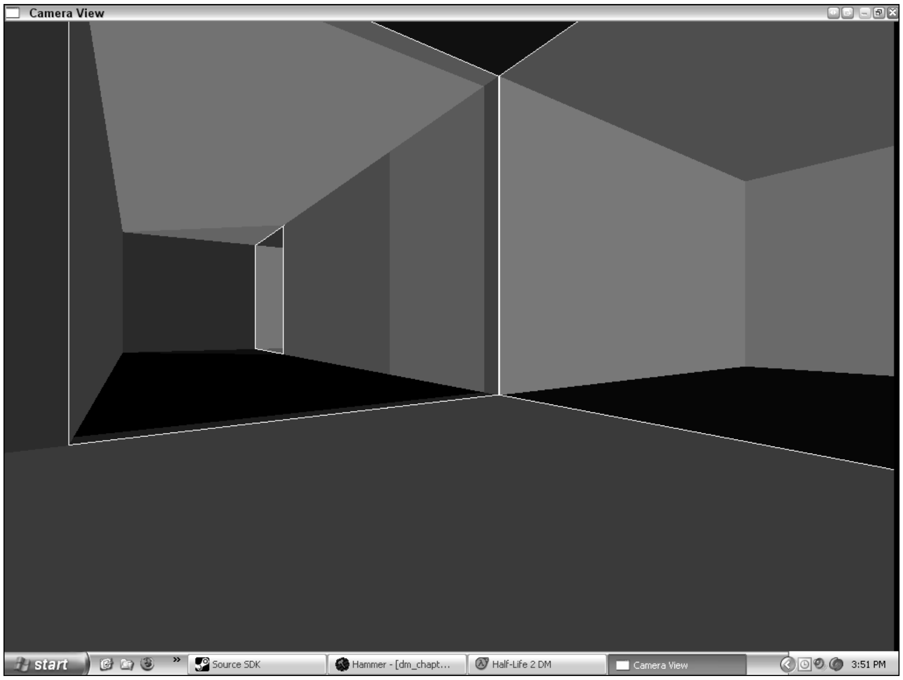


**Figure 10-21:** Your level, as shown within the GLView program.

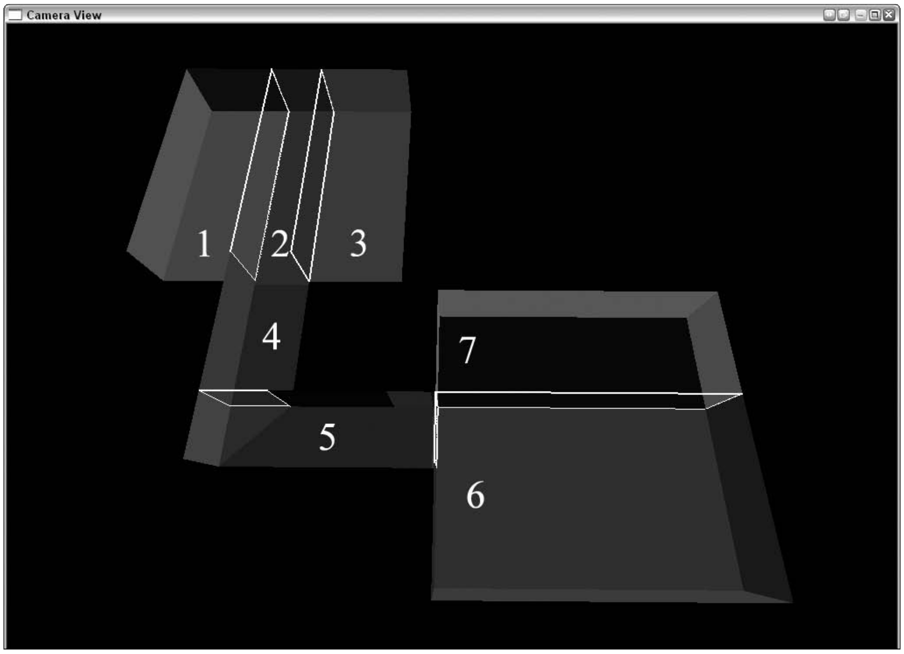


**Figure 10-22:** Each VIS leaf numbered, as shown, from overhead.

If the player is in Leaf 4, all but Leaf 7 is rendered in the game because you can draw a straight line between Leaf 4 and all the other leafs without leaving the confines of the map. A lot of other objects that need to be rendered could be in the map, and this is a waste of your computer resources.

Portals and Hint brushes now come into play. They can be used to create divisions in your map that lead to better placement of your leaves.

Close your GLView by pressing Esc. Exiting any other way could cause a crash in the program, so press Esc.

### Making new leaves

Figure 10-23 shows how you can divide the room in such a way that the two outer rooms can't see each other. Creating a portal on this diagonal effectively splits your map into three separate leaves, where Leaf 1 can't see beyond Leaf 3 and vice versa. Split the one large area into three smaller areas with an angled portal brush. I show you how to put this idea into action, and then you can see the benefit of such practice.

To create a portal, you need to add a brush that completely fills your opening from wall to wall and floor to ceiling. Then texture this brush on one side with the Hint texture — thus why it's sometimes called a *Hint brush* — and the other sides with the toolsskip texture. The toolsskip texture is ignored by the game, and the portal texture divides your map into leaves.

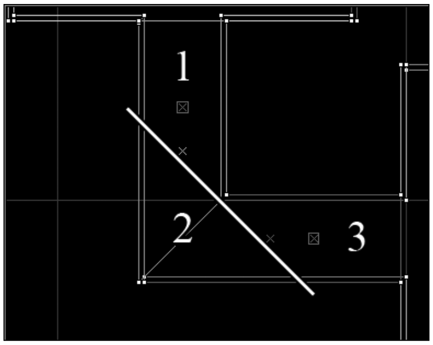


**Figure 10-23:** Splitting one large area into three.

TIP

Make sure that your portal brush creates a tight seal between the walls that it touches. Just like slicing through bread, if you don't slice all the way through, you don't have two separate pieces. You need to create two discrete leaves with your portal.

The most difficult part of this operation is creating the three-sided brush with a diagonal face. To do this, use the Clipping tool to slice a square brush in half on that diagonal. The process is similar to the one used for splitting the wall for the hall, as I discuss earlier in this chapter, except that you do it on a diagonal.

Before you draw your brush, you should locate and select the toolsskip texture. Because all but one side of the brush will be textured with the toolsskip texture, it makes sense to start with this one. To find it, press Browse within the Texture group on the right side of the editor. Then filter for the word *skip* and select the toolsskip texture. It is blue, with the word SKIP, as shown in Figure 10-24.

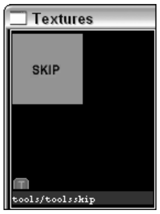


**Figure 10-24:** This skip texture will be overlooked in the game and editor.

You will also want to make sure that your grid is set to 8 units. Because the portal brush needs to fit between the walls of your hall, 8 units is ideal. Check the information bar at the bottom of the editor to see your current grid settings and adjust it if necessary using the [ and ] keys.

1. **With the Nodraw texture loaded, create a brush that fits into the hallway corner.**

   Press Shift+B for the Block tool and then draw your brush. Make sure that it meets your ceiling and floor. This brush needs to meet all four walls of the hallway and create a seal. In the example map, you need to drag the brush to at least 224 × 224 units in the 2D Top view port and 128 units in height, as in Figure 10-25.

   After it's drawn, press Enter to create the brush.

2. **Split the brush on a diagonal with the Clipping tool.**

   Press Shift+X to turn on the Clipping tool, and then place two cutting points over your portal brush. The easiest way to do this is to place your first point on the tip of the inner corner in the hallway and the other point one grid line down and to the right on the diagonal, as shown in Figure 10-26.

   TIP

   In Figure 10-28, you might be surprised that the second point wasn't even placed outside of the brush you're clipping. The clipping line will be created along the points regardless of where you place them. Placing the two points so close together makes it easier to create a 45-degree cut.

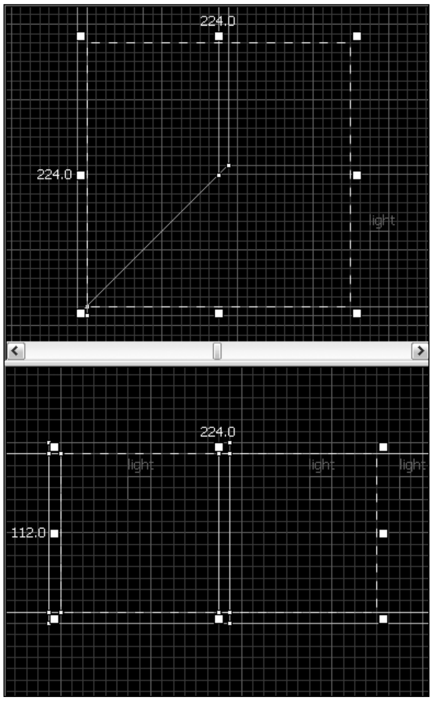


**Figure 10-25:** Create an oversized brush to clip on the diagonal for your portal brush.

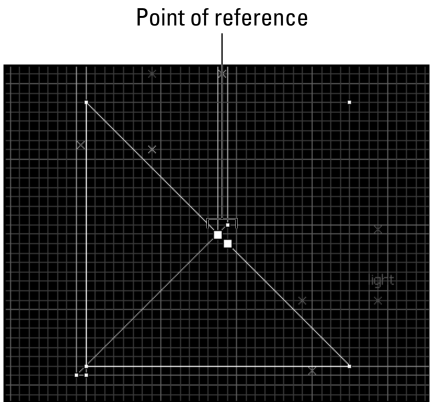


**Figure 10-26:** Your cutting points can be placed anywhere in your map to create the clipping plane.

TIP

If you find that you're cutting off the wrong half of the brush, press Shift+X to change your selection before pressing Enter to activate the cut.

With your clipping selection made and the clipping portion that you want to save is outlined in white, press Enter to cut the brush.

3. **Texture the portal side of the brush with the Hint texture.**

   Open the Texture Application tool by pressing Shift+A. Press Browse to locate and select the Hint texture (see Figure 10-27). It is pink and reads HINT — no other words.

   Right-click the diagonal brush face inside of your map to apply this texture. The result is a brush that has the Hint texture on one side and the Skip texture on all other sides.

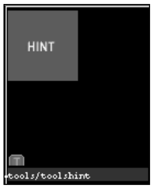


**Figure 10-27:** The Hint texture will hint to the existence of brush and create a new portal.

That's it for the portal that is now creating your new leaves. Because this portal creates a complete seal around all walls and splits the map into three separate areas, you have effectively created two new portals: one portal in each hallway.

Make sure to save the map at this time.

## Can You See Me?

After you better optimize your map, take a look at the benefits.

First, compile your map and view it in GLView. Press F9 to open the Run Map window. If you aren't already in the Advanced window for compiling, click Expert to get there. Make sure your new GLView configuration is selected. Then press Go! to compile and view the map.

Looking at your map this time from the top view as shown in Figure 10-28, you can see more leaves. The number of leaves in your map isn't important. What is important is that the leaves are arranged so that your computer doesn't have to work at rendering an area of the map that your player can't actually see.

WARNING!

You can have too many VIS leafs. If you divide your map into as many leaves as possible, you do yourself more harm then good. The compile times increase dramatically, and the game might actually start to slow down. Each leaf adds data to the map that must be considered by the game when running. Simply, place leaves where you need them.

Close the GLView now by pressing Esc. This time, compile the map and run it in the game. You can see how the map is affected after the placement of your new portals.

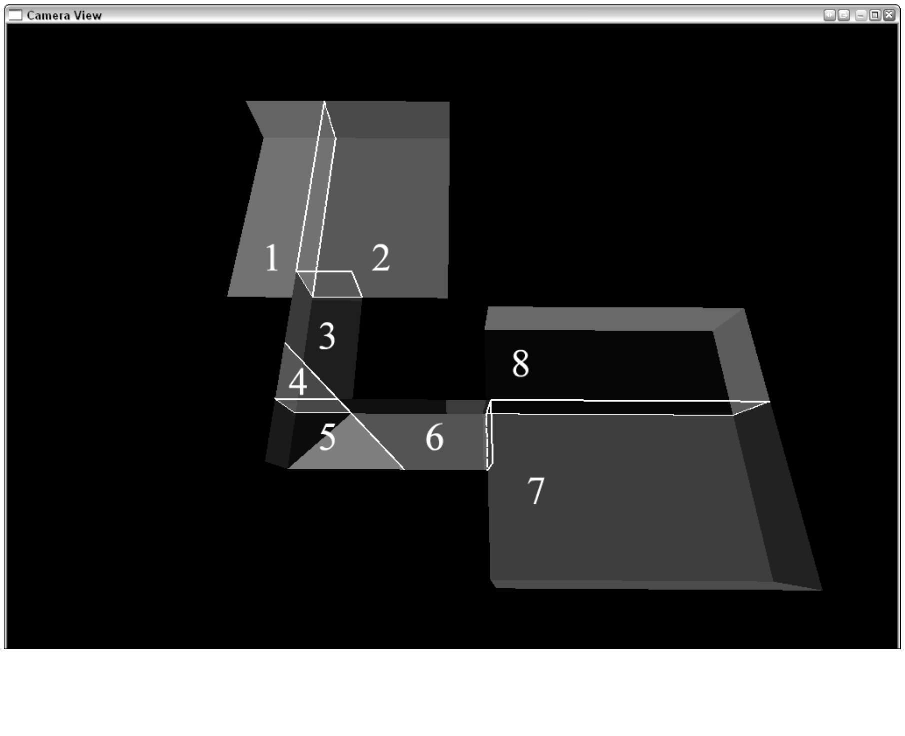


**Figure 10-28:** GLView shows you that your leaves are well placed.

Press F9 for the Run Map window and switch to the Normal view if you aren't already there. Make sure that all your compile commands are set to Normal and that you still have the additional game parameters entered as `-dev -console`. Then press OK to compile the map and launch it in the game.

When you enter the game, open the console by pressing ~. In the console, enter the command to view your VIS leafs — `mat_leafvis 1`. Then press Enter and close the console. Walking around your map, you can see the VIS leaves outline in red as your player steps into each leaf similar Figure 10-17. The leaves appear exactly as you saw them in the GLView program.

TIP

If you see additional splits in the hallway of your level, you need to go back and fix them. Your textures might not have been properly places, or you might have a gap between your Hint brush and one of the other brushes that's causing a leak.

Open the console again and type **mat_wireframe 3**. Viewing your map shows you that your Hint brushes didn't create any additional geometry in the game, which is great (see Figure 10-29). All you did was affect what is rendered, which will be of great benefit later when you start to fill in the map with details and additional objects.

Save your map again, and you're done. After the basics of optimization you're ready to move on to bigger and better things.

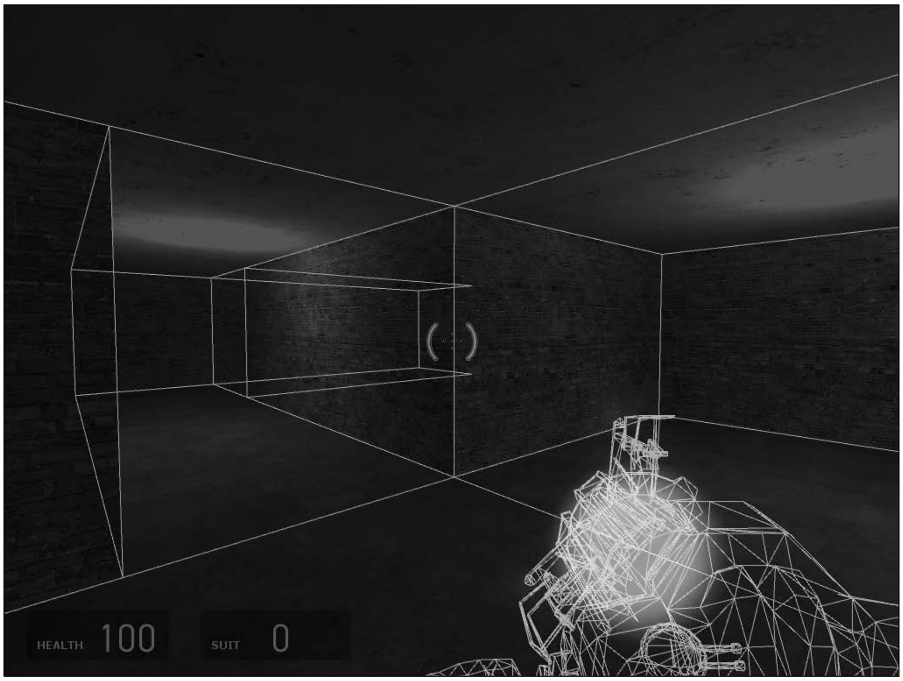


**Figure 10-29:** Only rendering (not geometry) was affected by your changes.

# Chapter 11

# Heading to the Great Outdoors

*In This Chapter*

- ▶ Adding an outdoor area
- ▶ Texturing the outdoors
- ▶ Creating doors between areas
- ▶ Using special lighting techniques
- ▶ Setting up the doors to move for the player

The preceding chapters help you get a well-optimized indoor map under way with a couple rooms and a hallway connecting them. However, this map is still pretty small. Continuing the expansion, here I show you how to head outdoors and work on a different type of environment.

This chapter shows you how to apply special textures to make the addition look like the outside world. Then you add some special lighting effects to complete the feel of your map. You need a way to get inside and out, so you install a doorway and an actual working door for your player to interact with.

## Building an Outdoor Addition

Building an outdoor addition is structurally the same as working indoors. You create a sealed area that extends from your current work, so you need walls, a floor, and a ceiling. However, instead of using concrete or brick textures on this outside area, you must apply textures that display a sky, dirt, and other outdoor ambiance.

Although building an outdoor environment is simple, plan a few important details before you begin. Structurally, an outdoor area is a box with a texture on the floor resembling grass or dirt; the rest of the brushes are textured with a sky texture. However, when the player approaches the edge of the box, nothing visually signals the edge of the map. The player just finds himself on the brink of nothing, unable to move forward. Figure 11-1 is an example of what a map looks like when the player can see over its edge. Because this can be confusing, you need something here that tells the player, "You can't go beyond this point."

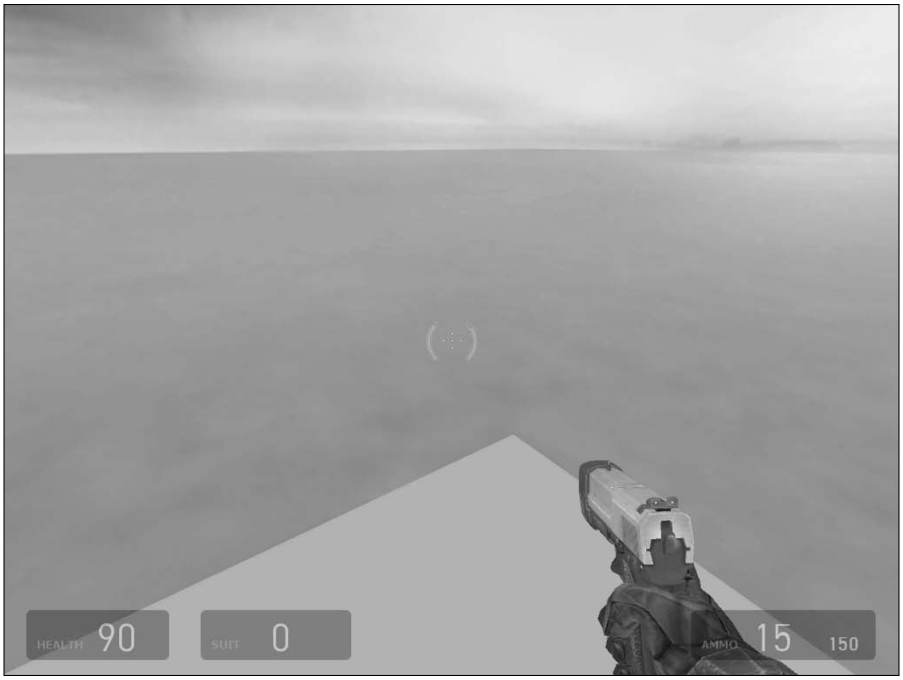


**Figure 11-1:** The edge of the map where sky meets structure.

To define the edge of the map, you need to add a tall wall. The wall extends all the way around the outside area and is tall enough that the player can't see over the edge. The rest of the box is textured with the sky texture.

When building your sky, its height needs some consideration. You need the sky to be tall enough that whatever is thrown at it — such as a grenade, a body, or whatever — doesn't hit and bounce off the ceiling of your sky box. That would look bad in the game. However, building the area larger than it needs to be is a waste of compile time. In Chapter 8, I explain the compile process and how the third process of compiling is that of light. The more area the light covers, the more processing time the map requires. Perhaps the extra processing time won't make too much of a difference in your small map, but if you start considering it now, you'll be in good shape for larger maps.

Here's what you need to do to begin work on the outside area in the map:

1. **Load `dm_chapter10.vmf` in the editor.**
2. **Make sure to have the Nodraw texture selected as your base texture.**
3. **Draw a large brush that positions the current structure in the lower-left corner, as shown in Figure 11-2.**

   After selecting the Block tool, I drew a box that is 1472 × 1488 × 320 units in size on the X, Y, and Z axes. The left and bottom side of this box should be 8 units away from the walls of your building, and the bottom of the box should be on the same plane as your floors. This creates an area large enough to feel like it's outdoors and to encompass the current structure. Also, there is enough room to account for the 8-unit walls on the sides that will be created when the brush is hollowed. You now have the basic shape that you can work from, but not the final box.

After the box is drawn, don't forget to press Enter to create the box in the editor.

4. **Use the Hollow function to make the solid brush into a room for your player.**

   Press Ctrl+H to open the Hollow option window. Enter 8 as the wall thickness for your hollowed room and then press OK.

5. **Select your new box and press Ctrl+U to ungroup the brushes.**

   You need to make some adjustments to these brushes coming up next, and that's easier to do when they're not all grouped.

   You should have a nicely hollowed box surrounding your map.

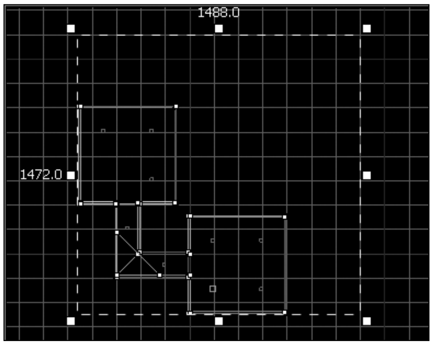


**Figure 11-2:** The outside area should be large enough to encompass your indoor area.

## Multiplying the ground

As it sits now, the ground brush overlaps the floors of your building. You need to fix that, or the game will display errors as it tries to figure out which texture it should display. The best solution is to split the floor into pieces that fit the area you need to cover:

1. **Select the ground brush and adjust it to fit in the upper-right corner of the map between the buildings.**

   The brush must extend under the walls that surround your outside area to keep the seal you have in your map but also must butt against the brushes in your building so as not to overlap (see Figure 11-3).

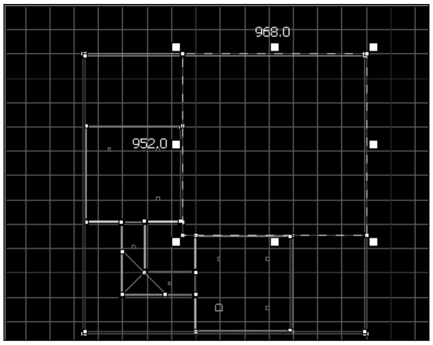


**Figure 11-3:** Adjust the floor brush so it fits tightly into the upper right of the map.

2. **From the Top 2D view port, use the clone function to fill the missing pieces of ground by duplicating the selected brush and moving it.**

   You're welcome to simply draw new brushes for each of the ground pieces, but I find the clone function easier to use in this situation. Select the brush you want to duplicate, Shift+click that brush and drag it to a new location in the map.

3. **Adjust that copy, which is now selected, to fit into one of the gaps in the ground.**

   Continue doing this until you complete the ground for your map. Use Figure 11-4 as a guide to placing those brushes. Use duplicates to fill in the missing ground.

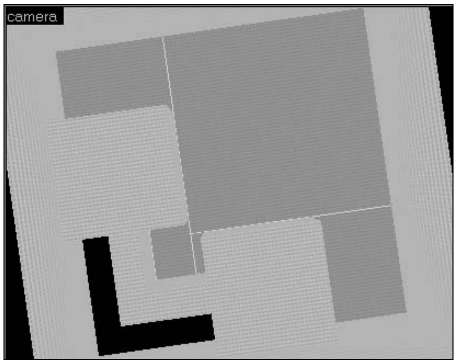


**Figure 11-4:** Adjust the ground brush so that it doesn't overlap the floor brushes.

## Filling in the useless corner

Looking at the top view of the map, you see a nice, large, outdoor area for your players. However, you also see a small area in the bottom left that isn't exactly playable. This small area is blocked by the L-shaped hallway connecting your two rooms. Because you won't use this area in your map, exclude it by adjusting the brushes in the map and filling in that corner as detailed in the next few steps.

To start, the two walls that meet in the lower-left corner need to be shortened. In case your players can see the tops of your buildings, you should leave these tops inside the map. Remove only the empty space from that bottom-left corner.

Stretch the brushes of the walls so that they meet up with the backs of the buildings. When you're done, it should look something like Figure 11-5. In this figure, you are looking at the hallway from within the outside area. Notice how the two walls on the left and right (representing the lower and left walls in the 2D window of the editor) extend to the length of the buildings and then stop. Later, when you're done manipulating the ground brush, you can fill in that corner.

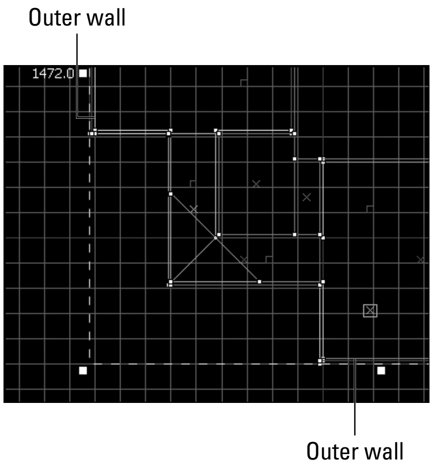


**Figure 11-5:** Adjust the brushes in the lower left of the outside area before closing that corner.

As for the hole you now have in the corner, you could draw in a few more wall brushes to close the gap, but here is a way to save yourself some work. Instead, draw two larger blocks that completely fill in that corner. Each new block needs to touch the ceiling brush of the sky and extend only to about 8 units below the tops of your buildings. (There's no point in extending the brushes below the tops of your inner rooms because the player cannot access this area, and you created the necessary seal.) The result looks something like Figure 11-6.

Using a solid brush in this case doesn't hurt your map in any way. Because it's solid, no light calculations are within it. The compiler sees it only as a fat brush that seals the inside of your level, preventing leaking.

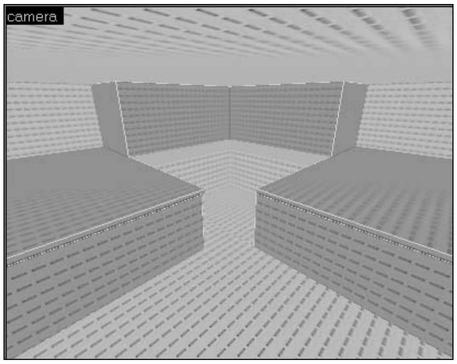


**Figure 11-6:** Fill in the hole in the lower-left corner with two large, solid brushes.

## Defining your boundaries

You want to apply two different textures to the outside walls to complete your scene. First, you want to texture half the outer brush faces with something that looks like a solid containment wall. This gives the player a sense that the level stops here. The rest of the outside area gets a sky texture.

You can get two textures on your walls by splitting the walls — the top half being sky, and the bottom half being solid wall. Here's how:

1. **Make sure you view your map in either the 2D Front or Side viewport.**
2. **Select all the brushes that make up the surrounding wall. Remember to select those blocks in the southwest corner of the map because your solid wall will reach above that point.**

   For the example in this book, you'll be making the wall 160 units tall, which is slightly higher than your buildings. A wall that's 160 units is a good height; it's tall enough that your player won't be able to see over the top when playing in the game.
3. **With all those brushes selected, use the Clipping tool to split them all at once at the 160-unit line on the Z axis.**

   Press Shift+X to turn on the Clipping tool. Click and place the first clipping point within the 2D viewport on the 160 Z axis and anywhere on the other axis.
4. **Click and drag your first point to place the second clipping point within the same window on the 160 Z axis and anywhere on the other axis.**

   This displays a horizontal line that represents your clipping line.
5. **Confirm that both halves of your clipping plane are outlined in white to be split. Otherwise, continue pressing Shift+X until they are.**

6. **Press Enter to split your selection, leaving you with top and bottom halves of your walls.**

   The result should look similar to Figure 11-7.

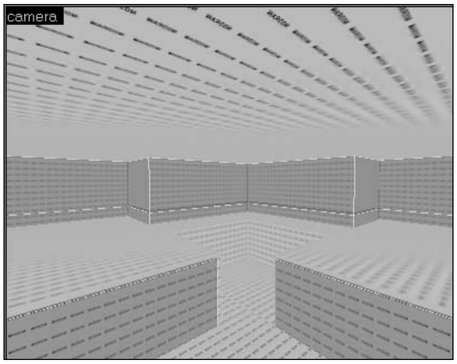


**Figure 11-7:** Split the walls of your outside area to define where the outside walls meet the sky.

## Making a Scene

It would be nice to test your map and see how it plays in the game. However, you still have some very important brush faces to texture. You haven't textured your outside walls and sky. Without it, the leak error will result in your map during the compile process, and you won't be able to load it in the game. Presently, everything in the outside area is covered with the Nodraw texture. The following sections show you how to dress things up with some proper textures, starting with the ground.

### Throwing dirt on the ground

Because this is an outdoor area, select an appropriate texture for your ground. Here's how:

1. **Press Shift+A to open the Face Edit Sheet window.**
2. **Click Browse to open the list of available textures for selection.**
3. **Enter the word** dirt **into the Filter to locate all the textures that would be best suited for this outside ground and select the texture `nature/blenddirtgrass001a`, as shown in Figure 11-8.**

   REMEMBER

   To select the texture, double-click it within this view. The viewer window closes, and your texture is selected.
4. **Move around your map within the Camera viewport and apply that texture to the brush faces on the ground by right-clicking.**

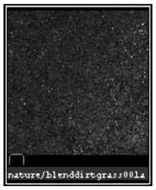


**Figure 11-8:** Apply a natural texture to the ground of your outdoor area.

## Fitting the outer building

With your ground taken care of, move onto the exterior of the buildings. Because the building's interior has a brick texture, select something similar for the outside.

When you created the texture for the interior of your buildings, you found your interior wall texture by filtering the textures with the word *brick.* You find a similar texture for the outside by using the same filter word. Here's how:

1. **From within the Face Edit Sheet window, press Browse and filter your textures with the same word, *brick.***
2. **Select the `brick/brickwall031d` texture, shown in Figure 11-9.**

   This texture is very similar to the one you used for the interior except that it has a concrete border on the top edge to help to make the rooftop look better from the outside.

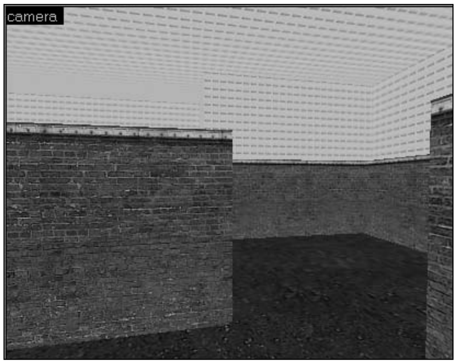


**Figure 11-9:** Apply a brick texture to the building exterior.

3. **Apply this new texture to the brush faces on the exterior of the two rooms. Right-click within the Camera viewport to apply the selected texture.**

After you apply your texture to the exterior walls, some corners appear to need mitering. It is not required that you miter these exterior corners as long as you apply the same texture to all the brush faces on all adjoining walls. The compiler will do the optimization work for you. As I describe in Chapter 10, the compiler merges brush faces that have the same texture with the same values applied to those brush faces. However, I personally prefer to miter all of my corners so that I am confident that the brushes are optimized regardless of texturing. This mitering step is described in the section, "Mitering the Outer Walls," later in this chapter.

4. **Go through the rest of the level and continue to apply this wall texture. Remember to apply it to all the wall surfaces of the building, including the thin strip at the top of the wall.**

If you change textures and then go back to reapply some of the wall textures, copy the texture from the adjoining wall. Otherwise, your wall might appear to have a seam where the two textures meet, and you also end up with an additional split in the wall. To copy the texture, select it from one of the surrounding walls. Then you can continue to reapply it with those same texture settings.

## Climbing the roof

After the walls are done, texture the rooftops. Your player won't access this spot in the map, so the texture that you use isn't important. However, because the player might see it when he jumps, select something that fits the scene.

Browse your textures and filter with the word *roof.* This turns up a few good textures for your map. Specifically, I like the texture `props/rubberroof002a`. Another texture, `props/tarpaperroof002a`, is also nice, but it will look very repetitive because of its design.

Select and apply the `props/rubberroof002a` texture, as shown in Figure 11-10. Make sure to apply it to all rooftop surfaces.

**Figure 11-10:** Apply a texture to the rooftops of the buildings.

## Walling in the yard and adding sky

Move on to the outer wall, applying two textures here:

- For the lower half of the wall, something solid and strong looking
- For the sky that appears over the wall

1. **Browse your textures and filter your list with the word *stonewall.***

   You can certainly use other filters, but this one helps you locate the texture I had in mind.

2. **Select the texture `stone/stonewall006a`, as shown in Figure 11-11.**

   This texture has a look that fits everything else done to this point.

**Figure 11-11:** Apply a stone texture to the walls surrounding the outdoor area.

3. **Apply this texture to the lower wall section that surrounds the interior of your map.**

   When you apply this texture, it begins to repeat itself vertically. Don't worry about this repetition. The texture doesn't have to fit, and you can work the overlap into the scene later.

With the solid walls covered, move onto the sky. Adding a sky that looks like it stretches into infinity is really a lot easier than you might think. It's nothing more than applying the right texture to the brushes. The game does the rest of the work for you.

1. **Browse your texture and filter the list with the word *sky.***
2. **Look through the textures and select the `tools/toolsskybox` texture, which looks like Figure 11-12.**

This blue texture bears the word SKYBOX written on it and doesn't really look anything like a sky, at least just yet. The game knows to apply the sky to this texture so it looks vast within the game.

3. **Apply this texture through the rest of the interior of your map.**

   When you're done, you should no longer be able to see any Nodraw texture from inside the playable area. The outside of the map should still be textured in Nodraw, but the player will never see that.

4. **Save what you have so far under a new name (`dm_chapter11.vmf`).**

**Figure 11-12:** The sky texture creates the illusion of a big sky.

TIP: Keep versions of your map to fall back on if a problem arises because of an unexpected mistake or error.

## Mitering the Outer Walls

When you optimize the interior of your rooms and hall in Chapter 10, I show you how to miter the corners that the player sees. At that time, the player could see only the interior of those three sections. However, now the player can walk around outside these rooms, and the corners outside haven't been mitered for optimization.

TIP: Let me state again that although brushes with matching textures will automatically optimize themselves during the compile process, I recommend that you miter your corners. Mitering is a good habit to get into as it ensures that your map is properly optimized.

Make your way around the map and miter the corners around the rooms. Six corners need your attention, as I point out in Figure 11-13.

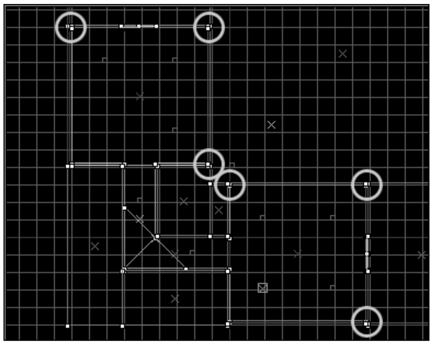


**Figure 11-13:** Don't forget to miter the outside corners now that they can be seen.

## Getting Outside

You've added a great set of brushes for your outdoor scene, but how do you get to it? Right now, your player will spawn inside one of the rooms but can't get beyond the two rooms and the hallway. You need to cut a set of doors for your player to use. In the next few sections, though, you make two sets of doors rather than one.

You have a number of ways to cut in your first set of doors.

- Create an entrance by stretching and duplicating brushes (just as you do to the ground of your outdoor area earlier, in the section "Multiplying the ground").
- Shrink the existing walls and draw new ones to replace the missing pieces.
- Use the Clipping tool to split the existing brushes, thereby making a doorway.

The last option of splitting your wall brush is the simplest method because it won't affect the texturing you applied to the interior and exterior of the rooms. If you move brushes around, the textures won't line up as they did before, which creates a visual seam where the walls are joined — as well as added geometry.

### Clipping out some doors

Here's what you need to do to get started creating your doorways. Use the Clipping tool, with Figure 11-14 as a guide:

1. **Begin by selecting the East wall on the lowermost room in the map.**
2. **Press Shift+X to turn on the Clipping tool and then start clipping the center of the wall for a door.**

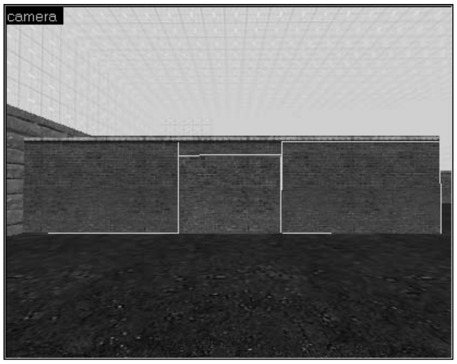


**Figure 11-14:** Use the Clipping tool to create the brushes for your doors and doorway.

You can use either the Top or Front viewport to perform the clipping. I recommend using the Front viewport. From here, you can see where your hallway is placed in relation to the door and center the two exactly.

3. **Make the first cut to the height of the wall on the Z axis on the –120-unit grid line.**

   Press Enter to initiate the cut on your clipping line.

   REMEMBER

   When clipping, split the brushes. After placing your points for clipping, make sure that both halves of your selected brush are outlined in white. This indicates that the brush will be split to give you halves. Press Shift+X to cycle through your different clipping options.

4. **Make the next cut on the 8-unit grid line of the Z axis.**

   This makes for a pretty large door, but that's okay. You're going to turn these into double doors, so 128 units wide for your doorway is ideal.

   The next cut defines the height of the door. A good door height in this game is about 96 units.

5. **Select the center brush only so as not to split the walls on either side of the door in the next step.**

   Press Shift+S to switch back to the selection tool and click the center brush within the Camera viewport. The switch back to the Clipping tool by pressing Shift+X.

6. **Split the selected brush on the Y axis on the 104-unit grid line.**

   Because the top of the floor is on the 8-unit Y axis, your split should occur 96 units above that. This is the 104-unit grid line.

7. **Select only the door brush and split it down the center to create the two doors (see Figure 11-15).**

   With these two doors still selected, you need to make them look more like doors rather than part of the wall, so make the doors more narrow than the adjoining walls. This also helps with the functionality of the doors. Because these will be sliding doors, it makes sense that they are thinner than the walls they slide into.

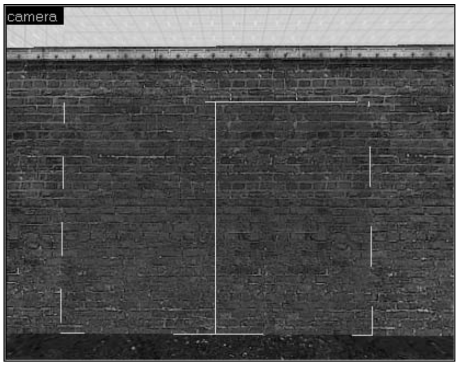


**Figure 11-15:** Clip your two doors to be 64 x 96 units each.

8. **Center your Top viewport over your two doors so you have a close-up view of them.**
9. **Change the grid size and make it two sizes smaller, equal to 2 units, pressing [ twice.**

   TIP

   If you can't see the grid lines in the Top viewport, zoom in on your doors until you can. You need to see these lines to make the next adjustment.
10. **Adjust both brushes simultaneously so that they are equally 2 units thinner on each side, as shown in Figure 11-16.**

    To resize the doors, grab one of the handles (white boxes) of the sides and corners of your selection. Then click and drag one of these handles to adjust its size.

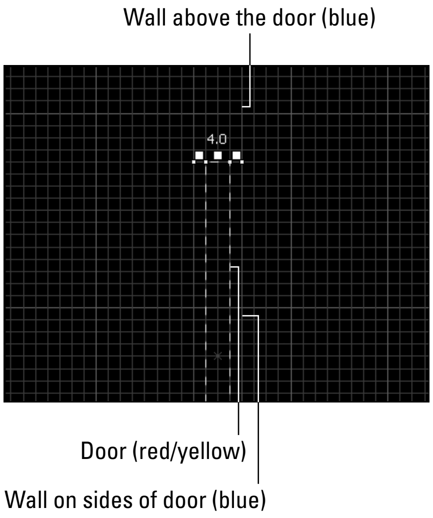


**Figure 11-16:** Change the door thickness to make them stand out to the player.

11. **Return the grid size back to 8 units (Grid8) by pressing ] twice.**

    You don't want to accidentally make changes to your map that you cannot see. When you zoom out, you're won't going to be able to see the 2-unit grid lines, and you could end up doing something that results in a leak of your map.

## Touching up the textures

The following sections describe a few examples of places where you might want to touch up the textures. For example, door jams are notorious for needing touch-ups. I can't think of any place where you would see doors that slide directly into brick. I can, however, think of doors that slide into metal frames that are attached to brick walls. The solution is to create a metal frame around the doors.

### Working around doors

First, the doors you created are in your way. You could try to work around them to apply the textures where they're needed, but here's an easier way: Temporarily hide the doors so that you can texture what you need, and then unhide them when you're done.

1. **Select your two doors.**
2. **Choose View⇨Hide Selected Objects.**

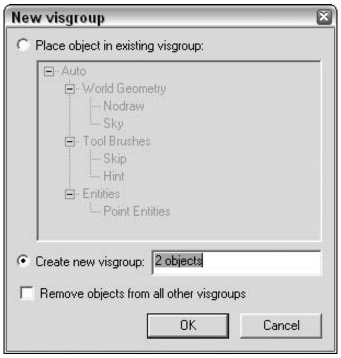
   TIP

   A new window opens, asking you to create a new visgroup like Figure 11-17.

   A *visgroup* is a group of objects that you can choose to hide or show within the editor while you work.
3. **Enter a name for your new visgroup; name this group Doors 1.**

   REMEMBER

   Name each visgroup so that telling them apart is easy. In this case, you can name the other set Doors 2.



[✓ Create new visgroup:] 2 objects

[ ] Remove objects from all other visgroups

**Figure 11-17:** Create groups of selected objects that can be hidden within the editor.

Do not select the check box here. If you do, this removes your selected doors from any other visgroup that exists. This is not necessary for your current selection.

4. **Press OK to create your visgroup and hide your doors.**

   Your doors disappear from view, like in Figure 11-18. However, they are available to you at any time you want to show them again. If you look at the VisGroups group on the right of the editor, you see your new group listed there — unchecked. Later, after making your texture adjustments, you select that check box next to your group to show them again in your map.

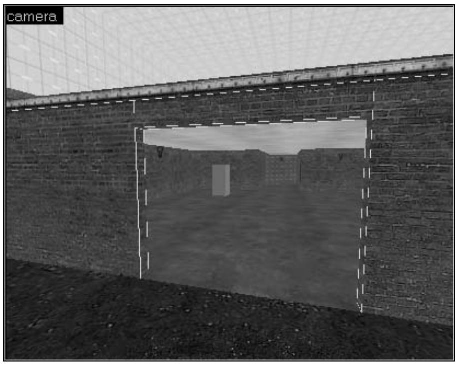


**Figure 11-18:** Unselected visgroups are hidden from view but remain in your map.

### *Creating a door frame*

After your doors are hidden from view, create that metal door frame. Here's how:

1. **Select the Nodraw texture from within the Texture group on the right side of the editor.**
2. **Activate the Block tool by pressing Shift+B and then draw a thin brush on one side of the doorway.**

   The brush should stick out from the doorway by 2 units and extend from the floor to the top of the doorway without overlapping any other brushes, as you can see in Figure 11-19.

3. **Switch back to the Selection tool by pressing Shift+S. Select your new brush and then click and drag it to the other side of the doorway.**

   This creates a duplicate of the first brush that you can place on the opposite side.

4. **For the top of the doorway, switch back again to the Block tool by pressing Shift+B. Then draw another brush on the top of your door.**

   Figure 11-20 is an example of what you should end up with.

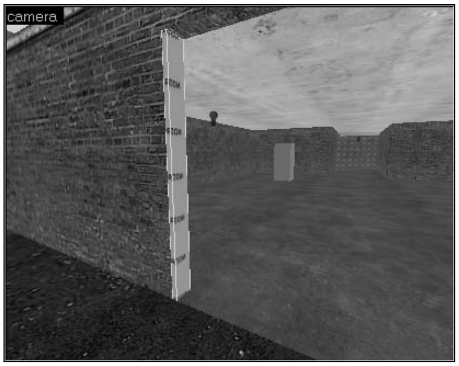


**Figure 11-19:** Create a thin brush along one side of the doorway.

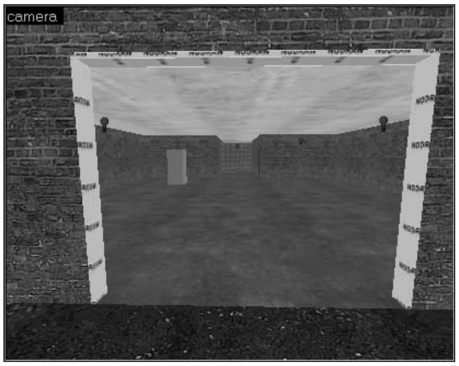


**Figure 11-20:** A door jam ready for texturing.

### *Adding texture to the door frame*

Texture these new brushes with a metal texture:

1. **Open the Texture Application tool and browse for a new texture.**
2. **Filter your texture with the word *metalwall*.**
3. **Locate and select the `metal/metalwall031a` texture, as shown in Figure 11-21.**

metal/metalwall031a

**Figure 11-21:** Select a metal texture suitable for a metal frame.

4. **Right-click and apply this texture to the visible sides of the brushes in your doorway.**

### Applying a new texture for the doors

From the VisGroup window on the right side of the editor, select the check box next to your group titled Doors 1. Your doors will unhide, and you can move on to retexturing them. Here's how:

1. **Browse for a new texture that will look good on your doors.**

   TIP: Filter your textures with the word *door* to make one easier to find.

2. **Locate and select the texture `metal/metaldoor013a` (see Figure 11-22).**

   You could select a texture that looks like a single door, but in this case, you could instead apply a texture that already looks like two doors.

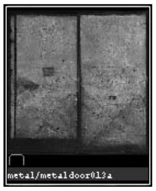


**Figure 11-22:** Select a metal texture for your doors.

3. **Right-click the door on your left to apply the new texture.**

   When you apply the new texture, it appears to not line up properly, as you can see in Figure 11-23. To resolve this issue, you can

   - Manually adjust the texture's location by shifting it with the Texture shift values at the top of the Face Edit Sheet window.
   - Use the Justify buttons on the right side of the Face Edit Sheet window. This is my choice.

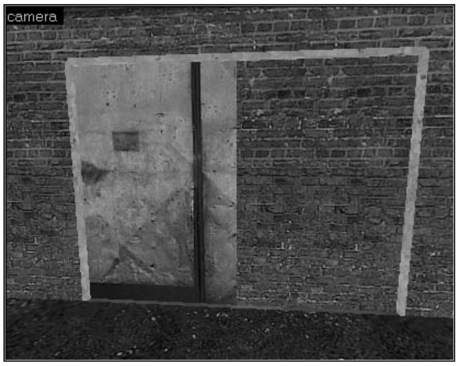


**Figure 11-23:** The door texture doesn't line up upon application and needs adjustment.

4. **Apply the texture to the door on the right so that they are both textured with the same texture.**
5. **Select both brush faces on the doors by Shift+Ctrl+clicking each.**

   You might not see that the door faces are selected, but they are.

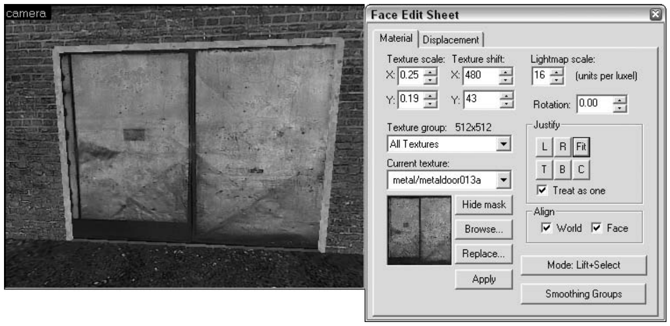
   TIP

   When selecting brushes faces with the Texture Application tool active, you might not see your selection. This is a visual issue that can sometimes occur. If you move your view around within the Camera view port, your selection shows up with a red hue, indicating that your selection was made. After gaining confidence in the editor and tools, you can feel secure knowing that your selection was made without having to rely on this visual to confirm it.
6. **Select the option to Treat as One.**

   The Treat as One option, as shown in Figure 11-24, applies any changes you make to all the selected brush faces as if they were one, large brush. This way, you can apply changes to your doors as a whole rather than having to work with them individually.
7. **Click the Fit button to fit the texture to your selection.**

   The Fit button, also shown in Figure 11-24, is in the Face Edit Sheet window and listed as one of the Justify options. This option fits the texture so that its height and width equal that of your selection. This is perfect for making your texture fit across the two doors.

   Your doors should look perfectly textured, as in Figure 11-24. However, you've textured only one side.



Texture scale: X: 0.25 Y: 0.19
Texture shift: X: 480 Y: 43
Lightmap scale: 16 (units per luxel)
Rotation: 0.00
Texture group: 512x512
All Textures
Current texture: metal/metaldoor013a
Justify
☑ Treat as one
Align
☑ World ☑ Face

**Figure 11-24:** Adjust both door textures at the same time for a perfect fit.

### Touching up the interior textures

Move your view into the interior of the room where your doors are located. Your door texture should still be selected as the current texture within the Face Edit Sheet window, and it is ready to be applied to the other side of the doors. If you closed the Face Edit Sheet window, go back to the outside area and reselect the texture on the outside of the doors.

On the inside of the doors, right-click the door faces to apply your texture. The texture applies itself in reverse as a mirror of what you applied earlier. Only one more application of your texture remains: namely, the portion of the doors that the player can see when the doors are open.

Reaching the area where the two doors meet might look difficult because they are against each other. However, nothing is stopping you from moving those doors into an open area where you can apply your texture, and then moving them back again. Here's how:

1. **Switch to the Selection tool by pressing Shift+S.**
2. **Select one of the two doors and move that door to an open area.**

   Here, you can adjust your view in the Camera viewport so that you can see the edge of the door that needs texturing.
3. **Open the Texture Application tool by pressing Shift+A.**
4. **Select the door texture from one of your door brush faces with a left click.**
5. **Apply this texture to the door edge with a right click, as in Figure 11-25.**
6. **Do the same for the other door and then move the doors back into place where they came from.**

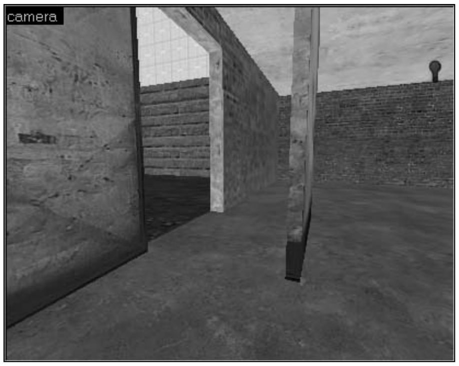


**Figure 11-25:** Don't forget to texture the edges of the doors that will be exposed later.

## Fixing a Bottleneck

Having multiple players in two areas divided by a single, small door can cause problems. If they all try to run through that door at the same time, they will get stuck. In addition, the volume and predictability of players moving through this area can make the game predictable. This is a *bottleneck.* To solve this issue, you either need to make the door larger or create a second door. The latter suggestion usually makes for a more interesting level in the game.

Right now, your map has two major areas divided by a bottleneck. You have an indoor area and an outdoor area split by a single doorway. To make the level flow better and to create more player options, you'd be better off placing a second set of doors in the map in a different location.

To avoid having a bottleneck situation in your map, add a second set of doors to the north wall of the north room. This makes it easier for your players to run between the inner and outer areas of the level, and the additional choice allows for more strategy by the players.

Select the north wall of the second room. Follow the direction outlined earlier in the earlier section, "Clipping out some doors." After you create your second set of doors, create the metal door jams that surround the inner doorway. Finally, texture the brush faces by following the instructions in the section "Touching up the textures." You should end up with what you see in Figure 11-26.

You've made a lot of changes at this point. So that you don't lose any of your hard work that was put into this map, save your progress (press Ctrl+S).

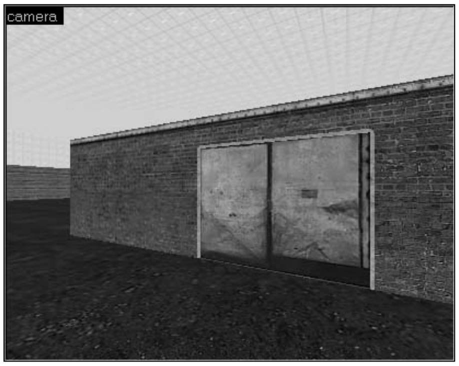


**Figure 11-26:** Create a second, identical set of doors on the northern wall of the north room.

## Sliding Open the Door

What's missing from your map at this point is a way to get through your doors. Right now, they're just solid brushes. The only differences between them and the walls around them are the textures. The doors should open for the players. Because these will be interactive elements of the game, you need to turn them into entities. In this case, door entities will suffice.

## Making that door move

Here's what you need to do to get the doors to move:

1. **Move the camera over to one of the door sets in the map.**
2. **Select one of the two doors in the set.**
3. **From within the New Objects group on the right side of the editor, choose toEntity.**

   This converts your solid brush into an entity that you can now define with specific properties. These properties are defined within the Object Properties window that comes up when you press the button (see Figure 11-27).
4. **Choose func_door from the drop-down list of classes.**

   The default class is func_detail. You need to change this to func_door by choosing it from the list.
5. **Name your door door_blue.**

   Select a name from the Keyvalues field and enter the name of your door in the empty text box to the right (see Figure 11-28).

   Later, you need to call to this door by name, so give it a name that you can use. In this case, I name this door and the door next to it both door_blue because they are adjoined with the room containing blue lights.

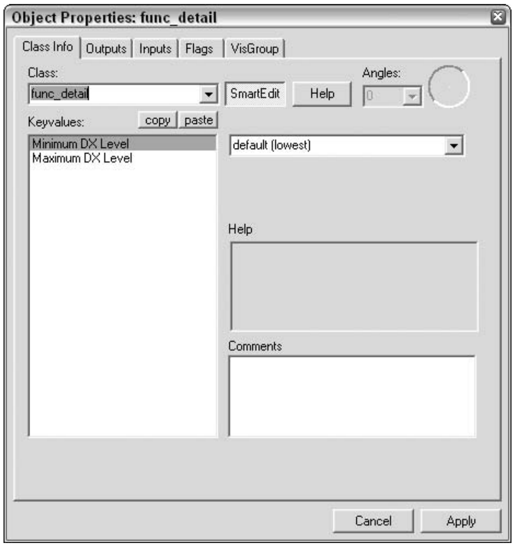
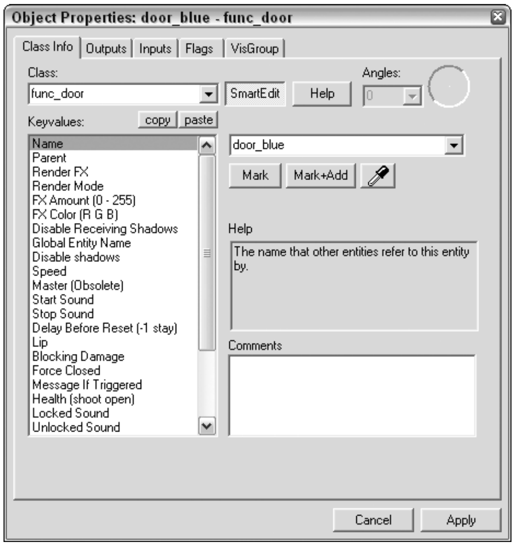


**Figure 11-27:** You can turn a regular world brush into an entity, such as a door.


Object Properties: door_blue - func_door

Class Info | Outputs | Inputs | Flags | VisGroup

Class: func_door
Angles: 0

Keyvalues:

Name
Parent
Render FX
Render Mode
FX Amount (0 - 255)
FX Color (R G B)
Disable Receiving Shadows
Global Entity Name
Disable shadows
Speed
Master (Obsolete)
Start Sound
Stop Sound
Delay Before Reset (-1 stay)
Lip
Blocking Damage
Force Closed
Message If Triggered
Health (shoot open)
Locked Sound
Unlocked Sound

door_blue

Help

The name that other entities refer to this entity by.

Comments

**Figure 11-28:** Uniquely name your doors so you can later call to that name to open them.

6. **Add a Lip value of 4 to the door.**

   When a sliding door opens, the distance it slides is equal to its length. When the door slides open, then, it seems to disappear into the wall. It would be more realistic if the door didn't actually slide all the way into the wall. The distance that is left sticking out of the wall is the Lip and it's measured in units. So, a Lip of 4 means that the door stops 8 units short of the total distance into the wall.

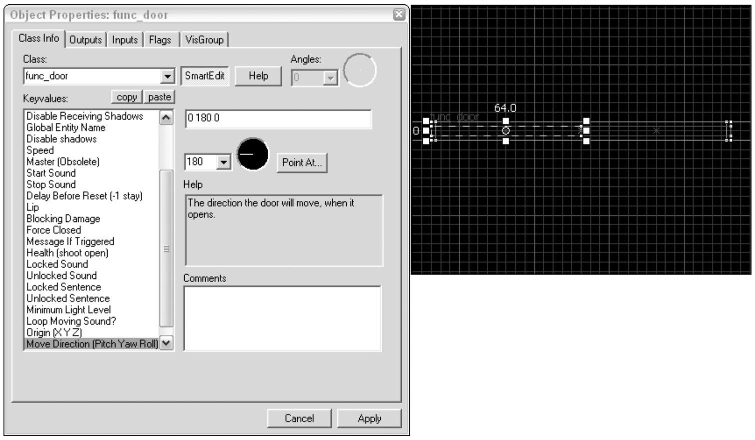
   Entering a negative Lip value causes the door to slide further into the wall. So if you entered a value of –4, the door would slide an additional 8 units into the wall.

7. **Adjust the Move Direction to make sure that the door opens into the brick wall.**

   The door can open any direction you want. It can open up, down, left, right, and so on. You want your door to slide into the brick wall that it touches.

   From within the Keyvalues list, choose the second-to-last option, Move Direction. Look at the black circle that shows up when you make this selection and confirm that the white line within it points in the direction you want the door to move, as I have in Figure 11-29.

   Make sure that when you define the direction of the door, you do so in relation to the 2D Top viewport. This is the only view that properly reflects the angle at which the door should move.



Class: func_door

Angles: 0

Keyvalues:

Disable Receiving Shadows
Global Entity Name
Disable shadows
Speed
Master (Obsolete)
Start Sound
Stop Sound
Delay Before Reset (-1 stay)
Lip
Blocking Damage
Force Closed
Message If Triggered
Health (shoot open)
Locked Sound
Unlocked Sound
Locked Sentence
Unlocked Sentence
Minimum Light Level
Loop Moving Sound?
Origin (X Y Z)
Move Direction (Pitch Yaw Roll)

0 180 0

180

Help

The direction the door will move, when it opens.

Comments

**Figure 11-29:** Set the direction that the door needs to move when it slides open.

8. **Click Apply to set your changes.**
9. **Repeat these steps for the other door next to this one.**

   You want to follow the same steps as with the first door, except that you use a different move direction:

   a. Select the door brush.

   b. Turn it into a func_door entity.

   c. Give this entity the same name as the door next to it, door_blue.

   d. Assign it the correct movedir.

TIP

Entities can have the same name. In this case, you create another entity later that controls both doors at the same time. Because these doors have the same name, you can control them simultaneously.

That completes one set doors. Go ahead and apply the same principles to the second set of doors. Remember to assign the correct move direction so that the doors slide into the wall. Also, give the other doors a different name such as door_red. This way, you can control these doors separately as a set.

## Sealing your area

After you turn part of the wall into an entity, you create a hole in the sealed area of that room. In Chapter 10, you create a hint brush in the hallway to split the map into three areas. This way, only one room is rendered in the game while the player is in it, and the other room is ignored. By creating a hole (by way of the doors) in one of the rooms, the area leaks into the rest of the map and renders everything again.

Thankfully, the solution to sealing this hole is simple. By adding an areaportal inside the doors that fits the entire doorway hole, you can effectively redefine the room as its own area and seal it again. The area remains sealed until the doors open and seals again when the doors close.

To create this doorway portal, follow these steps:

1. **From the textures group on the right of the editor, browse, filter, and select the texture `tools/toolsareaportal`.**

   This texture helps define the new brush that you are about to create as a portal. The texture is green and reads AREA PORTAL, as shown in Figure 11-30.

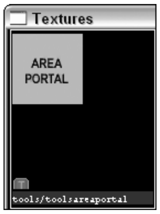


**Figure 11-30:** An areaportal must be textured with the `tools/toolsareaportal` texture.

2. **Draw a brush that fits completely inside both your doors and seals tightly against the brick wall and ground in all four directions.**

   Your areaportal must create a tight seal around the door to ensure that there is no leaking of the space it separates. If a leak occurs, this area won't be sealed, and no separation is created.

   Also, the brush you draw must be thin enough to fit inside your doors (Figure 11-31). Although the player cannot see this brush, it does cause everything on the opposite side to be hidden from the player's view. This creates some unwanted visual effects that can be avoided by placing the brush where it can't be directly seen.

TIP

In order to get your area portal brush to fit inside of your doors, you have to adjust your grid size to 1. When you are done sizing this brush, increase your grid size so that you can see the grid lines when you zoom out. If you don't, you may accidentally move a brush and not see it move because you are zoomed out and cannot see the grid lines.

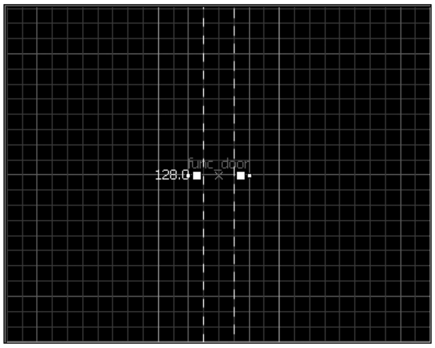


**Figure 11-31:** Create your areaportal brush to fit inside your doors.

3. **Change to the Selection tool and select your areaportal brush.**

   TIP

   After creating the brush, when you switch to the Selection tool (press Shift+S), the brush you just created should be automatically selected. If not, try to select it from one of the 2D windows. If that doesn't work, move or hide the doors so that you can easily select the brush from your Camera viewport.

4. **Click toEntity to turn your world brush into an entity.**

   This button is found on the right side of your editor in the New Objects group.

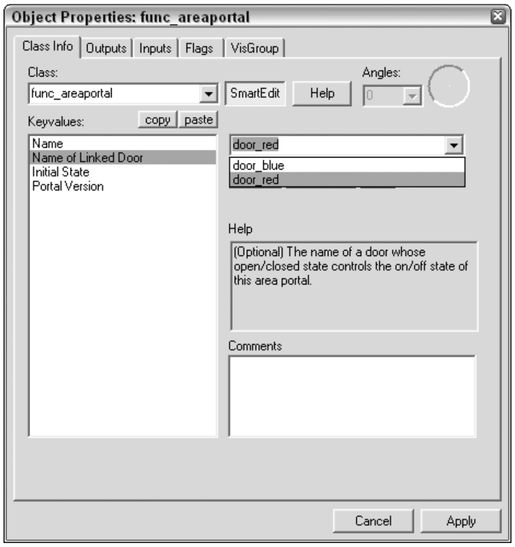
5. **Select func_areaportal from the list of classes.**

6. **Select Name of Linked Door from the list of Keyvalues and enter the name of your doors into the text box on the right.**

   If you named your doors as I did, the value entered here for the areaportal near the blue-lit room is door_blue. The name of the other doors is door_red. Also, if you forgot the name of your doors, you don't have to go back and look it up. The name is available in a drop-down list if you press the arrow to the right of the text box (Figure 11-32).

7. **Click Apply and repeat these steps for the other set of doors.**

   When completed, save your map again.


Object Properties: func_areaportal

Class: func_areaportal

Angles: 0

Keyvalues:

- Name
- Name of Linked Door
- Initial State
- Portal Version

door_red

- door_blue
- door_red

Help

(Optional) The name of a door whose open/closed state controls the on/off state of this area portal.

Comments

**Figure 11-32:** All the current entity names in your map are available in the drop-down list.

## Lighting from Above

For players to be able to see inside your building, you add light entities. Right now, if players go into the outdoor area, they won't be able to see anything because there is no light. Again, you must add light entities. However, in the case of your outdoor area, instead of just adding a light bulb to the ceiling, your goal is to replicate something that resembles the sun. This can be accomplished with the use of a new entity, light_environment.

### Adding virtual sunlight

Perhaps the best place for the placement of your sun is in the corner opposite your rooms. The light can shine down, casting nice shadows and creating a great-looking environment for your players.

Select the Entity Creation tool by pressing Shift+E. From the New Objects group on the right side of your editor, select the light_environment entity from the Object list. Click and place your new entity into your map, and then press Enter to insert the entity, as shown in Figure 11-33.

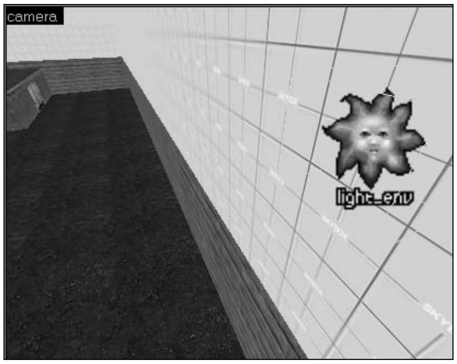


**Figure 11-33:** Add the light_ environment entity to create the effect of sunlight.

TIP

Technically, the placement of this light entity has no effect on the lighting of your level. The values that you set for this entity create and define the light attributes. However, if I ever need to make changes, I find it easier to locate this entity when I place it where I expect the light to be coming from. This is why I place the entity in this corner.

With the light_environment entity still selected, press Alt+Enter to open the Object Properties window for this entity. Enter the following Keyvalues:

- **Pitch Yaw Roll (XYZ): 0 225 0**

  This defines the direction that the light faces. By entering a Yaw of 225, you tell the light to shine toward the southwest corner of the map.

- **Pitch: –45**

  This setting tells the light to shine toward the ground at a 45-degree angle.

- **Brightness: 255 255 185 600**

  The brightness of the light sets the color and amount of light to shine. The first three digits represent the light's RGB color values. Because natural sunlight has a yellow coloring to it, so does this light. The brightness, represented by the value 600, defines how brightly the light should shine. Although 100 is usually the top value for brightness, you can exaggerate this value for a better effect, as I have.

- **Ambient: 190 200 220 80**

  *Ambient lighting* is what you find in corners or other areas that don't have any direct source of light. The light could come from a reflection of walls, ground, or anything else similar.

  The values for this setting follow the same outline as that for the Brightness. The color of this light — defined as RGB values in the first three numeric values of this settings — is usually darker, like what you would see at dusk. Also, the brightness — defined as the last numeric value of this setting — is usually much lower.

After you make these settings for this entity, click Apply. Then save your map.

At this point, you may want to take another step in making your virtual sunlight look more like an actual sun.

## Adding a virtual sun

You now have enough light in your outdoor area. If you stop here, the level will look good in the game. However, you could also take the extra step to add something in the sky that looks like the source of your light — a sun.

Adding a sun to your map is as easy as adding another entity called env_sun. This entity won't actually add any light to your map, but it creates the effect of a sun in the sky when the player looks toward it.

With the Entity Creation tool still selected (Shift+E), change your selected Object located on the right side of the editor in the New Objects group. Select the object env_sun from the list.

In the case of this entity, placement does matter. The player will see it — or, at least, the player sees the effects that it creates within the game. So, place this new entity into your map in the same corner where you placed the light_environment entity (see Figure 11-34).

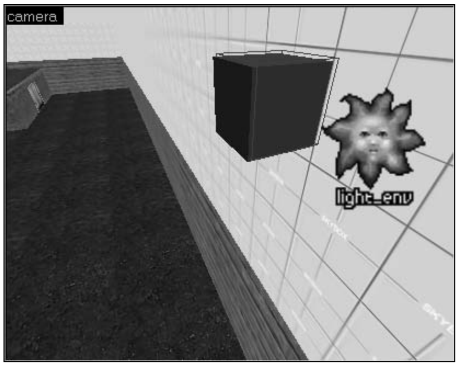


**Figure 11-34:** Place the env_sun into your map for a simulated sun effect.

When the fake sun is in your map, define the angle at which it shows and shines into the player's eyes when they look at it. You could do this by changing the properties of the env_sun entity, but that can be tricky to get just right. The easier method is to place something else into your map at which you can point your env_sun.

Select a new object within the New Object group on the right of your editor. Select info_target from the list. Place this new entity between your two rooms near their corners, as in Figure 11-35. With the sun shining down to this location, you wind up with the right effect.

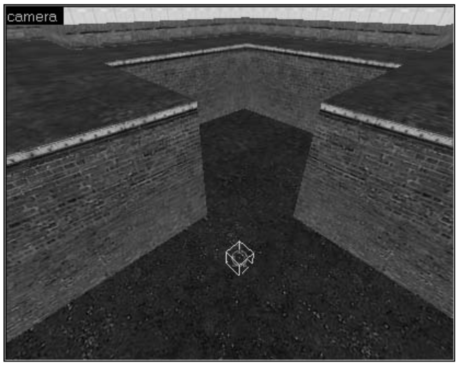
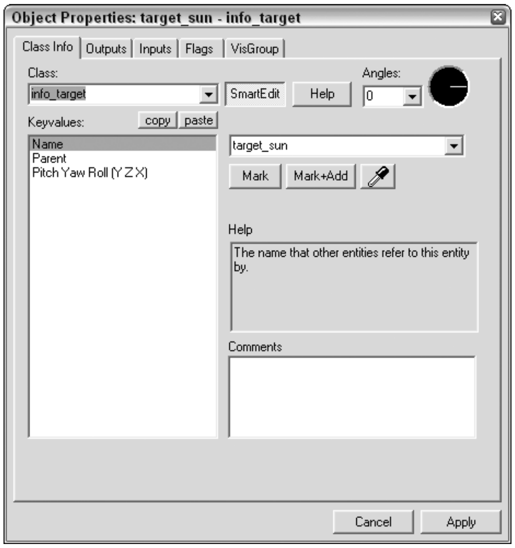


**Figure 11-35:** Place an info_target in your map that you can point the env_sun at by name.

With the info_target entity selected, press Alt+Enter to access the Object Properties window. Enter a name for this entity so that you can call to it from the env_sun entity. Make sure the Name is selected (in the Keyvalues list) and then enter the new value into the text box to the right, as in Figure 11-36. A good name here would be target_sun.



Class: info_target

Angles: 0

Keyvalues:

Name
Parent
Pitch Yaw Roll (Y Z X)

target_sun

Help

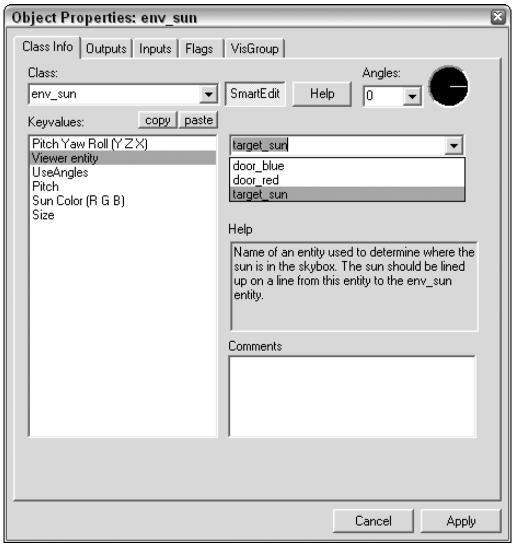
The name that other entities refer to this entity by.

Comments

**Figure 11-36:** Set a name for your target that can be called to by another entity.

After this name is entered, click Apply and close this window. Move your view in the Camera viewport to the env_sun entity and select it. Then press Alt+Enter again to open the Object Properties window.

Now that you have a target to point your sun at, you can define this property within the env_sun entity. Choose UseAngles from the Keyvalues list and make sure that is the UseAngles value is set to No via the drop-down list. This means that the game will ignore any directional settings within this entity. Then choose Viewer Entity from the Keyvalues list; and, from the drop-down list on the right, select your new target, target_sun (see Figure 11-37).



Name of an entity used to determine where the sun is in the skybox. The sun should be lined up on a line from this entity to the env_sun entity.

**Figure 11-37:** Target your sun by setting the Viewer entity.

## Testing Your Progress

Finally, after all the work you just put in, you can see how your new outdoor area looks in the game.

Compile the map; run it through the BSP process by pressing F9. Make sure your three compile options are all set to Normal and that the option to not run your map is not selected. Then press OK to compile and test your map.

As the map compiles, you will notice that it now takes a lot longer to finish. The editor has more open space and more light to calculate during the compile process. The more detail that is added to your map, the longer it takes to compile.

If you encounter a leak and the compile process stops, load your point file and follow the red line in your map to the hole and seal it up. I discuss how to find leaks in Chapter 8.

Take a look around your new map (see Figure 11-38). Look up at the sun as you run around the outside area. Most of all, have fun!

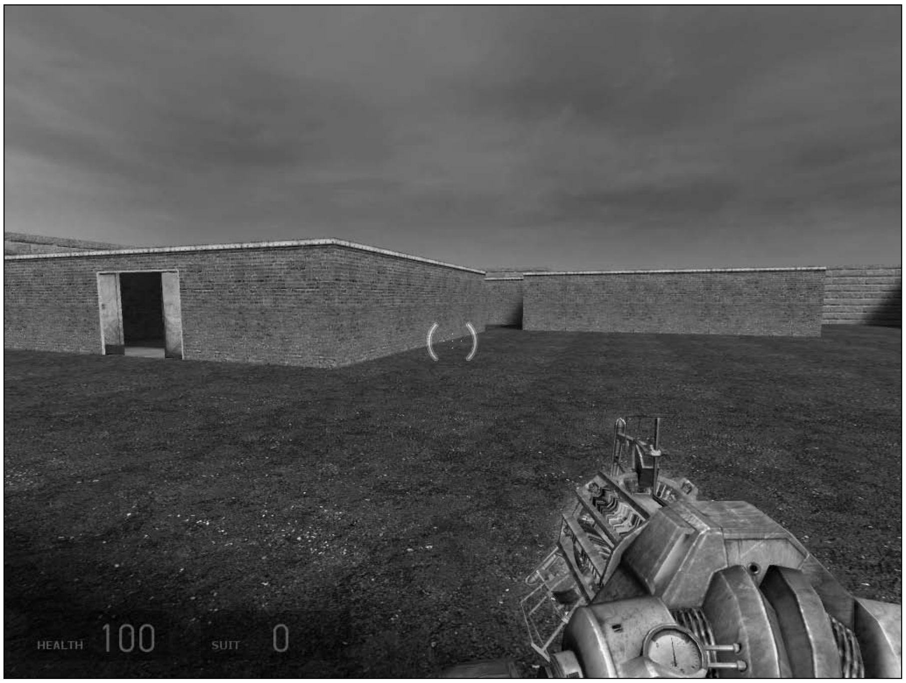


**Figure 11-38:** A well-lit map with nicely defined shadows from the direction of your sun.

## Chapter 12

# Adding a Few Details

*In This Chapter*

- Adding a few brushes for detail
- Teaming doors to work in unison
- Dropping crates in honor of the first-person shooter
- Placing pickups for the players

You've put a lot of work into making your first map for *Half-Life 2.* In the preceding chapters, you create two interior rooms connected by a hallway. By adding sliding doors, you extend the play area to go outdoors with a sky and outdoor lighting. With all the work you put into this map, you could call it a second home. However, what is a home without a little decoration? If your home had no furniture, no rugs, no details to make things interesting, it wouldn't be a fun place to hang out.

What's needed are a few details in the playing area. Some details are small and won't be noticed by the player, but others will make a huge difference in the game play of the level. However, all details, no matter the size, add to the overall feel of a map and make it worth playing over and over again.

## Doing Some Decorating

To start off, I show you how to add some minor details to the layout. These minor things might not ever be recognized by the other player but will really add depth and character to the environment.

First, I show you how to enhance the outside walls. Even though the texturing helps to make them interesting, extra depth created with the addition of just a few brushes makes your walls look strong and secure.

## Pulling a ledge

In this section, you add a ledge to the top of the wall. A ledge makes the top look and feel more prominent when the light casts a shadow. It makes the walls feel deep. The change doesn't need to be big, but it should be noticeable.

You draw a ledge around the entire outer wall at the top where your current texture starts to repeat itself. The brush should be 32 units tall and 8 units deep. This size is big enough to give the desired effect without be so big that it takes over the scene.

Here's how you add the ledge:

1. **Deselect all brushes and select the Nodraw texture.**
2. **Starting in the southeast corner of the map, draw your first brush in the direction of your building to 16 units tall and 8 units deep. Stop when your brush reaches the end before it intersects with another brush.**

   Draw this new brush flush with the top of the inner wall but not overlapping with any other brushes. This ledge should look like it is overhanging the top of the wall surrounding this outside area. The result should look like Figure 12-1.

   Note that because this brush doesn't overlap with the building, I could continue to draw the brush to the end of the next wall. If this brush were to overlap, I would have to stop and continue with a shorter brush that did not overlap.

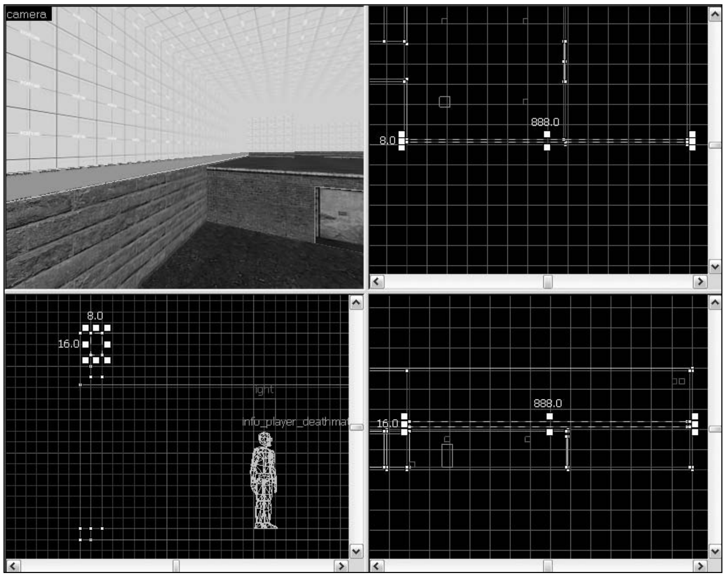


**Figure 12-1:** Create a ledge on the wall beginning in the lower-left corner of the map and working toward the building.

3. **Move your view back to the southeast corner and continue the ledge brush on the eastern wall.**

   The only spots that might cause you trouble are the corners. You need to miter them as you go along, just as you did in Chapter 10.

4. **Start the mitering with the southeast corner and miter each corner as you go along with your ledge.**

   Press Shift+V for the Vertex tool. It should look like Figure 12-2 when you're done.

5. **Continue adding the ledge brush around the walls until it is complete, making sure that the ledge wraps around the entire enclosed area.**

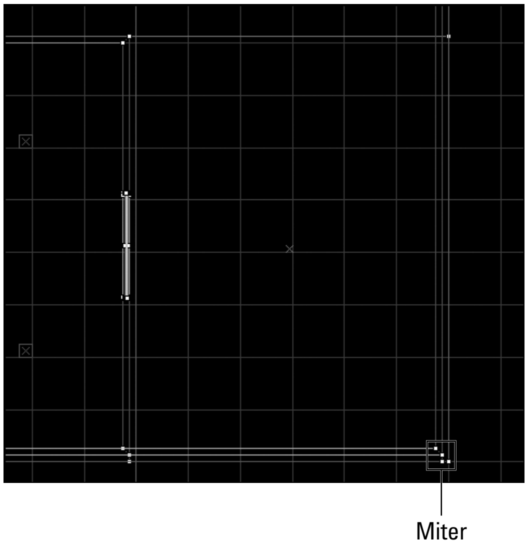


**Figure 12-2:** When you get to the corners, miter the corners to reduce splitting your geometry.

When you've added all the ledge brushes, it's time to texture these new brushes. You want to add something that doesn't completely blend in with the current wall but doesn't look out of place.

1. **Select the Texture Application Tool by pressing Shift+A.**

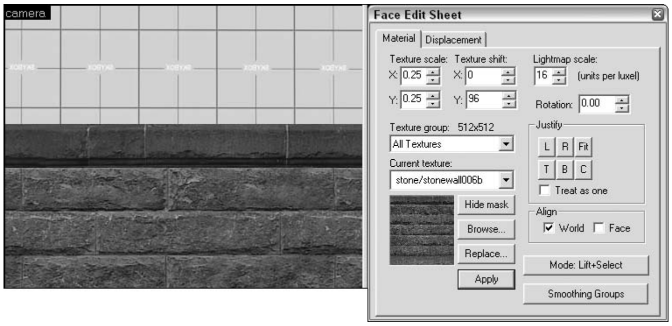
2. **Browse your textures and locate and select the texture** `stone/stonewall006b`**.**

   Yes, you could use the same texture already on your walls. However, you can make your scene more interesting by mixing other textures for more variety. In this case, you'll use the small, top edge of the texture to create your final look.

3. **Apply the texture** `stone/stonewall006b` **(shown in Figure 12-3) to the front brush face of the ledge brush.**

4. **Shift the texture so that the ledge portion can be seen.**

   This texture is a complete wall with a ledge on the top. It's the ledge that you want to apply to this brush face, without the rest of the wall. When you apply the texture, you will most likely not see the ledge. So, shift the texture on the Y axis to make it visible. Do so from the Face Edit Sheet window by increasing the value in the Texture Shift Y text box. The resulting value in my case was 96, as shown in Figure 12-3. However, on your computer, a shift value of 96 may not work. So, line up the texture by eye as you adjust the shift values.



Face Edit Sheet

Texture scale: X: 0.25 Y: 0.25
Texture shift: X: 0 Y: 96
Lightmap scale: 16 (units per luxel)
Rotation: 0.00
Texture group: 512x512
All Textures
Current texture: stone/stonewall006b
Treat as one [ ]
World [✓] Face [ ]

**Figure 12-3:** Select and apply the `stone/stonwall006b` texture to your new ledge.

5. **Apply the same texture to the bottom and top of this brush so that all visible Nodraw texturing is covered.**

   When you apply this texture, you're going to see that it doesn't line up well. Shift this texture around until you like how it looks. In my map, I shifted the texture on the Y axis to 102 and copied this to the top of the brush so it looks like Figure 12-4.

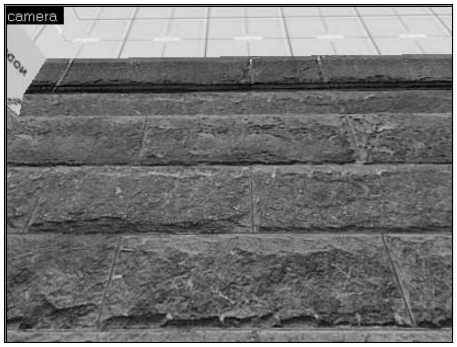


**Figure 12-4:** Apply the same texture and shift it into place under the ledge.

6. **Apply these textures to all of the remaining ledges around the walls of your map.**

   In some cases, when you apply the texture underneath the ledge, you will notice that is it going in the wrong direction, like it is in Figure 12-5. To fix this, rotate the texture 90 degrees by using the Rotation values in the Face Edit Sheet window.

7. **Now that your ledge is fully textured, save the map under the new name** `dm_chapter12.vmf`**.**

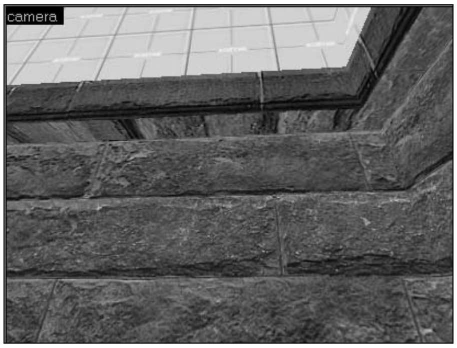


**Figure 12-5:** You may need to rotate some textures to better line them up.

## Lighting the porch

Something else that adds a lot of detail to a level is lighting. I don't mean just the general lighting that makes it possible for the players to see where they're going. I'm referring to the lights that accent points in the game, such as entrances.

With the following instructions, you add a light over each pair of doors. This makes the doorways more obvious to the player and makes them more interesting at the same time.

1. **Select the Block Tool and the Nodraw texture. Then move the camera around and in front of a set of doors, viewing it from the exterior.**
2. **Draw a brush that is 16 × 16 × 12 units (X, Y, Z) in size and place it right above the trim that surrounds the doors. Center it above the doors so that it looks something like Figure 12-6.**

   This brush is the light's basic structure, but you need to shape it so it doesn't look quite so boxy.

3. **Reduce your grid size to Grid 4 by pressing the [ key. Then press Shift+V to show the brush edges.**

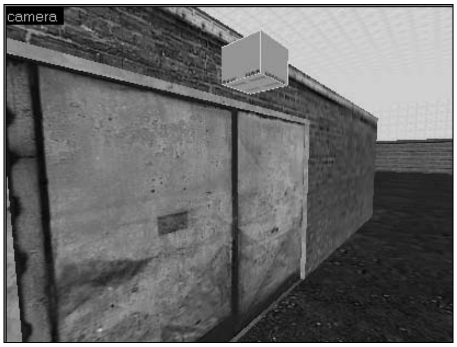


**Figure 12-6:** To start the light structure over the door, draw a box with the Nodraw texture.

4. **Drag the top-front brush edge down so that the top of the light fixture is sloped downward but there is still a 4-unit lip in the front.**

   WARNING!

   Be careful not to drag the edge and create a three-sided brush because this type of brush manipulation will create an error during the compile process. The result should look like Figure 12-7.

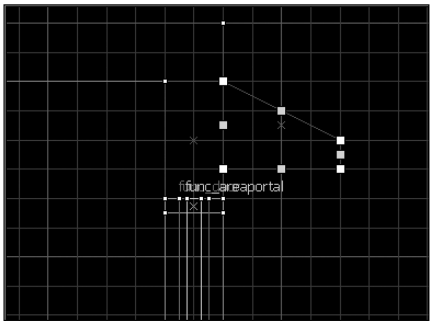


**Figure 12-7:** Move the edges of your brush to create a shape that looks more appropriate for a light box.

That completes the light fixture structure. Now you need to texture it:

1. **Select the Texture Application Tool by pressing Shift+A.**

   There are a lot of nice accent lights here for you to play with.

2. **Browse to and select the texture** `lights/combine_lightpanel001`**.**

   This texture looks like a round light in a metal casing.

3. **Apply this texture to the bottom of the light box that you just created and Fit it to the brush face.**

   You need to click Fit from within the Face Edit Sheet to adjust the size and placement of this texture. It's larger than your brush face, and using the Fit button is the simplest method of adjustment.

4. **As for the remainder of the light fixture, texture it with a Basic Steel texture.**

   A fine texture I found for the outside of your light box is `metal/metalwall031a`. It looks a lot like the metal on your doors and doesn't look out of place in the map.

### *Optimizing the light fixture*

In the preceding section, you completed the creation and texturing of the light, but doing so really hurt the map's optimization. In Chapter 10, which discusses optimization, you see how your map is split into multiple polygons in the game. Every place two brushes met, they created a split.

If you were to take a look at your current map in the game by using the `mat_wireframe` console option also mentioned in Chapter 10, you would see multiple splits occurring around the light structure that you just made. In Figure 12-8, I have enabled `mat_wireframe 3` and also used a new console command, `mat_drawflat 1`. You can now see the splitting of brushes more clearly. More splits occur around the light, and this added geometry will slow your map, especially as you add more detail to the level.

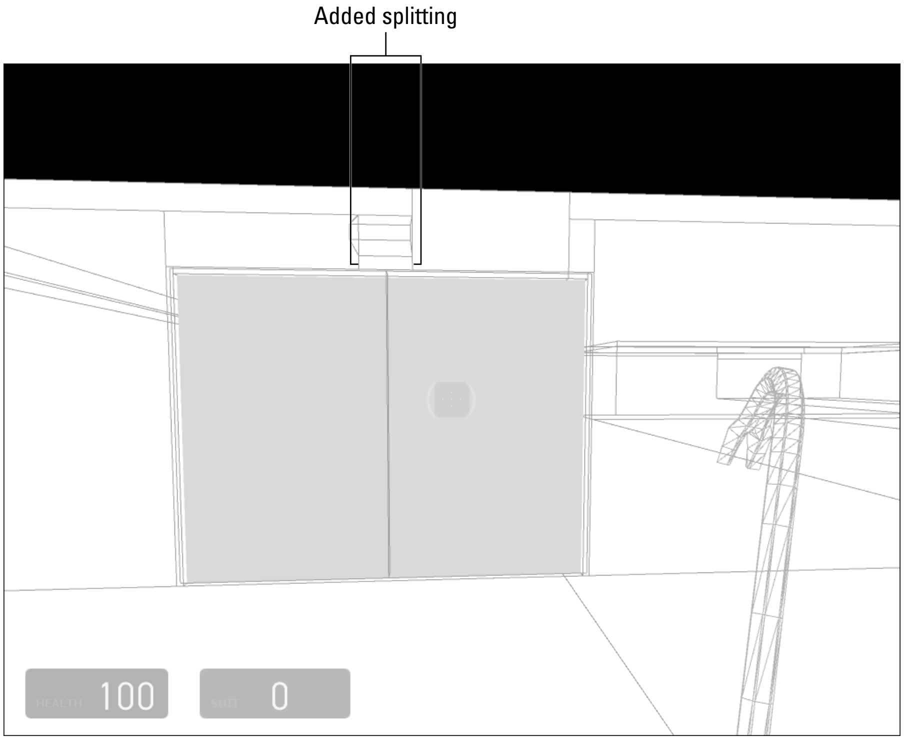


**Figure 12-8:** The right console commands can make finding problems easy.

TIP

The console command `mat_drawflat 1` renders your level without textures and with flat colors in their place. I sometimes use this to get a better view of the geometry in my map when textures are obscuring the picture.

To solve this problem of splitting brushes, you must define your brush as an entity, not as a structural brush. Only then will it be calculated during the BSP process as something that doesn't create all these splits. You want to turn it into a `func_detail` entity. This entity is meant for converting regular brushes into simple entities.

1. **Select the light fixture brush that you just created.**
2. **In the New Objects group on the right side of the editor, click toEntity.**

   Your brush is turned into a `func_detail` entity by default.

   The Object Properties window pops up with the default selection of `func_detail` selected.
3. **Click Apply to convert your brush into the entity that will save your map from a horrible fate: slowness.**

Now, when you go back into your level within the game and view the wireframe drawing of splits, you will see that the splits are gone, as shown in Figure 12-9. The light box will look the same in the game without creating the added geometry.

TECHNICAL STUFF

In some current games and many older games on the market, you were able to turn a structural brush into a detail brush. This reduced the geometry in the same manner as with the `func_static` entity used in the section "Optimizing the light fixture." However, this function of the editor is being phased out and replaced with an entity. This makes the editing process less confusing and performs the same function in the game.

### Adding a light entity to your light fixture

So, the light fixture looks complete as a structure, but this won't shine any visible light in front of the door. To shine a little light on things, you need to add a light entity, or instead, a `point_spotlight` entity:

1. **Select the Entity Tool (Shift+E) and select `point_spotlight` as an Object in the New Objects group on the right side of the editor.**
2. **Place the entity under your light fixture and press Enter to insert the light into your map.**

   This new entity won't look like a lightbulb. It will look more like a colorful, 3D asterisk similar to Figure 12-10.

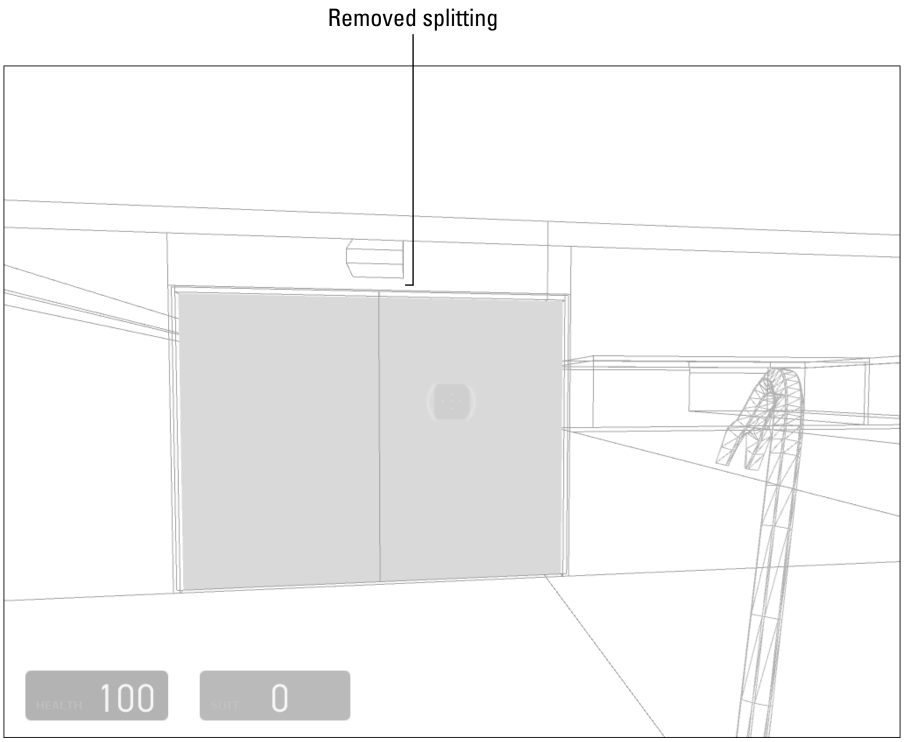


**Figure 12-9:** Turning the light fixture into a `func_detail` entity has rid your wall of added splitting.

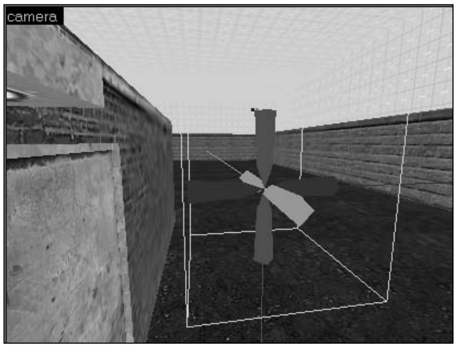


**Figure 12-10:** A `point_spotlight` will give the effect of light without adding light to your level.

3. **Press Alt+Enter to open the properties for this** `point_spotlight` **object and adjust the Angles to Down and the color to medium gray.**

   You want the spotlight to shine downward, so you need to specify that. On the right side of the Object Properties window, you'll see an Angles drop-down list. Select Down from the list.

Also, this light produces an effect so that, within the game, when a player looks up at it, a bright, white light obscures the view. To turn down this brightness, you must darken the color of the light. Adjust the Color of the light to 136 136 136, which is a medium gray.

4. **Position the light entity so that its center point is directly under your light fixture, as shown in Figure 12-11.**

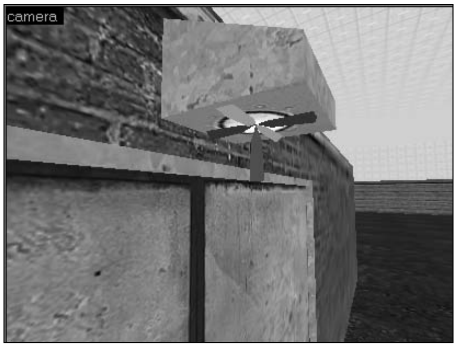


**Figure 12-11:** Position the center of the entity where you want the effect to begin.

### Duplicating your light fixture

After you've completed your light fixture, you can duplicate what you have done for the other set of doors, rather than remaking everything. To copy what you have already created, follow these steps:

1. **Select the brush and spotlight in your map.**

   To select multiple objects, hold the Ctrl key while clicking them in your camera viewport.

2. **With the Selection tool active, hold the Shift key and click and drag your selection in the Top 2D viewport to the other side of the map, where you want to place the duplicate.**

3. **With your duplicate now near the other set of doors, click on your selection again in one of the 2D viewports to activate the rotation handles around these brushes.**

   The rotation handles look like circles on each corner of your selection in your 2D viewports (see Figure 12-12).

4. **Rotate your selection so that it is properly oriented above your door.**

   Click and drag one of the circular handles around your selection to rotate it.

5. **Move it into place over the doors.**

   Position your light over these doors just like the other light fixture.

   You're done over here, and you should have a nice layout like I have in Figure 12-13.

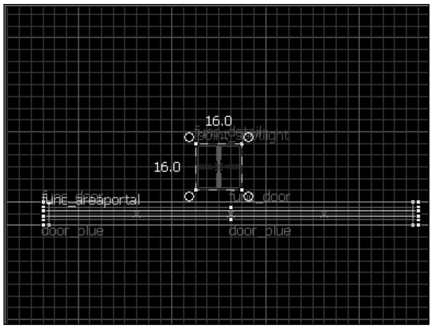


**Figure 12-12:** Position the center of the entity where you want the effect to begin.

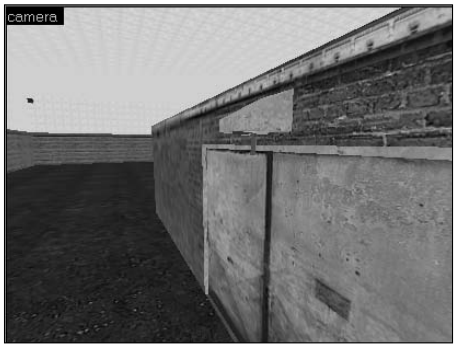


**Figure 12-13:** Duplicate, rotate and position your first light to create a second.

## Defining the indoor lights

For the outdoor lights, you created a brush and textured it with something that looks like the casing for a light structure. When the player notices the light over the door in the game, it's obvious that this is where the light is emanating from. However, you didn't do this for the lights inside the rooms. The lights indoors just seem to magically appear; they should be defined by a source.

The following instructions show you how to create light fixtures for the interior of your rooms:

1. **Using the Nodraw texture, select the Block Tool and draw a brush 16 units square and 4 units tall inside one of the rooms.**
2. **Position the brush above one of the existing lights in the room.**

   The result should look like the brush in Figure 12-14.
3. **Texture the four sides of your light box with the same metal texture you used to surround your outdoor lights,** `metal/metalwall031a`**.**

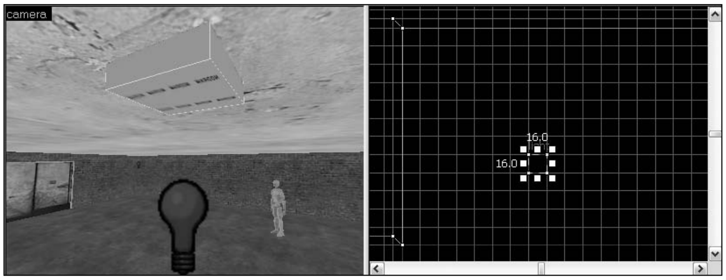


**Figure 12-14:** Creating a light box over each of the lights inside of your building provides a logical source for the light.

4. **Texture the underside of the light box with the** `lights/fluoresentcool003a` **texture.**

   This fits a square brush and looks good. After the texture is applied, make sure to click Fit in the Face Edit Sheet to align the texture properly to the brush face.

   After the structure is created and textured, you need to turn this brush into a `func_detail` entity, just like you did with the outside lights in the section "Optimizing the light fixture," earlier in this chapter.

   TIP

   Take a look at the texture `lights/fluoresentcool003a`, shown in Figure 12-19. The white dot under the selected texture indicates that this texture gives off its own light when applied to a world brush. The problem is that the lighting property doesn't work on an entity like `func_detail`. Rather than add more geometry to the map by not making your light box an entity, you can apply the texture to the entity and add your own lights underneath it.

5. **Select the new light box, click toEntity, select** `func_detail` **from the list of classes, and apply the changes.**

   The toEntity button is in the New Objects group on the right side of the editor. When you click it, the Object Properties window appears, and you can choose your entity and apply your changes. Converting this world brush into an entity keeps the geometry in the game to a minimum and keeps your level running smoothly.

   Close the Object Properties window after the changes have been applied.

6. **Duplicate the light structure (but not the light itself) and place it above each of the light sources within the indoor areas of your map.**

   Remember, select the object you want to do duplicate and then press Shift+click and drag your selection to make a duplicate.

At this point, I decided to compile and run my map to see how things look. While looking around inside the room, I realized that the lights did not look as good as they could (as shown in Figure 12-15). This was for two particular reasons:

- The lights look way too bright on the ceiling.
- The color of the light entity does not match the color of the light fixture.

Some of this can be fixed with some tweaking of the light entities. However, the color difference is an obstacle that can't be overcome at this time. Looking at the texture of the fluorescent bulbs, you can see the light coloring isn't even a close match. Later in the book (Chapter 13), I get into creating custom textures, and you discover the techniques used to create a new texture. You can then apply what you learn to making a custom light texture that radiates the color light that you require. However, for now, I recommend making a good-looking map by doing away with the color in the rooms and sticking with a white light that better suits *Half-Life 2* in general.

TIP

Colored lights similar to what you had up to this point are popular in arena deathmatch levels. The bold colors add to the intense game play that occurs in arena matches. Now that you understand the basics of coloring lights, I now want to show you some more contemporary methods of lighting a level.

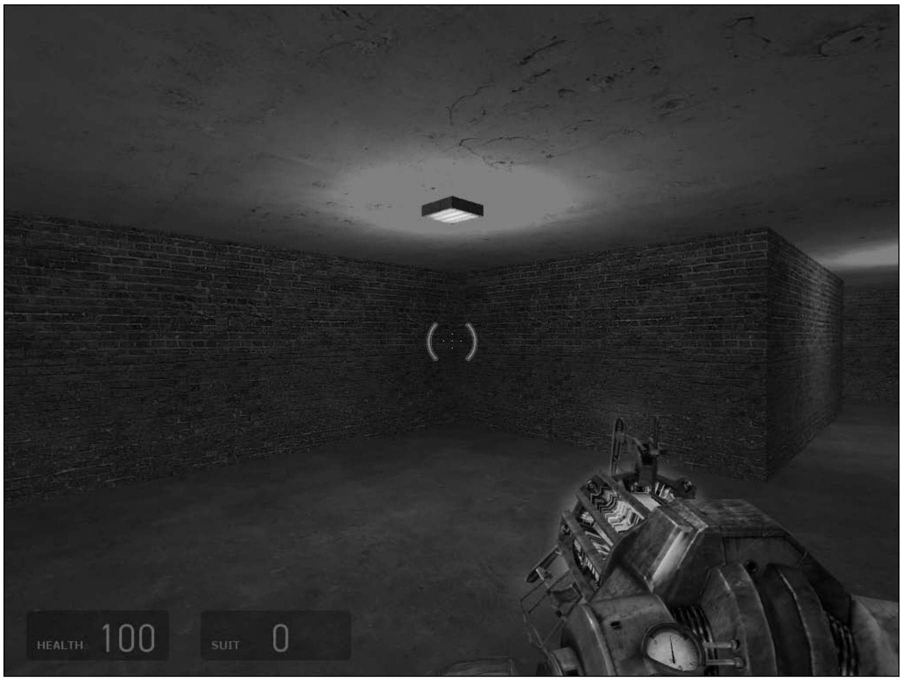


**Figure 12-15:** The lighting in the new level doesn't look as good as it could.

You need to change the color of each of the lights located in the two rooms and the hallway. You can perform the same changes to all these lights at the same time by pressing Ctrl and clicking on each light within the camera viewport. Then follow these steps:

1. **Move the lights down until they evenly fit between the floor and ceiling.**

   This can be done by clicking inside your selection from within the side or front viewport and dragging it down, as shown in Figure 12-16.

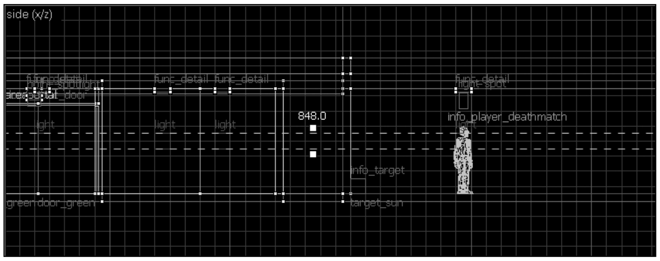


**Figure 12-16:** Move the lights down to reduce the brightness on the ceiling.

2. **While your lights are still all selected, press Alt+Enter to open the Object Properties window.**
3. **Select Brightness from the left column in this window and enter** 235 255 210 40 **in the text box on the right.**

   The first three integers define the RGB color value of the light as a very light yellow, similar to that of normal fluorescent lighting. The last integer defines the lights' brightness, which is much lower than the default 200. You compensate for this lack of light shortly.
4. **With the new settings in place, click Apply and then close the window.**

   Your lights now all look the same color and are equally positioned throughout the rooms. But now you need to add a little more definition to your lights.
5. **Select the Entity Tool and change the object to** `light_spot` **in the New Objects group on the right side of the editor.**

   You next add a spotlight under each of the light fixtures to produce the downward lighting that is expected.
6. **Position the entity under one of your light fixtures and press Enter to add the new object.**

7. **Switch to the Selection Tool and move the light so that it is resting directly under the light fixture, as shown in Figure 12-17.**
8. **Pres Alt+Enter to open the Object Properties window for your new spotlight.**
9. **Select Brightness on the left, and on the right, enter** 235 255 210 200**. In this case, the default brightness is fine, however the color should match that of the light underneath it.**

   By default, the spotlight should look like it's pointing downward in the camera viewport. If it isn't, then adjust the Angles defined in the top-right corner of the Object Properties window. Selecting Down from the spotlight's angle should fix itself.
10. **Click Apply and close the window.**
11. **Place copies of your spotlight under each of the other light fixtures within the rooms and hall.**

   You can do this by pressing Shift+clicking and dragging your selection under another light box. This is easily done from the 2D top viewport.

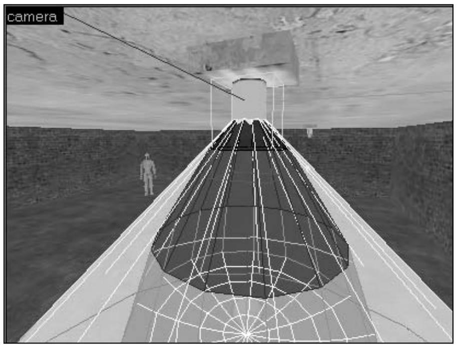


**Figure 12-17:** Position the spotlight directly under the light fixture.

This time, when I compile and run my map, the lighting looks much better. Figure 12-18 shows that the light is neither too bright nor too dark. It nicely fits the new environment.

At this point, you might want to consider fine-tuning the operation of the doors. Right now, when you walk up to one either pair of doors in your level, the first door of the pair you approach opens before the other. You could instead pair the doors so that they work as a team. Then, when one door is activate by an approaching player, both doors will open simultaneously.

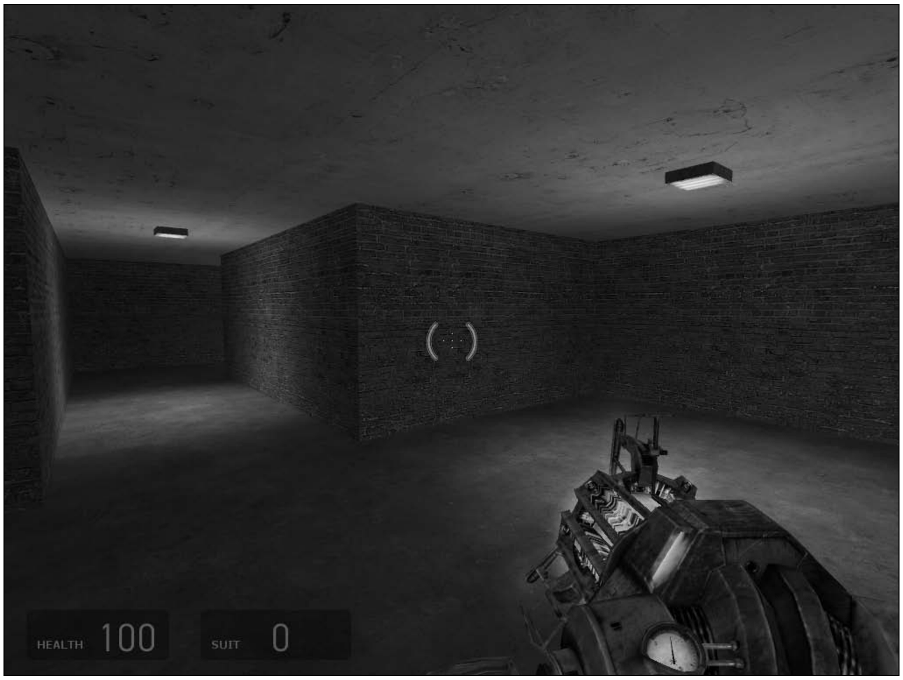


**Figure 12-18:** The new lighting greatly improves the look of the level in the game.

## Pairing the Doors

You created two sets of doors. Both sets offer a way to move between the indoor and outdoor areas in the map. In testing your map, you might have noticed that it was possible to walk up to a set of doors and make only one of the two open. In some cases, this might be something that you want. However, in this map, it would make more sense to have both doors open simultaneously regardless of the angle at which the player approaches them.

You can tell the game that a group of doors are to work in tandem. The method is to define the doors with the same targetname and then trigger them with a separate entity so that when the trigger is activated, both doors open simultaneously.

1. **Select the** `tools/toolstrigger` **texture from within the Textures Group on the right side of the editor.**

   To create this new trigger entity, you must first create a trigger brush. This is simply a brush that is textured with the trigger texture. This texture is a dark yellow with the word *TRIGGER* written on it.

2. **Select the Block Tool and draw a brush that encompasses the doors and extends at least 64 units out from both doors.**

   In Figure 12-19, I created a brush that was 144 x 152 x 152 (X, Y, Z) units in size and centered around the two doors. This brush is large enough to trigger the doors before the player walks into them but not so large that the player can't get close to the doors before making them open.

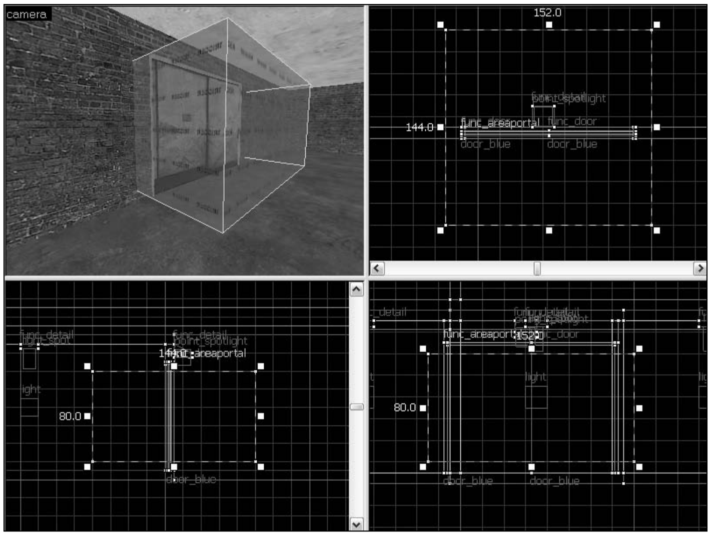


**Figure 12-19:** The trigger brush should encompass both doors.

3. **Switch to the Selection Tool, click toEntity in the New Objects Group, and turn this trigger brush in to a `trigger_multiple` entity.**

   The New Objects Group is on the right side of the editor. When you click toEntity, the Object Properties window opens. Select `trigger_multiple` from the list of classes. Then press Apply to apply this new entity to the selected brush.

4. **Click the Outputs tab in the Object Properties window.**

   The Outputs tab (shown in Figure 12-20) is where you define what happens when your trigger is activated.

5. **Click Add at the bottom of the window to add a new output action and instruct the trigger to open the doors when the player touches the trigger entity.**

First, you need to tell the trigger that when the player touches it, the doors should open. After clicking Add, make the following selections for each option in the window:

- *My Output Named:* **OnStartTouch**. This means that when the player first starts touching the entity, something should happen. That "something" is about to be defined.
- *Targets Entities Named:* **door_blue**. In this case, you're working with the `door_blue` door entities. These are the doors that you want affected by this trigger entity.
- *Via This Input:* **Open**. You want the doors to open (this is the "something" mentioned above) when the player touches the trigger.

6. **Click Add again to add a new output action and instruct the trigger to close the doors when the player stops touching the trigger entity.**

   The action you build in Step 5 tells the doors to open. Now you need to tell the doors to close when the player stops touching the trigger entity. After you click Add a second time, make the following selections for each option in the window:

   - *My Output Named:* **OnEndTouch**. This means that when the player is no longer touching the entity, something should happen.
   - *Targets Entities Named:* **door_blue**. This is the entity I want affected by this trigger entity.
   - *Via This Input:* **Close**. You want the doors to close when the player stops touching the trigger.

   Your Outputs tab should look something like Figure 12-20.

   TIP

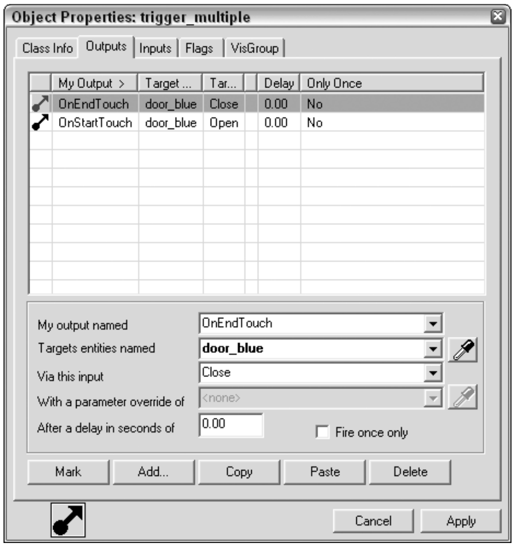
   You could have entered an output name of OnTrigger and set the input to Open instead of the two options listed above. Then you could skip the second output name. However, I find it better in the game when the doors quickly close behind the player as they run through rather than allowing the doors to wait their default wait time to close on their own.

7. **Click Apply and close the window.**

Follow the same steps above for the other set of doors. However, rather than triggering the `door_blue` entities, you want to trigger the `door_red` entities.

TIP

You could have specified the actions of the doors via the doors themselves. If you open the Object Properties for your doors and switch to the Inputs tab, you will see the same values that were set on the triggering entity's Outputs tab.



Object Properties: trigger_multiple

Class Info | Outputs | Inputs | Flags | VisGroup

| | My Output > | Target ... | Tar... | Delay | Only Once |
|---|---|---|---|---|---|
| | OnEndTouch | door_blue | Close | 0.00 | No |
| | OnStartTouch | door_blue | Open | 0.00 | No |

My output named: OnEndTouch
Targets entities named: door_blue
Via this input: Close
With a parameter override of: <none>
After a delay in seconds of: 0.00
Fire once only: ☐

Mark | Add... | Copy | Paste | Delete

Cancel | Apply

**Figure 12-20:** Define the actions of your trigger in the Outputs tab of the object's properties.

## Adding Some Crates

I can count on one hand the number of first-person shooters that don't have any crates. What's a first-person shooter game without a few crates? Crates provide players a place to hide, climb, or change the strategy of game play. They also add to the environment of the level so it doesn't look barren.

In this section, I show you how to build a crate for *Half-Life 2.* Then you logically place some crates in the map, both inside rooms and outside under the virtual sky.

You can place your crates anywhere in the map that you want; however, some places make more sense than others. You can use your crates to alter how the map is played. Here are a few ways you can use crates in your map:

- Restrict access to an area.
- Guide the players to follow a certain path.
- Offer places to hide weapons or players.
- Provide a means of climbing up to an otherwise inaccessible location.
- Create a safe place for the player to spawn into the map without being immediately spotted.

In some games, a crate isn't anything more than a textured brush. However, in *Half-Life 2,* the developers gave these old FPS (First Person Shooter) icons a purpose. These crates can be picked up with the gravity gun and can contain additional supplies, such as health or ammo. Because these crates have interactive properties, they are placed as entities into the game and defined as a model.

Start by placing a few nondescript crates indoors; it's the placement that can make the level more fun during play. Here's how you do place those new crates:

1. **Select the Entity Tool.**
2. **From the New Objects group on the right side of the editor, select** `prop_physics_multiplayer` **as a new object.**

   You need to use this entity because you're creating a multiplayer map. Using `prop_static`, `prop_physics`, or `prop_dynamic` may cause your model not to be displayed in the game unless it's a single-player game.

   Here is what each of the above mentioned entities add to your map:

   - `prop_static` (`_multiplayer`) is used for stationary models that have no player interaction.
   - `prop_physics` (`_multiplayer`) allows for interaction within the game.
   - `prop_dynamic` (`_multiplayer`) offers the ability to control this entity via scripting.
3. **Place this entity into your map near the bottom-right corner of the South room; then press Enter to insert the new entity.**

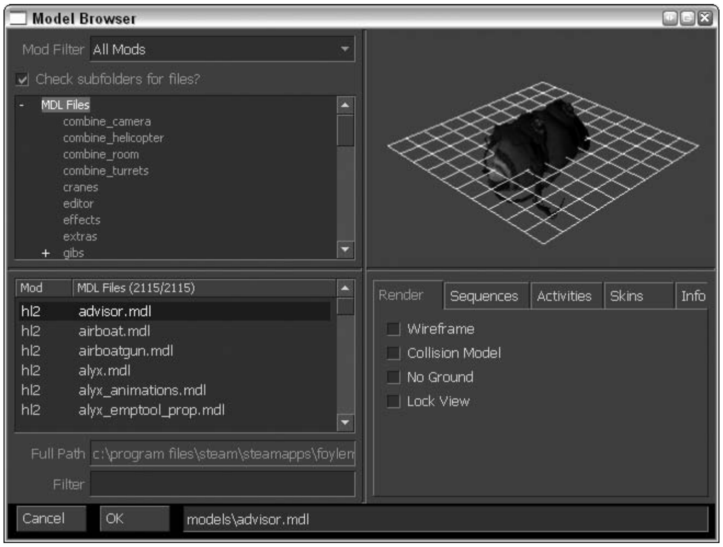
   When you insert this entity, it will look like a small, red dot. This is because you haven't yet defined what this item should look like.
4. **Press Alt+Enter to open the Object Properties window; select World Model from the list on the left.**

   The World Model defines the model that is displayed in the game.
5. **Click Browse under the text area on the right side of the Object Properties window and locate and select a new model from the Model Browser.**

   The Model Browser (see Figure 12-21) is a visual means of selecting models for placement in the game.

   - **Mod Filter:** Leave this set to All Mods because it can sometimes cause problems when selecting models.
   - **Directory Tree:** The models are divided up into directories for organization, just like the files on your computer.

- **Models List:** Below the directories list is the list of models. You can click through each of the models and see visual representations of them on the upper-right corner of the window.
- **Full Path:** Below the list of models is the file path for the model you want to select. You cannot change the file path; it's there just for information.
- **Filter:** Below the Full Path is the Filter text box. Enter a single word here to narrow the list to models that contain that word. You use this text box in the next step.
- **Model Preview:** The upper-right corner of the window shows a preview of the selected model. If you haven't selected a model, then the word *Error* appears here in 3D lettering.
- **Advanced Specifications:** On the bottom right, you can define more advanced characteristics of the model for the game.



Mod Filter: All Mods

[✓] Check subfolders for files?

- MDL Files
  - combine_camera
  - combine_helicopter
  - combine_room
  - combine_turrets
  - cranes
  - editor
  - effects
  - extras
  - + gibs

| Mod | MDL Files (2115/2115) |
|---|---|
| hl2 | advisor.mdl |
| hl2 | airboat.mdl |
| hl2 | airboatgun.mdl |
| hl2 | alyx.mdl |
| hl2 | alyx_animations.mdl |
| hl2 | alyx_emptool_prop.mdl |

Full Path: c:\program files\steam\steamapps\foylem

Filter:

models\advisor.mdl

**Figure 12-21:** The Model Browser is a great help in selecting models for placement in the game.

6. **Enter** crate **in the Filter text box and select the MDL Files directory from the list above.**

   When you enter the word *crate* into the filter text box, only those models whose names contain the word *crate* will be displayed. It will help you to narrow down your selections.

   You can preview each of the models within the subfolders listed in yellow near the top of the Model Browser window. Each of these subfolders help you to better organize all the models into smaller groups. By clicking on each model in the list beneath the folders list, you can preview that selected model.

However, by selecting the MDL Files category at the top of the list — the main folder that contains all of the subfolders — you will have a list of all the available models available to scroll through without regard to sorting by each subdirectory. Because you aren't looking at every single model but rather only those with the word *crate* in their titles, it's easier to click and preview each one.

TIP

To quickly browse through the models visually, click the first one in the list. Then press the up-arrow and down-arrow keys to move through the list. Each model will be displayed in the model browser as it is selected.

7. **Find and select the model** `props_junk/wood_crate001a.mdl` **from the list of models.**

   This model is the generic crate that you want for your map. You don't want to use the crate with the yellow barcode image because the player will think it has something useful inside of it, just like the original game.

8. **Click OK to confirm your model selection.**
9. **Click Apply and close the Object Properties window.**

   Your red box is now a wooden crate inside of your map.

Now you may need to move your new model around in the map for better placement. Make sure it is resting on the floor and also leave enough room for the player to walk by it. A player needs at least 40 units of space to fit between brushes, so space your crates at least this distance from the wall to allow access.

Now you have your first crate in the map, as shown in Figure 12-22.

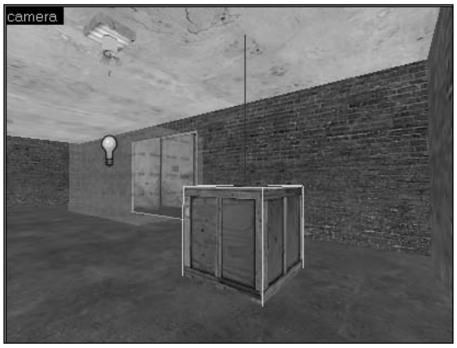


**Figure 12-22:** Place the crate on the ground and away from the walls.

## Placing crates for strategy

You have your first crate built and placed. Now it's time to add strategic opportunities to the game by adding more crates. Without the additional obstacles in the game, your players have little opportunity to hide and plan out an attack or retreat. It's your job to add elements to the level that give players choices. That is one of the functions of the crate.

To place more crates within your map, follow these steps:

1. **Move your first crate that you just created near the bottom-right corner of the South room. Leave just enough room — 40 units — for the player to squeeze through between the crate and the walls.**
2. **Duplicate the crate you just placed. Shift-click and drag your first crate to create a duplicate and move this duplicate just to the left of the first crate.**

   Rather than leave this new crate flush and parallel to the first one, rotate it, as outlined in the following steps, to add some variety and whimsy to the placement of the crates in the corner.
3. **Click the second crate multiple times within one of the 2D viewports until the rotation handles are visible.**

   The rotation handles look like circles on each of the four corners of the selection.
4. **Click and drag one of these rotation handles to rotate the crate to the desired angle.**

   Figure 12-23 shows how I placed the crates in this room.

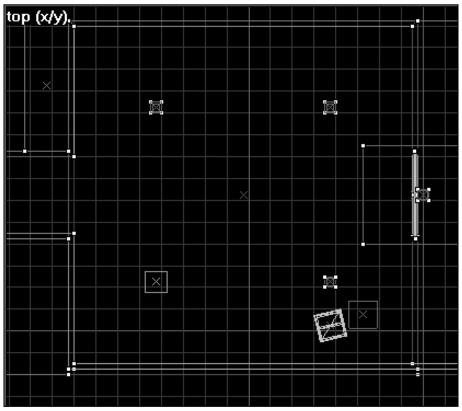


**Figure 12-23:** When placing copies of an object, rotate them or place them off-center from the others.

5. **Make another duplicate of the first crate and place it flush against the wall and to the left of the second crate.**

   Placing it here means the player can't run into the corner from this side. However, a player can forcibly move the crates with a weapon, thereby adding another element of strategy to this corner. This slows the player down a little bit while the other side is open for the player to simply run through.

6. **Place a couple more crates in this corner, but rather than setting them on the floor, place them on top of the existing crates.**

   This creates a really good hiding place or trap for the player. In Figure 12-24, I've not only placed duplicate crates on top of the others, but I've also rotated them slightly to make it more interesting. You should do the same.

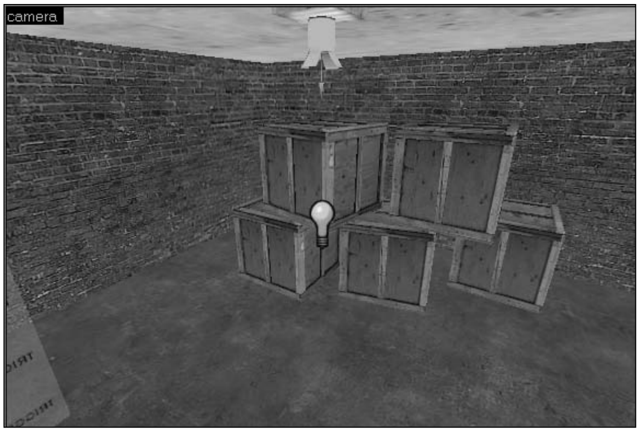


**Figure 12-24:** Stacked crates can make for good hiding places.

7. **Go through the rest of the indoor area of your map and place a few more crates.**

   Try to build some strategic areas for the players, but don't allow the placement to interrupt the flow of the game. Placing a crate in the middle of the hallway would make it difficult to traverse and possibly less fun. However, placing a few crates to the sides of the rooms may make it more interesting while also providing places to hide. Use Figure 12-25 as a guide if you need some ideas on how to place your crates.

## Making crates for other environments

In most cases, you could say that crates are crates, no matter where they are placed. However, when you consider how the player will interact with them within the game, crates become much more than just crates. The crates inside the rooms of your building not only provide something for the player to hide behind, but also something to climb over or throw. They become part of a player's strategy.

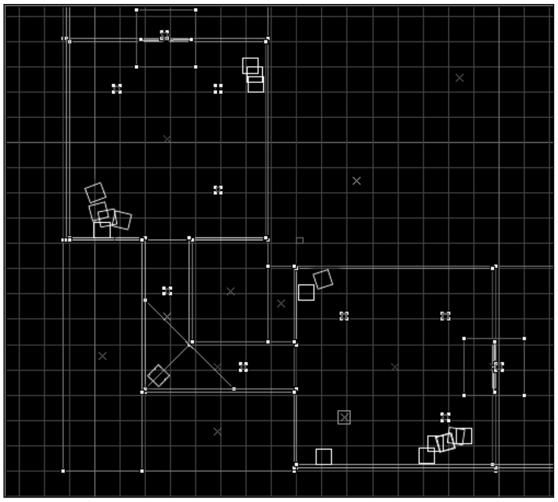


**Figure 12-25:** Continue placing crates throughout your level.

WARNING!

When you place crates outside, you should be careful. You don't want the players climbing the crates to be able to see "behind the curtain." You don't want them to know that there isn't anything behind the wall or allow them to walk along the roof of the building. Think carefully about all the possibilities as you build and place crates in the outside area of your level.

So, when you place additional crates outside of the building, they must provide limitations to the player. Such limitations can be made simply by making the crates larger. But, to make them larger means that you either need a different model or you need to build them with brushes. I chose the latter so that I could insert something custom as you will soon see.

To build your crate from brushes rather than a model, follow these steps:

1. **In the outside area, create a brush with the Nodraw texture. Make it 256 x 128 x 128 (X, Y, Z) units in size, resulting in a large rectangle.**

   Make sure the brush is resting on the ground and that it isn't too close to any of the buildings. You want your player to be able to run around it.

2. **Open the Texture Application Tool and browse for the** `props/metalcrate002d` **texture. Apply this texture to the long sides, the top, and the back.**

   I am considering the front of this large crate as being the side that faces away from the buildings.

   I chose this texture (shown in Figure 12-26) because the goal is to create a container that looks like it belongs on a ship or a tractor trailer. A large metal box with doors on one end is perfect.

Figure 12-26: Apply the texture to all sides but the front of this brush.

When applying the texture, use the alignment buttons in the Face Edit Sheet window and shift the textures as needed. Try to judge by eye the placement of the texture. If it looks good to you here in the editor, then it will look good to you in the game.

3. **For the front of this large crate, apply the** `props/metalcrate002c` **texture.**

   This texture (shown in Figure 12-27) looks like a set of doors that c ompletes the look of this crate.

Figure 12-27: Apply the props/metal crate002c texture to the front of the crate.

4. **Turn this world brush into a** `func_detail` **entity to reduce the splitting of geometry in the game.**

   Select the brush and click toEntity in the New Objects group on the right side of the editor. Make sure the class is set to `func_detail` inside the Object Properties window. Then click Apply and close the window.

Now just make a couple duplicates of this crate. Place one next to the original and the other one in the empty corner of your outside area. Be sure not to place them so close to any walls that the player can't squeeze by. Also, rotate them slightly to give them more of a personal touch.

When you're done, your courtyard should look something like Figure 12-28.

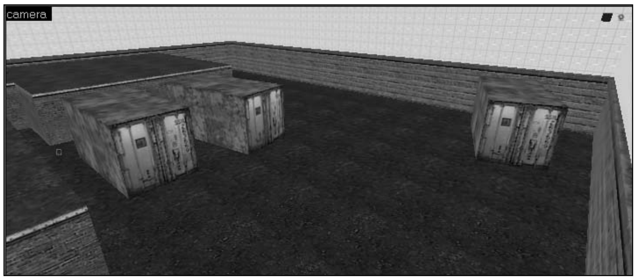


**Figure 12-28:** Your crates should look random even though you placed them with purpose.

## Picking Up on Pickups

You run around the corner in your level while playing against someone else online. As you emerge, your opponent shoots several rounds into your player, and you escape with only a sliver of health left on your health bar. The next thing on your mind as a player is, “Where can I find a health pickup to recuperate and get back in the game?”

*Pickups* are an important part of the game. They range from health and armor to weapons and ammo. In a multiplayer game, pickups are your only means of supplying the player with additional weapons.

Placing these items is very simple. They are entities, and all you need to do to place them in your map is to select one of them as an object when the Entity Tool is active, click in your map, and press Enter.

Selecting a place for your pickups is simply a matter of choice. However, you should consider their placement as you did with the crate. Pickups can be lures to encourage players to access different areas of the map as well as make those players more vulnerable to attack.

### Adding armor

In *Half-Life 2,* your suit is your source of protection. Charging your suit is like adding armor. Start by placing some batteries in the hall between the two rooms. Give players a reason to run though the halls by placing a row of three battery pickups in each elbow of the hall:

1. **Select the Entity tool.**
2. **Select** `item_battery` **as your object in the New Objects group located on the right side of the editor.**
3. **Click in your map to position the entity and press Enter to place it.**

4. **Move the object to the center of the hall and make sure it's positioned on the floor and not above or under the floor.**
5. **Duplicate and place another five of these pickups in a row along each hall.**

   See Figure 12-29 for placement.

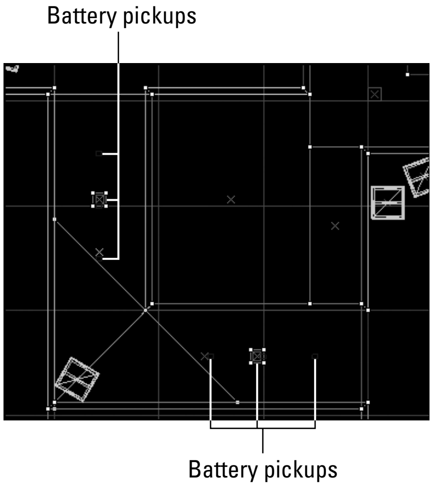


**Figure 12-29:** Place a series of battery pickups in the hallway to encourage the player to enter it.

Now that the players have a reason to run through the halls, you have created a flow for the level. The players have a reason to run from one end of the map to the other end. This encourages interaction between opposing teams and makes for a more enjoyable game. By placing a pickup that restores only partial armor to the players, you are giving them a reason to return for more later when that pickup re-spawns. In multiplayer games, pickups re-spawn a few seconds after being picked up by the player. In single-player games, pickups do not re-spawn.

## Restoring health

You should place some health in the map. The health pickup would do well in a separate place of the map other than that of the armor. This means that the players have to choose between grabbing armor or health when they need it, and options like these open up the level to more interaction between the players and the level.

Start by placing an `info_healthkit` in the outside area of your map:

1. **With the Entity Tool selected, select the `info_healthkit` object within the New Objects group.**
2. **Place one health pickup in the upper-left corner of the map in the outside area by clicking in the 2D viewport and pressing Enter.**
3. **Adjust the pickup's position in the corner so it's on the ground and then rotate it if necessary.**

   Figure 12-30 shows you the placement.

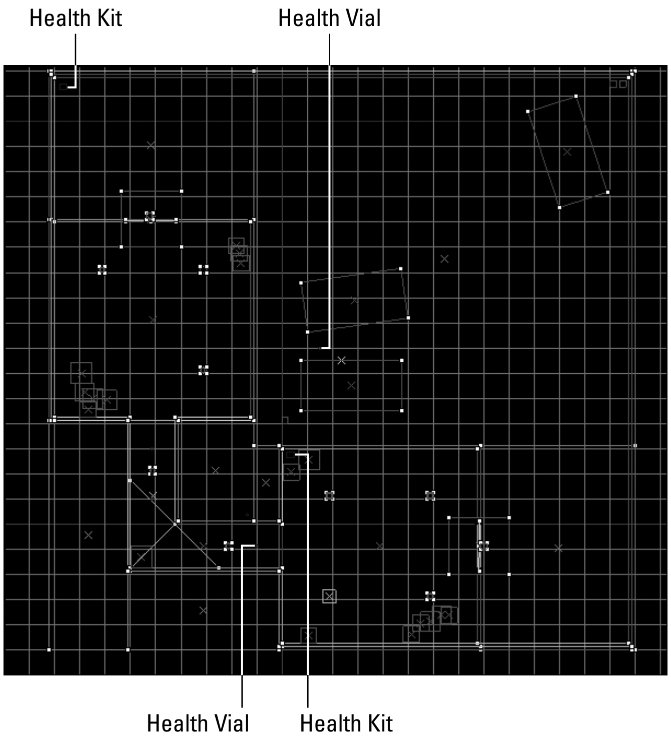


**Figure 12-30:** Place health kits and vials in various places to encourage exploration.

One health pickup is rarely enough. Another good location for a health pickup is behind the crates in the upper-left corner of the South room. Now you are encouraging the player to explore the level for more goodies. Placing them in plain sight is sometimes too easy.

In many games, especially multiplayer games, health is the most important pickup. This is particularly the case in smaller maps because the players are forced to run into each other more often. Rather than placing health kits, this time place a couple health vials. They are smaller and don't replenish as much of the player's health. Place one between the crates in the outside area and one in the outside corner between the two buildings. You can refer to Figure 12-36 for placement.

## Finding weapons

What is a deathmatch without weapons? Unless you're planning on using your level as a picnic area, you need some weapons pickups.

TIP

When placing weapons in a map, consider the proximity at which the players will be fighting. In smaller maps, close-range weapons such as the shotgun become much more valuable to a player than long-range weapons, such as a rocket launcher.

Here's how to add weapons pickups to your map:

1. **With the Entity Tool selected, select the** `weapon_shotgun` **object within the New Objects group.**
2. **Place this weapon in the outside nook between the halls by clicking in the 2D viewport and pressing Enter.**
3. **Select and place the** `weapon_357` **in the lower-right corner of the map, outside the building.**

   Position and rotate as needed so that the pickup is resting on the ground and can easily be seen by the player.
4. **Select and place the** `weapon_smg1` **in the North room in the upper-right corner, behind the crates.**

   Again, position and rotate the pickup so that it rests on the ground and is easy for a player to see.

   You can refer to Figure 12-31 for placement of these weapons.

TIP

Place prized weapons in highly visible areas to make them tougher to obtain. This draws players out in the open and makes for a more interesting battle.

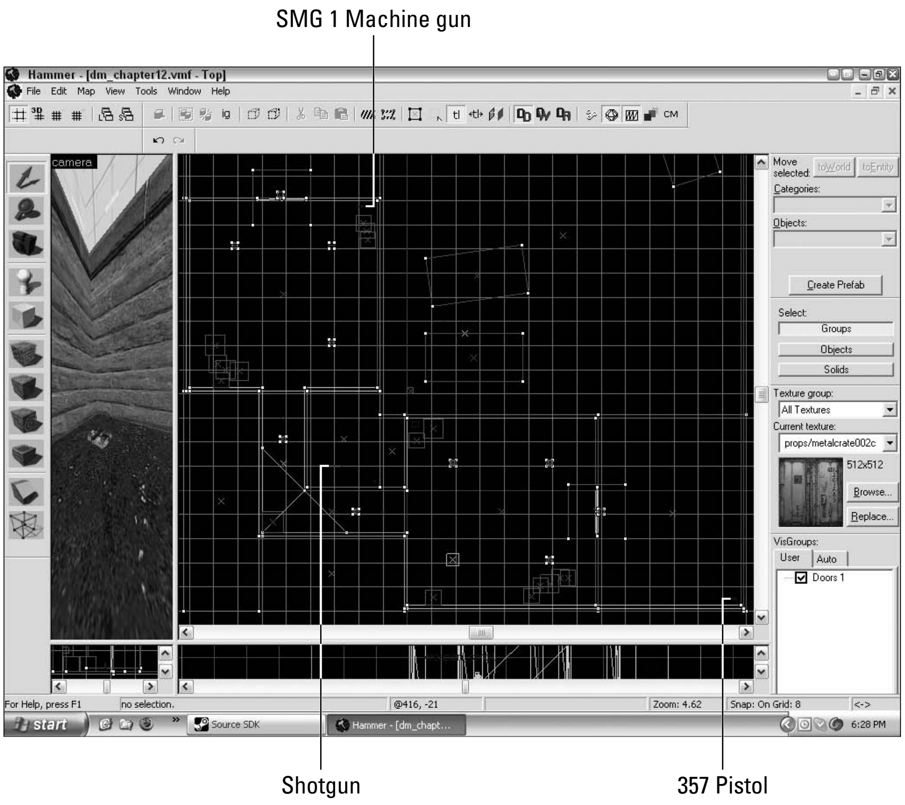


**Figure 12-31:** Place weapons appropriate for your map in various places of the map.

## Grabbing ammo

Now move on to adding some ammo pickups to the map. Without ammo, the weapons are of no use to the players. Ammo is important. You should place ammo in slightly more visible locations to the players to keep the game going and avoid frustrating the player.

So far, all the pickups you've added have been on the floor. This implies that if players are looking for a pickup, they need to look for it on the floor. You want to change that and get players to investigate the level more fully by placing the next pickup on one of the smaller crates:

1. **With the Entity Tool selected, select the** `item_box_buckshot` **object within the New Objects group.**

2. **Place this shotgun ammo on the edge of one of the crates in the North room in the lower-left corner. Place it by clicking in the 2D viewport and pressing Enter.**

   By placing the shotgun ammo far away from the shotgun itself, you force the player to go elsewhere in the level to restock. This also helps the players in case they run out while on the run.

   Make sure the ammo is sitting on top of the crate near the edge, as illustrated in Figure 12-32, so that it's easier for players to pick it up as they run by.

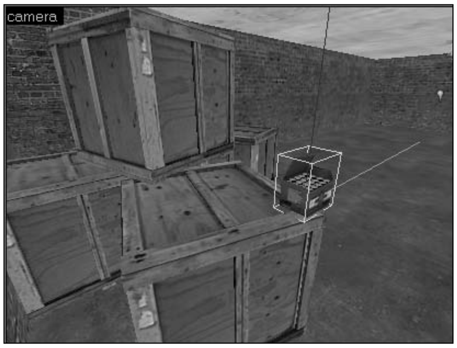


**Figure 12-32:** Place items where they can be reached by the player.

3. **Place some SMG1 machine gun ammo on the crate in the lower-left corner of the South room. The object name of this pickup is `item_ammo_smg1`.**

   This is another place that could use some player action, and adding ammo here helps with that goal.

   When you place the ammo, make sure you move it toward the edge of the crate so the player doesn't have to reach for it in the game. Figure 12-33 shows how I placed it.

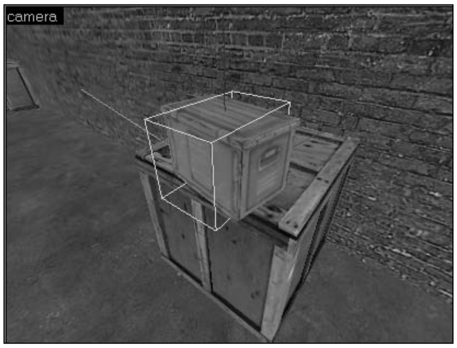


**Figure 12-33:** This ammo is placed on the edge of the crate so the player can grab it.

4. **Place some 357 pistol ammo in the upper-left corner of the North room. The object name of this pickup is `item_ammo_357`.**

   After you've placed it on the floor, angle the entity toward the center of the room for better visibility. Again, this is far away from the location of the 357 pistol and might also spark some interest in a player to go looking for the weapon.

For the ammo pickups mentioned above, refer to Figure 12-34 for placement within the map.

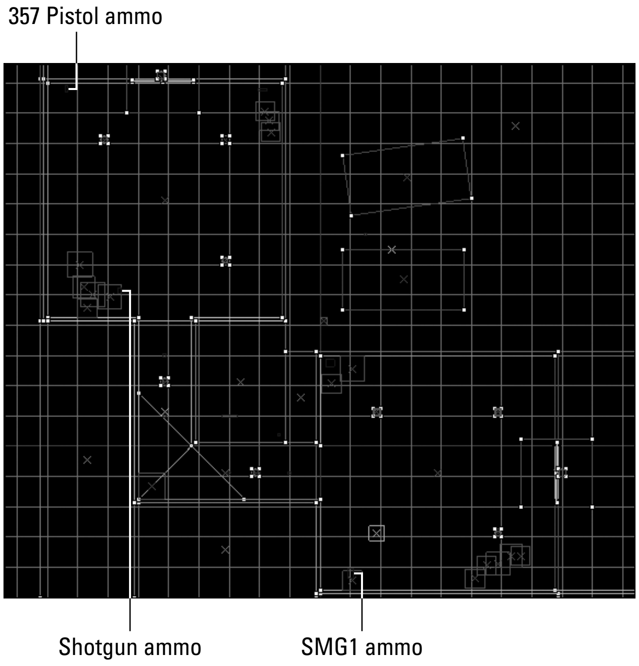


**Figure 12-34:** Place ammo for the available weapons in locations other than where they find the weapon.

## Testing and Having Fun

That takes care of your map for now. You add more to it in the subsequent chapters.

Compile your map by pressing F9 and confirming the setup. When it's done compiling, let it load and run in the game and check it out as a player.

One more thing you might be interested in checking out with your new level is its optimization. When you were placing many of the detailed brushes, such as the crates, light fixtures, and so forth, you turned the brushes into `func_detail` entities. You made this change to avoid splitting your geometry, which would slow down your level. As I discuss in Chapter 10, you can enter the console command to view how the game renders the level:

1. **Open the console by pressing ~.**
2. **Enter the command,** `mat_wireframe 3`.
3. **Press Enter and close the console by pressing ~ again.**

After you enter these commands into the console, your map should display the outlines of everything it is rendering for the player. The results should look like Figure 12-35, with simple geometry and not too much of the level being drawn all at once.

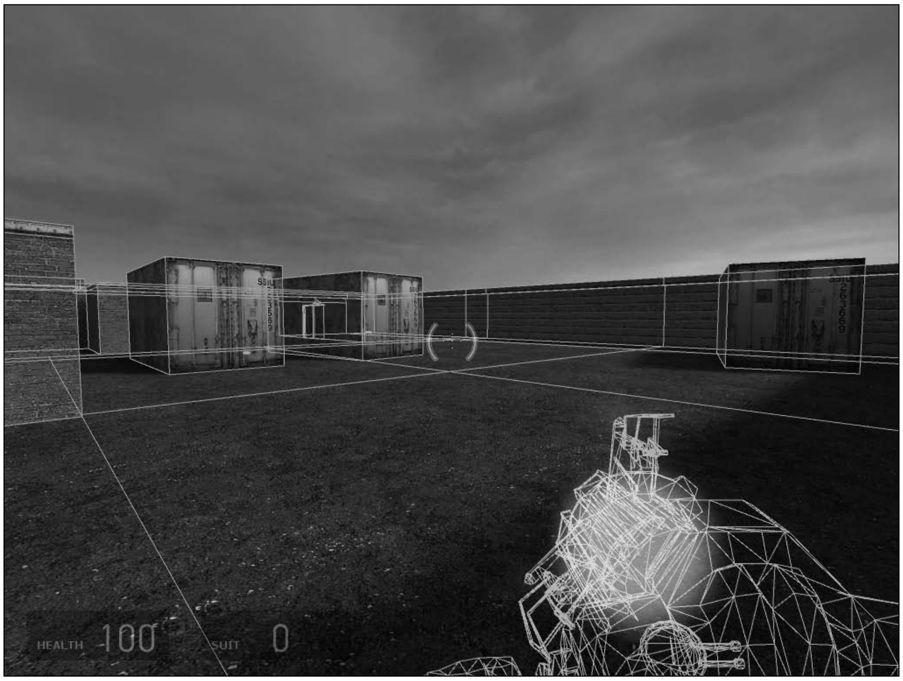
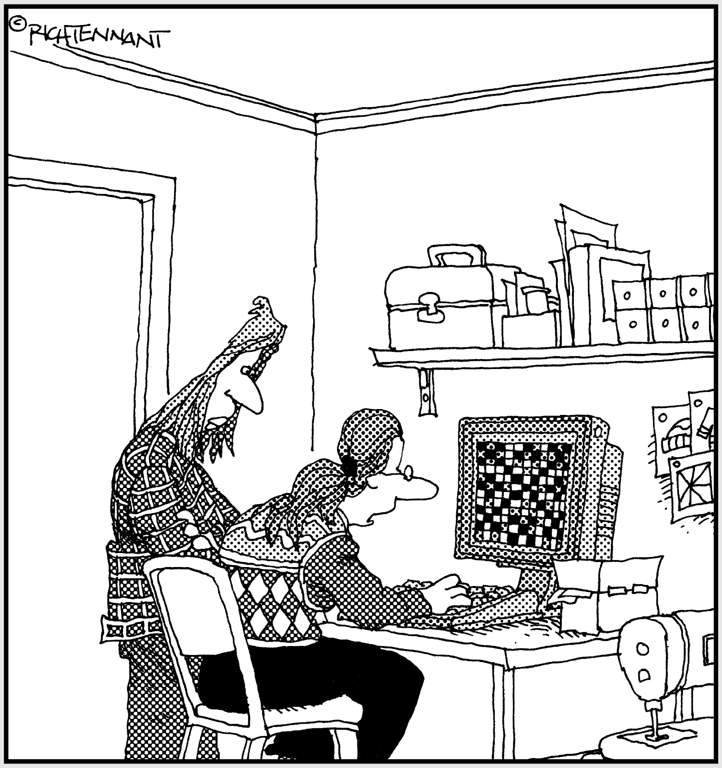


**Figure 12-35:** Your func_detail entities don't split your other brushes, and you retain a well-optimized level.

# Part IV
# Going Beyond the Basics

The 5th Wave By Rich Tennant



"I've used several Half-Life 2 editors, but this is the best one for designing quilt patterns."

## In this part . . .

You can do only so much with the building blocks of a map. To go further, you need to be able to bring your own images into the game — to create your own textures and convert them for use within the game.

Then, after everything is finally in place, it's time to go public with your creation. In this part, I show you how to put everything into an easy-to-use package and how to explain to your audience what you have provided to them and what they can do with it. Get it out there so everyone can see!

Chapter 13

# Creating Custom Textures

*In This Chapter*

▶ Understanding what goes into making a texture

▶ Taking a real-world image and making it tileable

▶ Writing a material file for the game

▶ Putting textures to good use

When I step into the gaming world, I often feel like I'm taken to another place or time. I could be taken to an alien stronghold in outer space, to a grass hut on a paradise island, or even to the homeroom of my old high school.

What makes all these scenes different is not their structures — they're all scenes that take place in a room. Instead, it's the textures. It's the difference between the cold steel of a spacecraft, the dry straw of the hut, and the white paint and chalkboards of my homeroom.

You will find that you can create only so much with the textures that were provided by the game. Eventually you'll want to do more, and that means making your own textures.

The map you've been working on up to this point could be transformed into any setting. As it is, it looks mostly like an abandoned building or garage. With all the brick, the grassy ground, and the fortress-like walls, your map feels very strong and foreboding. You could turn the ground into sand and the building into wood, creating a scene that feels much more dry and barren. On the other hand, you could brighten the lighting, make the building walls look more like clay, and stack bricks around the area, making it feel like it belongs in the Southwestern United States.

When choosing a location, you need to consider what materials you have at hand.

As the perception of reality in games increases, so does the demand for quality textures. The highest quality you can achieve is with actual photographs. Depending on the type of scene you're working with, you can create your own textures with image programs, such as Adobe Photoshop, or modeling software, such as 3D Studio Max. However, nothing beats the look of an actual photograph for modeling something that is real.

With this in mind, you need to consider what photographs you can get your hands on — without breaking any copyright laws, of course. In my case, I can easily walk outside and find plenty of brick walls to take pictures of. So, this is what I use to show you the ins and outs of creating your own textures. I take a picture of a brick wall, turn it into a texture for the game, and then apply this texture to the walls surrounding the outdoor part of the map.

## Slicing Up the Texture

A texture is actually more than just an image that's been brought into a game. A texture also contains information about how the brush it is applied to should react in the 3D world. How that surface reacts to light and whether bumps and grooves are in the image are also defined in the textures.

A texture most commonly consists of multiple images. Each of these images is referred to as a *map* because each image maps out a single aspect of a texture as a whole. These maps include

- **Diffuse map:** This is the color that is shown in the texture. Generally, this is the actual image without any changes.
- **Specular map:** This defines the way light reacts to the texture. Parts of a texture can be shinier than others, and this fact is important in the game world, in which light and shadows occur in real time.
- **Normal map:** This defines how a texture appears to have bumps that point out or in. The bumping effect made here can actually affect the polygons upon which this texture is applied. Although this is a normal map image, the game refers to it as a *bump map.*

To tie all these images together, you create a *material* file called a Valve Material File (VMF). This is a text file that defines the location of each of these images and which image is to be used for which purpose, as well as a few other specifics about how the texture should react in your map.

## Installing Some Tools

Before you start making your own textures, you need to install some tools for the job. The first thing you need is image-editing software. Something as simple as Windows Paint won't suffice. You need a program that can open and edit Targa (TGA) images. Targa is an image format used in *Half-Life 2.*

For the purpose of this chapter, I use Adobe Photoshop. This professional image-editing program, which happens to come with a free trial, has been provided to you on the CD located in the back of this book. If you haven't already, I recommend installing it by following the instructions provided on the CD.

The second tool you need is a special filter plug-in for Photoshop — the NVIDIA Photoshop Plug-ins. This filter takes one of your images and transforms it for use with your custom texture into a *normal map*. I explain how this tool works in more detail later in this chapter. For now, just know that you must install this plug-in to complete your texture. You'll find a copy of the installer on the CD in the back of the book.

The final tool you need is a special converter that comes with the *Half-Life 2* game. This tool takes your Targa image and converts it into something that *Half-Life 2* can use in your maps. This new image format is Valve Texture Format (VTF). If you installed the SDK, you also installed the converter, which you get to use shortly. But before using it, follow a few steps now to set it up:

1. **Open your file explorer program.**

   For me, that program is Windows Explorer, which is opened by choosing Start➪All Programs➪Accessories➪Windows Explorer.

2. **Navigate to the folder** `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp`**, where** *`username`* **is your Steam user name and** *`Valve`* **is optional based on your installation.**

   Because the map you're working on is for the multiplayer deathmatch game type, this is the path you want to explore. Otherwise, you would need to navigate to your specific mod's folder.

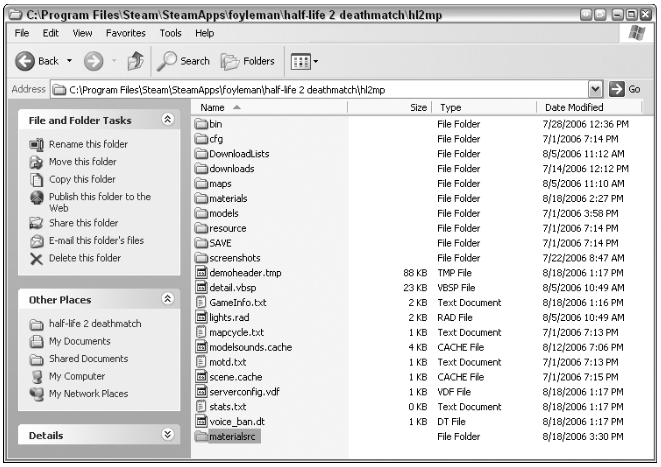
   If you installed the game from the Steam download client, then your installation path will be `C:\Program Files\Steam\Steam\SteamApps\`*`username`*`\`. If you installed the game from disks that came in a box, then your installation path will be `C:\Program Files\Valve\Steam\SteamApps\`*`username`*`\`. Note that the second folder name in the two paths above is different based on your installation.

3. **If it does not already exist, create a folder named** `materialsrc` **within the hl2mp folder, as shown in Figure 13-1.**

   If you're using Windows Explorer create a new folder by choosing File➪New➪Folder. A new folder aptly named `New Folder` appears, ready to be renamed. Simply type **materialsrc** to rename it.

4. **Create a shortcut on your Desktop to the program** `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\ sourcesdk\bin\vtex.exe`**.**

   Navigate to the folder and file above. Right-click `vtex.exe`, and select Send To➪Desktop (Create Shortcut).


C:\Program Files\Steam\SteamApps\foyleman\half-life 2 deathmatch\hl2mp
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Program Files\Steam\SteamApps\foyleman\half-life 2 deathmatch\hl2mp Go
File and Folder Tasks
Rename this folder
Move this folder
Copy this folder
Publish this folder to the Web
Share this folder
E-mail this folder's files
Delete this folder
Other Places
half-life 2 deathmatch
My Documents
Shared Documents
My Computer
My Network Places
Details

| Name | Size | Type | Date Modified |
|---|---|---|---|
| bin | | File Folder | 7/28/2006 12:36 PM |
| cfg | | File Folder | 7/1/2006 7:14 PM |
| DownloadLists | | File Folder | 8/5/2006 11:12 AM |
| downloads | | File Folder | 7/14/2006 12:12 PM |
| maps | | File Folder | 8/5/2006 11:10 AM |
| materials | | File Folder | 8/18/2006 2:27 PM |
| models | | File Folder | 7/1/2006 3:58 PM |
| resource | | File Folder | 7/1/2006 7:14 PM |
| SAVE | | File Folder | 7/1/2006 7:14 PM |
| screenshots | | File Folder | 7/22/2006 8:47 AM |
| demoheader.tmp | 88 KB | TMP File | 8/18/2006 1:17 PM |
| detail.vbsp | 23 KB | VBSP File | 8/5/2006 10:49 AM |
| GameInfo.txt | 2 KB | Text Document | 8/18/2006 1:16 PM |
| lights.rad | 2 KB | RAD File | 8/5/2006 10:49 AM |
| mapcycle.txt | 1 KB | Text Document | 7/1/2006 7:13 PM |
| modelsounds.cache | 4 KB | CACHE File | 8/12/2006 7:06 PM |
| motd.txt | 1 KB | Text Document | 7/1/2006 7:13 PM |
| scene.cache | 1 KB | CACHE File | 7/1/2006 7:15 PM |
| serverconfig.vdf | 1 KB | VDF File | 8/18/2006 1:17 PM |
| stats.txt | 0 KB | Text Document | 8/18/2006 1:17 PM |
| voice_ban.dt | 1 KB | DT File | 8/18/2006 1:17 PM |
| materialsrc | | File Folder | 8/18/2006 3:30 PM |

**Figure 13-1:** Create the folder materialsrc to hold your custom textures.

## Bricking Up the Joint

I took a picture of a wall that will work well as a texture. In Figure 13-2, you can see that my picture has well-defined features. There is nice contrast between the bricks that are sticking out and the joints between the bricks that sink in. These aspects make for a good texture.

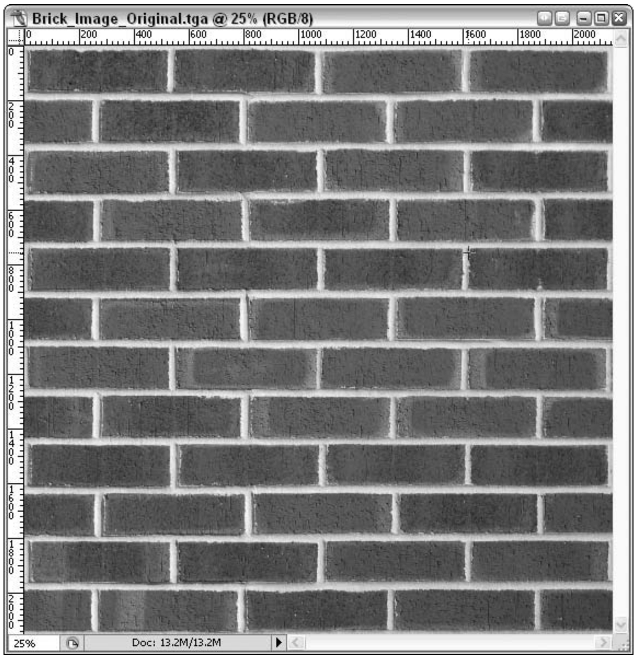


**Figure 13-2:** A well-taken picture often makes for a good-looking texture.

Any image can be turned into a texture. You don't need to choose one with contrasting highs and lows. This brick picture was chosen to make it easier to illustrate the methods used in making a custom texture.

If you have your own image that you want to use, open it in Photoshop now. Otherwise, use the same brick image titled `Brick_Image_Original.tga` that I provide on this book's CD.

Start the Adobe Photoshop program. After the program has finished loading, open `Brick_Image_Original.tga`. Choose File⇨Open, locate the file you want to open in the File Explorer window, and then click Open.

## Finding the Right Size

To begin making your custom texture, you need to determine the final size of the image. *Half-Life 2,* as well as most games, requires that the width and height of your image in pixels evaluate to a power of two. Basically, this means that your texture must be some combination in width and height using, but not limited to, the following numbers:

8, 16, 32, 64, 128, 256, 512, 1,024

The reason for the size restriction is that the game needs to resize these images while you're playing the game. The images are scaled down to half or quarter sizes, and the game can do this only if the sizes are divisible by two or four.

While you're playing, the game displays smaller or larger versions of the same image, based on your distance away from the textures. This is called Level of Detail (LOD) and helps reduce the amount of computing power required to display all the imagery on the screen at once.

When determining the size of your texture, you need to consider two things:

- **You don't want to take up all the computer's memory with one texture.** Each texture in the game is loaded into your memory before the game starts. If your textures' files are too big, the game could run out of memory and crash.
- **There is no reason to make a texture larger than what is used within the game.** That would be like buying an 8 × 10 photo for your wallet — it's a waste of space.

Take both of these considerations into account to create an optimally sized image. The trick to determining the best size is to know where this texture will be used in the game. If you're making a texture to be placed inside your buildings in the game, take note of the height of your walls. This height in mapping units should be equal to or less than the height in pixels for your image.

The wall for which you're going to create a texture is the wall that surrounds the outside of the buildings, as shown in Figure 13-3. The visible height of this wall is 128 units, which is an allowable texture size because it's a power of two. However, some experience plays a role at this point.

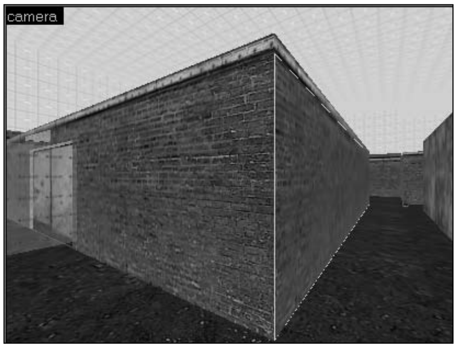


**Figure 13-3:** To help determine the size of your texture, consider where it will be applied in the game.

Keep a few details in mind when you decide on the size of your texture:

- **No texture other than a sky texture should ever be 1,024 pixels in width or height.** This won't be an issue in this circumstance.
- **The bigger an image, the clearer the details.** When you shrink an image, you lose some of the details that make it look so great. The less shrinking you do, the better it looks.
- **The larger the image, the more memory your computer requires to display it in the game.** You don't want to slow down players' computers by making them load and display a texture in huge files.

Knowing these facts, you need to find the balance that makes your image look great in the game and allows the game to run great with your image. This is something that you might have to experiment with at first. However, as you work more frequently with custom textures, your experience will help you make the right choices.

Based on my experience with textures, I recommend that you make your image 256 pixels high. This is twice the exact size of the wall in your map, and it will provide a very nice-looking and nice-running image in the game.

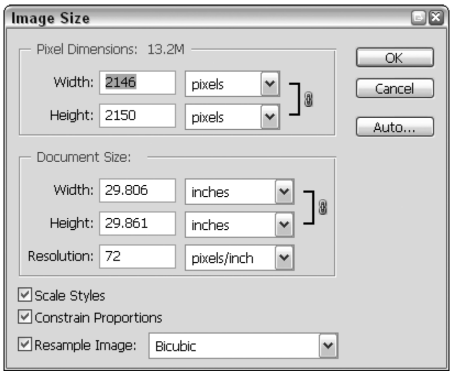
Now, it would make sense to go ahead and resize your image to the new size. However, you need to work with the image and create the different pieces I list earlier (see the section "Slicing Up the Texture") that make up the final texture. It's much easier to work on a larger image and then resize it to its final size just before saving it. Therefore, instead of resizing the image to 256 pixels high, resize it to a multiple of that number closest to its original height. The image I provided, `Brick_Image_Original.tga`, is 2,150 pixels high. The integer power of 2 that is closest to 2,150 is 2,048 (256 × 8). So resize the figure to 2,048 pixels high.

As for the width, looking at the brick texture provided, it is pretty close to being square. Therefore a width equal to the height of the image would be ideal. If the image were wider, such as 3,000 pixels wide, the image would probably look better when sized as a rectangle because there would be less distortion after the resize. Since the image is going to be square, the result will be an image sized to 2,048 × 2,048 pixels. After you make some changes, it can later be saved to 256 × 256 pixels.

Here's how you resize the `Brick_Image_Original.tga` image in Photoshop:

1. **Choose Image➪Image Size.**

   This opens a dialog box, as shown in Figure 13-4.



Pixel Dimensions: 13.2M
Width: 2146 pixels
Height: 2150 pixels

Document Size:
Width: 29.806 inches
Height: 29.861 inches
Resolution: 72 pixels/inch

☑ Scale Styles
☑ Constrain Proportions
☑ Resample Image: Bicubic

**Figure 13-4:** When sizing your image for editing, keep it large so as not to lose details.

2. **In this dialog box, make sure the Constrain Proportions check box is deselected.**
3. **Select pixels as the unit of measurement to the right of the Height and the Width fields.**
4. **Enter** 2048 **into the Height field and** 2048 **into the Width field.**
5. **Click OK to apply the changes.**
6. **Choose File➪Save to save your resized image.**

You should now have a pretty nice looking image that's close to being ready for the game.

## Tiling on Forever

In some cases, you use your texture for only a single application. Perhaps it's a new door, the side of a crate, or some other structure that doesn't require the image to repeat. You create this texture, apply it to the brush face, and then click Fit to properly adjust the image.

However, in most cases, you want your texture to repeat endlessly. For the case of a brick wall, it is best to create one smaller image that can repeat over and over again as it goes on down the wall. Then you don't need to make a humongous texture to fit the long wall. This act of repeating is *tiling.* Just like the tiles in your bathroom or kitchen, the tile is applied, and it repeats until the entire surface is covered.

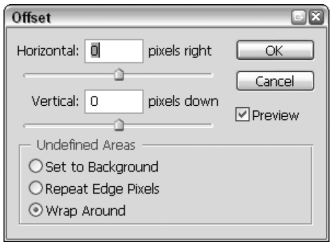
### Shifting the image to expose the seam

Making the image tile is easy. The game can do this on its own. When you apply your texture in the mapping editor, the image repeats automatically if the surface you apply the texture on is larger than the texture itself.

Making this image tile without the player seeing where one tile ends and the other tile begins — that is the trick. So, what I want to show you now is how to create a tileable image in Photoshop.

1. **In Photoshop, choose Filter➪Other➪Offset.**

   The Offset dialog box pops up, as shown in Figure 13-5. This filter allows you to adjust the image so you can see where the seams exist.


Offset

Horizontal: 0 pixels right

Vertical: 0 pixels down

[✓ Preview]

Undefined Areas

Set to Background

Repeat Edge Pixels

[✓ Wrap Around]

**Figure 13-5:** The Offset filter in Photoshop is perfect for finding and fixing seams in tileable images.

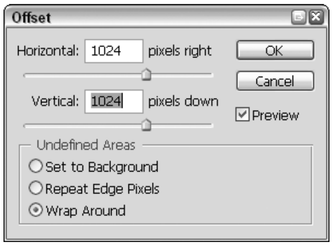
2. **Select the Preview check box (if it isn't already selected) and select the Wrap Around option.**

The Preview option shows you any changes you make in advance, before you apply them, and Wrap Around is the key to using the Offset filter.

The purpose of the Offset filter is to shift the image horizontally and vertically. It's like pushing the texture up or to the side. When you allow the image to wrap around during this process, it's like you're wrapping a package. One end starts in the middle of the package, and the other end wraps around the package until it meets up with the starting end. Thus the term *Wrap Around*.

3. **Because the image you're working with is 2,048 pixels wide, enter a value equal to half that — 1,024 — in both the Horizontal and Vertical text fields.**

   The image adjusts itself (because you have Preview selected). The points at which the start and end of the image meet while wrapping will be right in the center of your screen. The result before applying the change looks like Figure 13-6.



Horizontal: 1024 pixels right
Vertical: 1024 pixels down
[✓ Preview]
Undefined Areas
Set to Background
Repeat Edge Pixels
[✓ Wrap Around]

**Figure 13-6:** Adjust your image with the Offset filter by half its total size both horizontally and vertically.

4. **Click OK to apply the changes created with the Offset filter.**

   You might have a hard time seeing the seam running through your image, but it's there. In this case, I provided you with an easy image to work with that doesn't require a lot of adjustment. However, when you zoom in really close to the middle of the picture, you see the seam. This will look even worse when tiled in the game. Figure 13-7 shows a close-up of this mismatch.

   The goal now is to get rid of this seam in the middle of your texture. That way, when it is tiled in the game, you won't see where one tile begins and one ends.

5. **Zoom in so you can clearly see the seams.**

   To zoom your image, use the Navigator window either by entering the zoom percentage or by sliding the scale rule. The Navigator window is usually in the upper-right corner of the screen and is displayed in Figure 13-8. It is often the simplest way to zoom in on your image.

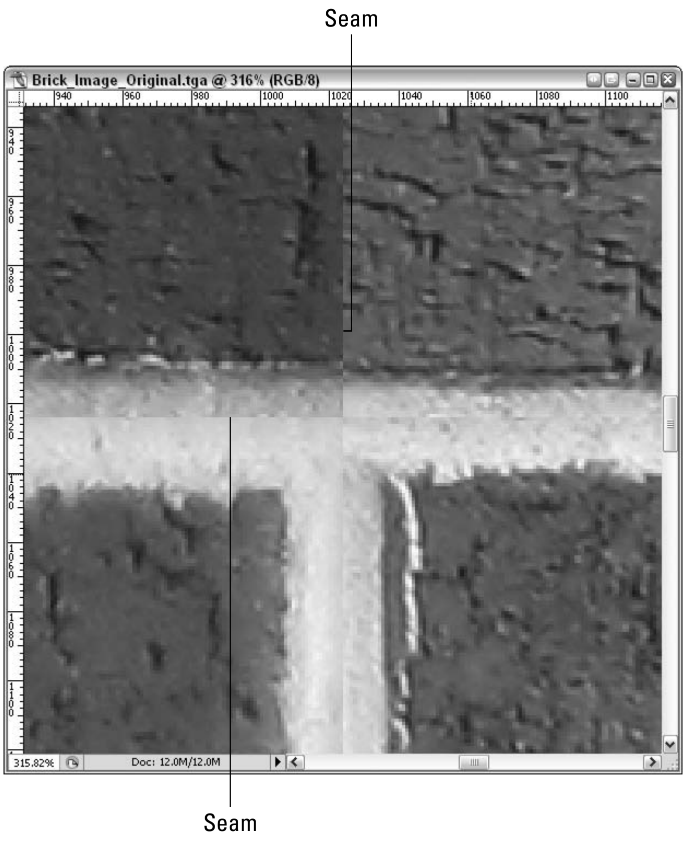


**Figure 13-7:** After applying the Offset filter, zooming in close shows the seam.

Now that you can see the seams, you can start getting rid of them. I show you how in the next section.

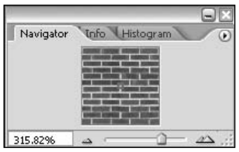


**Figure 13-8:** Move around a large image in Photoshop via the Navigator tab.

## Stamping out your seams

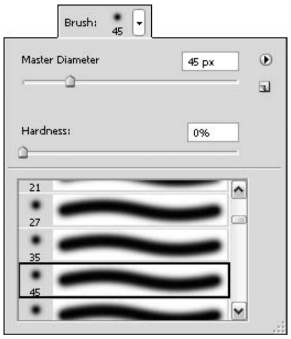
Now that you can see the details of the image and where the seam lies, hide that seam by using a Photoshop tool called the Rubber Stamp. You use the Rubber Stamp to copy a selected area of the image and apply it to another area. Think of the selected part as the ink for your stamp; you just stamp that ink over your seam to make it disappear.

1. **Select the Rubber Stamp tool from the toolbar on the left side of the screen.**

   The tool's icon looks like a little rubber stamp, as shown in the margin.

   Next you need to select a brush size.

2. **On the toolbar at the top, you can see the word Brush. To the right of this word is a dot, a number, and a small arrow. Click this arrow to open a menu, as shown in Figure 13-9, and then move the slider to the right of the curvy images until you locate the one labeled 45. Select it, and the window closes.**


Brush: 45

Master Diameter: 45 px

Hardness: 0%

**Figure 13-9:** Select an appropriate brush size for your rubber stamp.

The brush size determines how much ink the stamp selects to be applied later. A good brush size for this image is 45. It's about the same size as the mortar between the bricks, which means the Rubber Stamp tool can stamp over half the mortar while overlapping the brick slightly. This results in a nicely blended stamping, as you see soon.

With the tool ready for use, it's time to select the ink for your stamp. This tool copies a portion of your image as if your image were ink in a pad and then pastes that copy anywhere you want. After you stamp down your first copy, you can continue to stamp on down the seam, and the selection you copied from moves with you. If this sounds confusing, it makes more sense when you use the tool. If you make a mistake, you can simply undo your last few operations by pressing Ctrl+Z.

3. **Place your mouse pointer, which now looks like a circle, exactly one brick to the right of the seam you want to cover and Alt+click.**

   Use Figure 13-10 as a reference. This is your selection that is pasted over the seam. Try and be as precise as possible in the placement of your stamp tool. You want your copied section, which you are selecting now, to be pasted between the bricks where the seam is located. This pasting action is the next step.

4. **Carefully move your cursor over the seam in the image in a straight line from your selection point to over the seam. Then click to apply the stamp.**

   In that one small spot where you applied the stamp, your seam will now be hidden.

5. **Continue to click the seam while you move up and down the image, as I have started to do in Figure 13-11.**

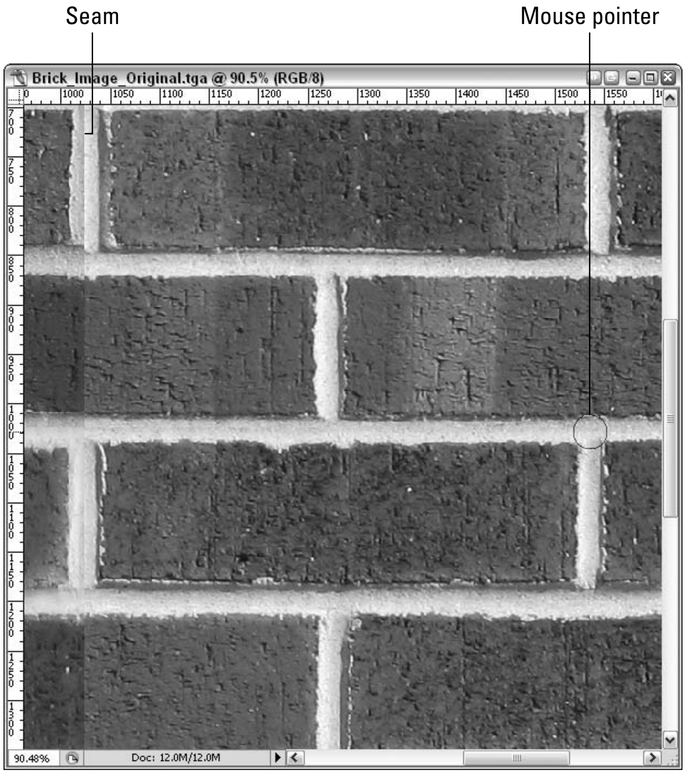


**Figure 13-10:** The rubber stamp mouse pointer looks like a circle.

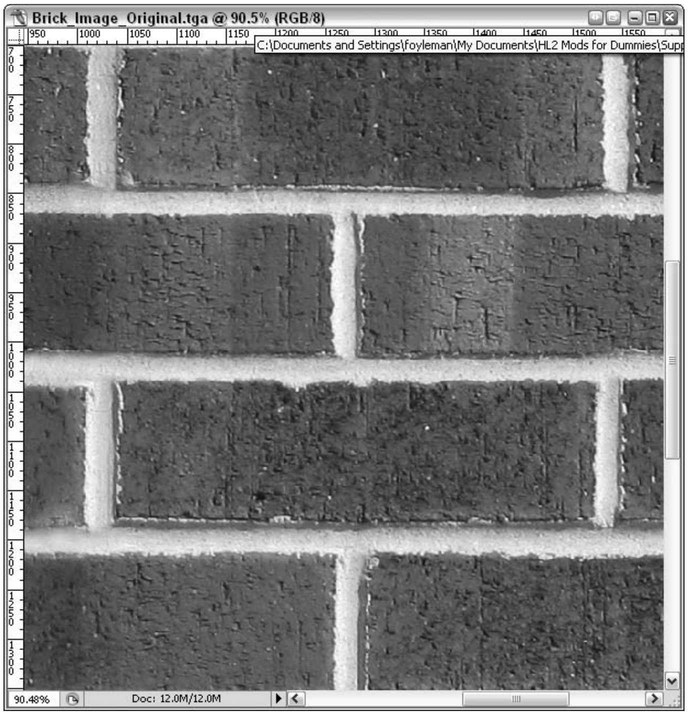

Brick_Image_Original.tga @ 90.5% (RGB/8)
C:\Documents and Settings\foyleman\My Documents\HL2 Mods for Dummies\Supp
90.48%
Doc: 12.0M/12.0M

**Figure 13-11:** After you apply the rubber stamp to your image, the seam disappears.

Your original selection position for the rubber stamp moves in relation to your mouse, so you don't have to reselect the part you want to copy. You can also click and drag the cursor, and the application of the stamp follows you. This is great if you have a steady hand while dragging across the image. However, you can stick with individual clicks if you want to be more precise.

REMEMBER

As you apply the Rubber Stamp selection over the seam, you might come across areas that don't look good even after you stamp. Just undo your last application, make a new rubber stamp selection (by Alt+clicking a new area), and apply that selection appropriately.

6. **When the seam can no longer be seen, you're ready to move onto the next operation.**

TIP

When using the rubber stamp, I constantly reset the position from which I copy. I'm somewhat of a perfectionist in my mods and would rather take the extra time making the texture as perfect as I can make it.

Take your time stamping out the seam in your image. Remember that the seams run both vertically and horizontally and that both need to be cleaned up. It might take some time to get it just right, but it's worth it. It looks better in the game. When stamping out the seam that runs horizontally, take your copied selection from above or below the seam. You don't want to copy and then paste the same seam that you are trying to hide.

As you move along hiding your seam, occasionally zoom out so that you can see the image as a whole. Inspect it for any seams that you may have missed. If you did miss any seams, go back and stamp them out with the rubber stamp tool. Otherwise, you are ready to continue below.

## Shifting back to ground zero

When the seam is completely stamped out, return the image to its original position.

TIP

In the case of this image, offsetting the image back to its original position isn't required, but it is a good practice. When you return the image to the starting position, you might notice something that looks out of place and needs more touching up before you move on.

Here's how you set your tileable image back to its original position:

1. **Choose Filter⇨Other⇨Offset.**

   Because you shifted your image exactly half its width, all that's left is to shift it again that same distance.

2. **Make sure the Horizontal and the Vertical entry is** 1024 **pixels, as it was before.**

3. **Click OK to apply the change.**

   Now your texture can be tiled, and you won't see any seams where one tile ends and the next begins.

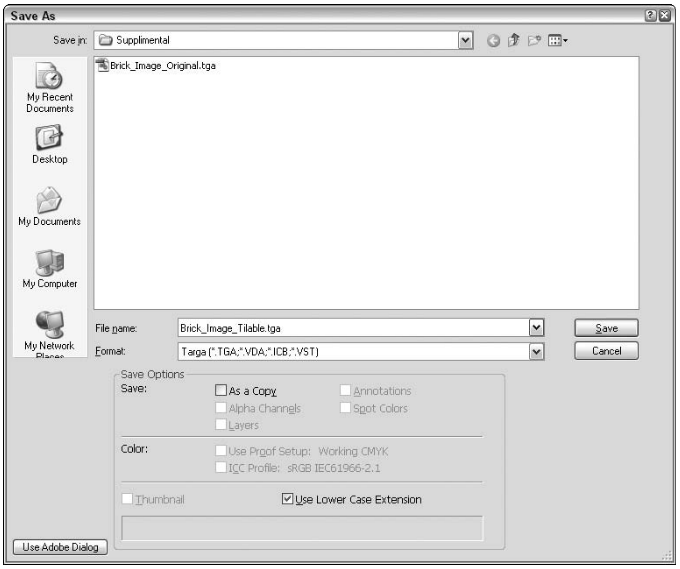
   Before moving on, save this image someplace safe. For the texture, you will be transforming this image in different ways, and it's a good idea to have the original at hand in case you want to go back and make some more changes later on.

4. **Choose File⇨Save As and find a good place on your hard drive to keep it.**

   The game isn't going to be using this image — it will use the resized version instead — so where you save it is not important. Also, note the location where you save the image because you may need to load it again within the image editing program — and knowing where you saved it will help you find it at that time.

5. **Select Targa from the Format drop-down list.**

   JPEG and GIF file formats compress your image, which can lower the detail and quality. Instead, choose Targa from the drop-down list located under the File Name field, as shown in Figure 13-12. This is a good-quality format and happens to be the same format that will be used for the final images used in the texture.



Save in: Supplimental

Brick_Image_Original.tga

File name: Brick_Image_Tilable.tga

Format: Targa (*.TGA;*.VDA;*.ICB;*.VST)

Save Options

Save: ☐ As a Copy ☐ Annotations ☐ Alpha Channels ☐ Spot Colors ☐ Layers

Color: ☐ Use Proof Setup: Working CMYK ☐ ICC Profile: sRGB IEC61966-2.1

☐ Thumbnail ☑ Use Lower Case Extension

**Figure 13-12:** After you have the image adjusted, save it under a new name for safekeeping.

6. **Click Save.**

   You're prompted with a selection of Targa options.

7. **Select 24 bits/pixel and click OK.**

TECHNICAL STUFF

A 24-bit Targa image is the high-quality image size for this file type. A 16-bit image has lower quality because it limits the number of colors used in the image; 32-bit images are reserved for images that have transparency.

## Mapping Your Images

The image is now ready to be transformed into the different pieces that make up the texture. You need to make five different images from this one, each to serve its own purpose. Later, the game will combine each of these pieces and present the player with the final texture.

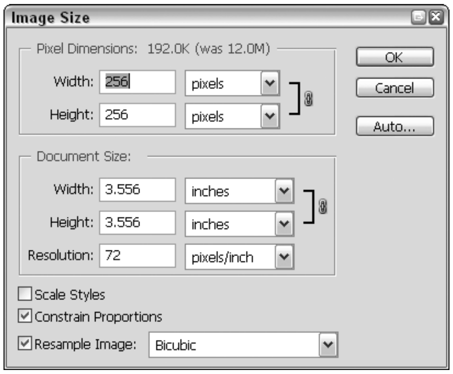
## Splashing on some color: The diffuse map

The first image you need to create is the *diffuse map.* This image represents the color for your texture and is really nothing more than the image you now have displayed in Photoshop. All you have to do is resize it to its final size and save it.

1. **Choose Image➪Image Size within Photoshop.**
2. **In the dialog box that pops up, as shown in Figure 13-13, select the Constrain Proportions check box.**

   This means that when you change one size value, the other changes in the same proportion.
3. **Enter a width of 256 pixels.**

   This size is the goal set forth at the beginning of the chapter. Because you have Constrain Proportions selected in the bottom of the dialog box, the Height changes automatically to 256 pixels as well.
4. **Click OK to apply the size change.**



Pixel Dimensions: 192.0K (was 12.0M)
Width: 256 pixels
Height: 256 pixels

Document Size:
Width: 3.556 inches
Height: 3.556 inches
Resolution: 72 pixels/inch

☐ Scale Styles
☑ Constrain Proportions
☑ Resample Image: Bicubic

**Figure 13-13:** Resize the image.

The image appears pretty small on your screen after you apply the above changes, but that's also because you're zoomed way out. If you enter the numbers as I outline, everything is as it should be.

5. **To save the file, choose File➪Save As.**

   You have to save this image and each after it in a specific location so that the game can find and use these files.
6. **In the Save As dialog box, access the folder** `C:\Program Files\` *`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\`

`hl2mp\materialsrc`, where *`username`* is your Steam user name and *`Valve`* is optional based on your installation.

In the materials folder that you just navigated to, you can create your custom texture folder. This folder can have any name that you desire; however, I recommend that you name it the same as your map file. This way, when you distribute your map, this folder and the files within it are less likely to be overwritten by another map with the same name because it's unlikely that your map has the exact same name as someone else's.

7. **Because the name of this map will be `chapter13.vmf`, create a new folder called `chapter13` and then open it.**

   Click the Create New Folder icon to create a new folder in this directory. Then change the name of this highlighted `New Folder` to `chapter13`. Then access this folder by double-clicking it. In this new texture folder, which is now `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materialsrc\chapter13`, you can save your diffuse map.

8. **Select Targa from the Format drop-down list and name your map `brick_diffuse.tga`.**
9. **Click Save to save the file.**
10. **When prompted for the Targa resolution, select 24 bits/pixel, and you're done.**

Later, when you convert your image from the Targa format to the VTF format, you'll need another folder named `chapter13`. This folder needs to be in `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materials`. You could create this folder later when you're ready to perform the conversion; however, I recommend that you do it now so you don't forget.

1. **Open your file explorer program of choice.**

   I use Windows Explorer, accessed by choosing Start➪All Programs➪Accessories➪Windows Explorer.

2. **Navigate to the folder `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materials`.**
3. **Create a new folder named `chapter13`.**

   In Windows Explorer, choose File➪New➪Folder. A new folder is created and highlighted so you can rename `New Folder` to `chapter13`.

Now you can close your file explorer. The folder is created and ready to receive your converted image files later when the time comes.

## Picking out the highlights: The specular map

The second image you need to create is the *specular map.* This image defines how the light reacts. Some parts of the image might be shinier than others, but some might not reflect any light at all. By making shiny parts of your image white and dull parts black, you can tell the game how to reflect the light off your texture.

TECHNICAL STUFF

The specular map image can be applied to your texture in two different ways. Depending on whether or not you plan to include a normal map — in the next section, "Bumping polygons in and out: The normal map" — within your texture determines how this should be done. In either case, I recommend making the Targa image outlined below. Later, this Targa image can either be combined with your normal map or used separately.

Before proceeding, you need to go back to the larger version of your image. The one open in Photoshop now has been scaled down to 256 × 256 pixels.

To go back to the previous version, you have two options. You can close the one you have and open the high-quality version that you saved earlier, or you can go back in the history of your alterations in Photoshop to a point before you scaled the image down in size. I show you the latter.

Photoshop has the ability to save the history of changes made to an image. Look on the right side of the program and find a tab labeled History, as shown in Figure 13-14. If you find it, click it; otherwise, choose Window➪History to display it. On the right side of this History tab, you can scroll through the changes that you made to your image. Select the history item that comes just before *Image Size.* That is the image state before you change the size of the image.

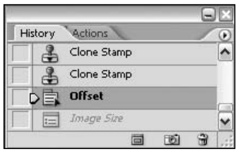


**Figure 13-14:** Use the History tab to go back to a previous stage of the image.

Now that you're back to the original image, you can continue making a new image map.

Brick isn't normally very shiny. Looking at the picture, what you can assume about the way light reacts to the brick is that any spot where you see black coloring is a spot that isn't getting any light. This is where there is either too much shade on the objects in the picture or there is no shiny quality about it. These black spots actually appear to absorb light and are therefore dull. The rest of the image is just flat in terms of light. The image doesn't reflect light and it doesn't absorb light.

So, you need to tell the game that all the black areas of the texture are dull and that the rest of the image is neither shiny nor dull, but rather flat. With Photoshop, this is easy to do.

1. **Zoom in 125 percent to the image in Photoshop.**

   The "Shifting the image to expose the seam" section offers several zooming methods.

   You can tell when you're at 125 percent by either looking at the title bar on your image's window or by looking at the Navigator window. Both show the zoom percentage, as shown in Figure 13-15.

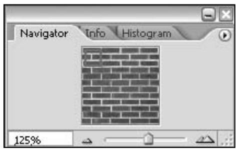


**Figure 13-15:** Zoom in to your image so that you can see the distinct colors of your brick.

2. **Choose Select⇨Color Range.**

   A dialog box opens, and an eyedropper replaces your mouse pointer. You can use this eyedropper either in the box shown in this small dialog box or on the image you're working on in Photoshop. The idea is to select the black color in your image. This selection will then be translated to the game as the dull area of the texture.

3. **Before selecting the black color, change the Fuzziness value to 150 by adjusting the slider near the top of the window.**

   This is the percentage of color to be selected. Because there is so little black in your image, 150 percent results in a better selection.

4. **Use the eyedropper on the image that is currently behind the Color Range dialog box by pointing and clicking the image with your eyedropper.**

   Because you zoomed in, you can more easily select the black color in your image.

   When you select the black color, your selection should be displayed as white in the small dialog box, as it is in Figure 13-16.

5. **Click OK to make your selection and close the dialog box.**

   All the black in your image is selected. The selection outlines in your image might even look like dancing ants. You now need to use this selection to your advantage.

6. **Create a new layer in Photoshop.**

   In the bottom-right corner of the editor is a tab labeled Layers. (If this tab isn't opened, select it now.) At the bottom of this tab is a group of icons. The second icon from the right looks like the one shown in the margin. You can use this icon to create new layers in Photoshop. Click it once.

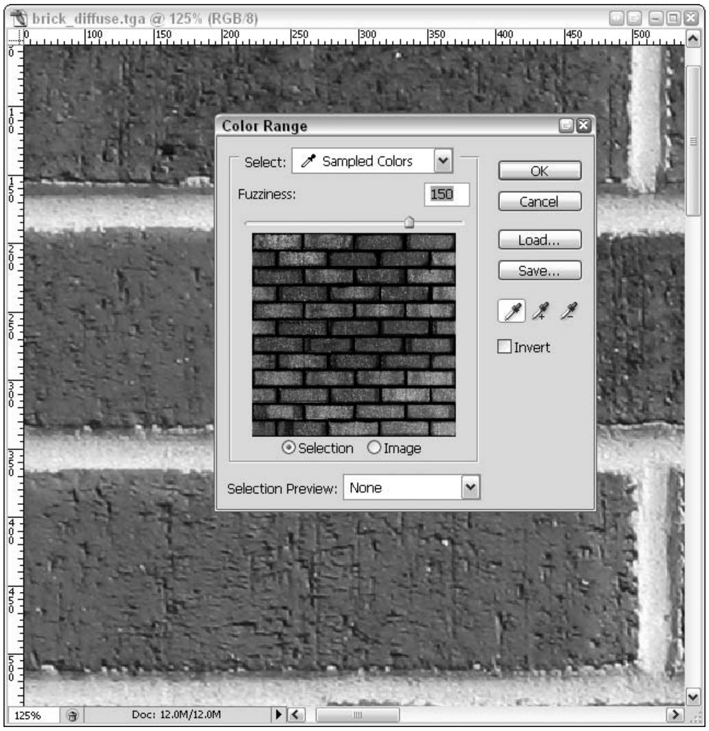


**Figure 13-16:** When you select black, it appears as white in the Color Range preview window.

When you add a new layer, it is added to the list of available layers, as shown in Figure 13-17. The layer is named Layer 1, but the name isn't important. What's important is that you can add more imagery to this layer that will lay over the existing layer. That's because it is listed above the other layer called *Background.*

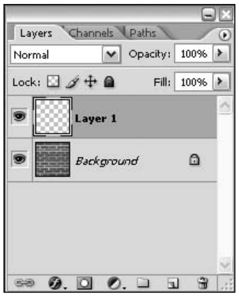


**Figure 13-17:** Create a new layer on the Layers tab.

7. **Choose Edit➪Fill.**

   A dialog box, like the one in Figure 13-18, opens. In this dialog box, you can define which color fills in your selected area.

8. **From the Use drop-down list, choose Black and then click OK.**

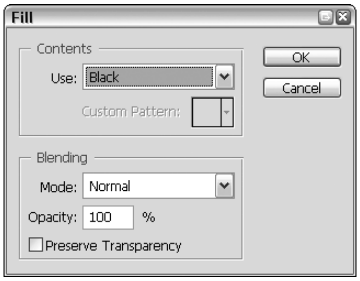


**Figure 13-18:** Fill your selection with black.

You might not see much of a difference yet, but it's coming.

9. **Deselect everything. You can do this either by choosing Select➪Deselect or by pressing Ctrl+D.**
10. **In the Layers tab, select the Background layer.**
11. **Add another new layer by clicking the Create a New Layer icon.**

12. **Fill this new layer with a flat, gray color by choosing Edit➪Fill, selecting 50% Gray as your fill color, and then clicking OK.**

    A level of 50 percent gray is considered neutral in the game, and anything that is black won't be shiny, whereas anything that is white will be shiny. The result looks like Figure 13-19.

    When you present this to the game as the specular map, each of these colors tells the game something specific. Anything that is 50 percent gray is basically ignored. This gray color is considered neither shiny nor dull. Anything that is black is considered dull. Anything that is white, if white is in this image, would be considered shiny. That leaves all the shades of gray in between that represent different levels of shininess.

    All that's left is to resize and save it for the game.

13. **Resize the image by choosing Image➪Size and entering a width and height of 256 pixels. Click OK to apply the change.**
14. **Choose File➪Save As and make sure that the same game folder is still selected.**

    That folder is `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materialsrc\chapter13`, where *`username`* is your Steam user name and *`Valve`* is optional based on your installation.

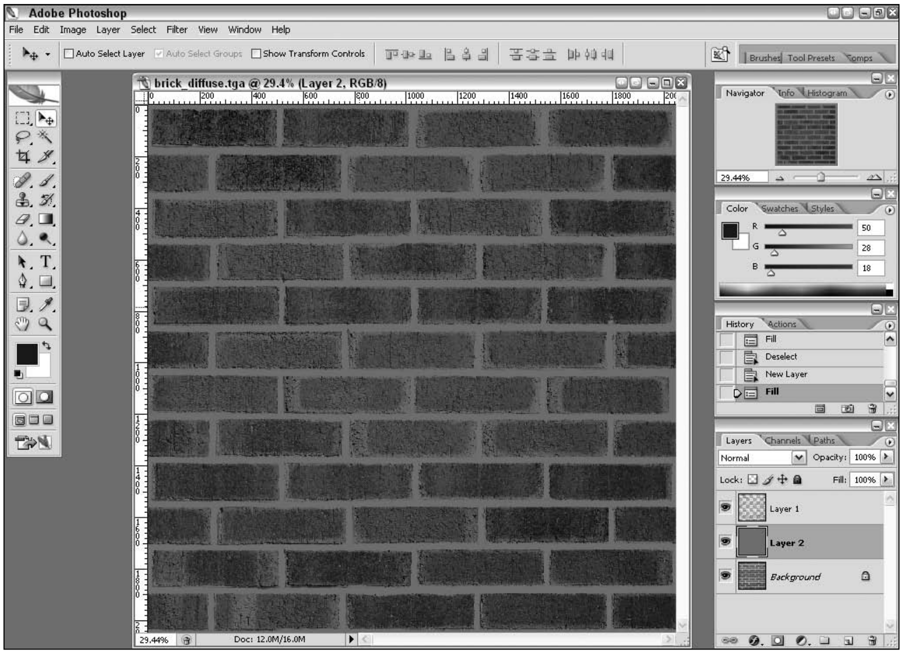


**Figure 13-19:** Fill the layer with a neutral color.

15. **Select Targa as the Format for your image and then enter** brick_spec.tga **for the name.**
16. **Click Save and select 24 bits/pixel when prompted.**

    You're ready to move on to the next piece of your texture.

## Bumping polygons in and out: The normal map

The next image up is the normal map. This image describes the surface of the map; it tells the game what parts stick out and what parts bump back in. It's very similar to the bump map except that this *normal map* also affects the way light reflects off the texture.

A normal map actually can affect the geometry of a game map. It's meant to move the polygons under which it is applied so that they bump out or in. This is why light is affected by this texture element — because light bounces off a curve differently than off a flat surface.

Before starting, use your History tab in Photoshop to revert to the image before it was altered for your specular map. Scroll up until you find the Color Range history layer and select the layer before it.

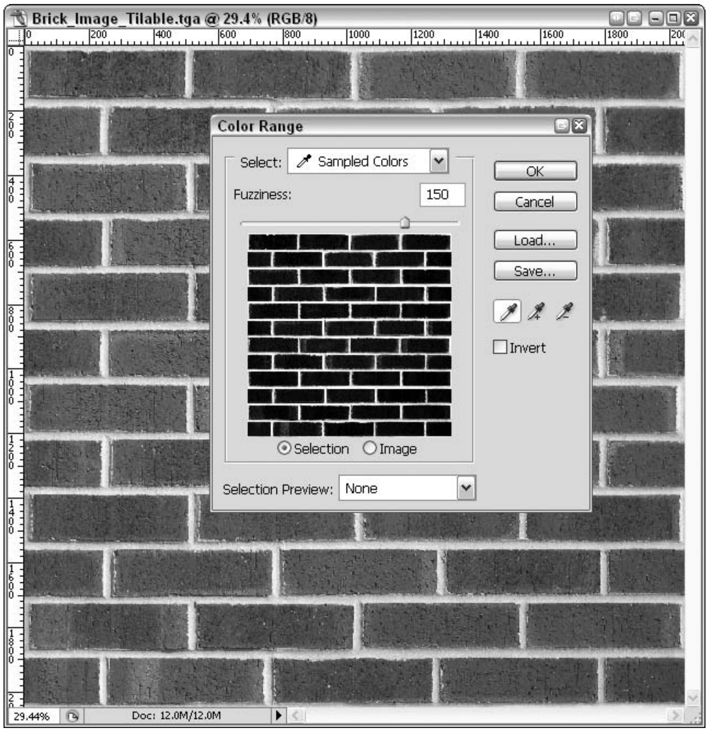
In the beginning of this chapter, I mention that you need to install a conversion tool (found on this book's CD) called `Photoshop_Plugins_7.83.0629.1500.exe`. Well, now is the time to use it, so make sure you've installed it by running the executable installation program provided. If you need confirmation that it is installed, in Photoshop, choose Filter. Near the bottom of the drop-down list you should see an item called NVIDIA Tools. If you see this filter, you're ready to go. If you're unable to install or use the Photoshop plug-in as provided, save the image you currently have open in Photoshop as `brick_normal_pre.tga`. Then skip to the end of this section, where I discuss an alternative to creating a normal map.

Now that you're ready to create your normal map, it's time to tell Photoshop what parts of the image will stick out, lie flat, or stick in. Like the specular map you created, you define these different aspects of the image by using white, black, and shades of gray. White defines the areas that stick out, such as the bricks, and black defines the areas that stick in, like the mortar. You do all this by using the color-selecting tool in Photoshop:

1. **Open the color selector by choosing Select➪Color Range.**
2. **In the image, click the mortar with your eyedropper.**

   Starting with the mortar is easiest because it's mostly a solid color.

3. **Turn the Fuzziness down to 100 by moving the slider near the top of the window. Then, when you have what looks like Figure 13-20, click OK to make your selection.**



Brick_Image_Tilable.tga @ 29.4% (RGB/8)

Color Range

Select: Sampled Colors

Fuzziness: 150

[✓ Selection] Image

Selection Preview: None

☐ Invert

29.44%

Doc: 12.0M/12.0M

**Figure 13-20:** Select the white color in the mortar between your bricks.

The selection you just made sticks in on the texture. Therefore, it is defined with the fill color of black.

4. **Create a new layer in the Layers tab in Photoshop.**
5. **Choose Edit⇨Fill, select Black as the Use color, and then click OK to apply the color fill to the new layer.**

   The rest of the image sticks out, and no in-between part of the image lies flat.
6. **Select the Background layer in the Layers tab and create another new layer.**
7. **Deselect everything by pressing Ctrl+D.**

8. **Then fill the new layer with white by choosing Edit⇨Fill and applying White as the Use color.**

   The result looks like Figure 13-21.

   Now for the NVIDIA filter that you installed earlier. In order to use this filter, you need to combine all your layers into one layer. In Photoshop, this is *flattening.*

9. **Choose Layer⇨Flatten Image.**

   All your layers are combined into a single layer. You can now apply the NVIDIA filter.

10. **Choose Filter⇨NVIDIA Tools⇨NormalMapFilter.**

    A dialog box opens. A lot of options are in this tool, and for this filter to work with the game, it's important that you set them correctly. As shown in Figure 13-22, options are in three columns, and you should set them as follows:

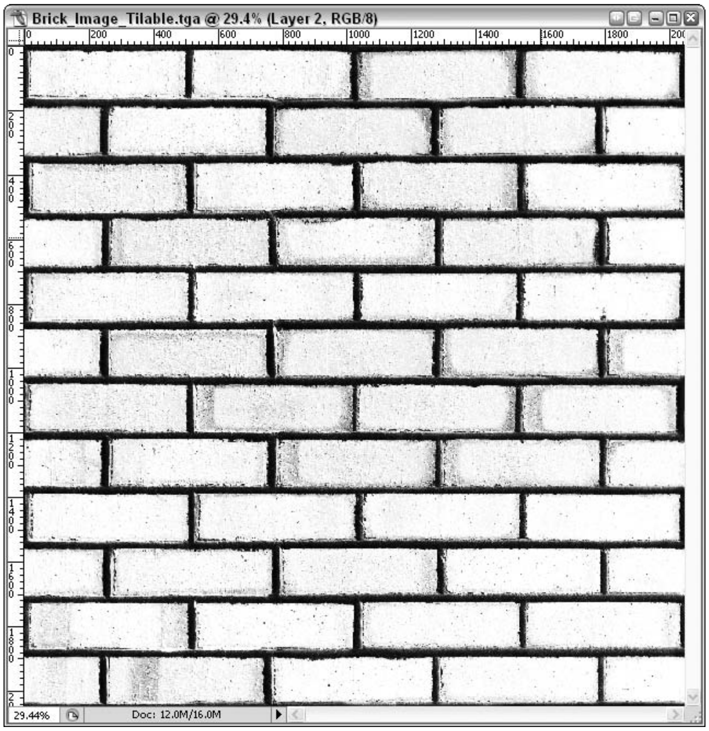
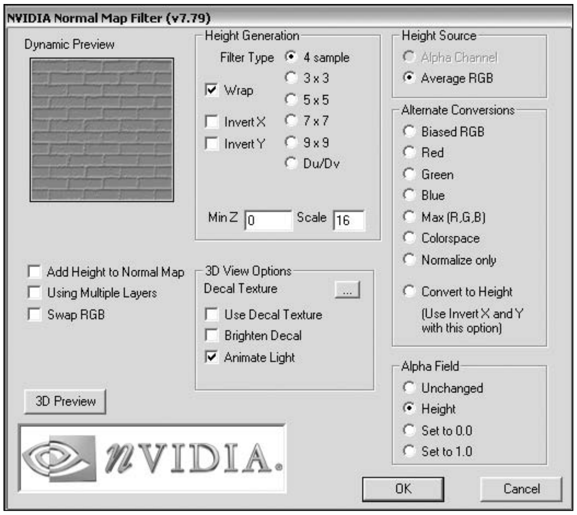


**Figure 13-21:** Apply color information: white represents what sticks out and black what sticks in.



NVIDIA Normal Map Filter (v7.79)

Dynamic Preview

Height Generation
Filter Type [✓ 4 sample] 3 x 3 5 x 5 7 x 7 9 x 9 Du/Dv
[✓ Wrap]
Invert X
Invert Y
Min Z 0 Scale 16

Height Source
Alpha Channel
[✓ Average RGB]

Alternate Conversions
Biased RGB
Red
Green
Blue
Max (R,G,B)
Colorspace
Normalize only
Convert to Height
(Use Invert X and Y with this option)

Add Height to Normal Map
Using Multiple Layers
Swap RGB

3D View Options
Decal Texture ...
Use Decal Texture
Brighten Decal
[✓ Animate Light]

Alpha Field
Unchanged
[✓ Height]
Set to 0.0
Set to 1.0

3D Preview

OK Cancel

**Figure 13-22:** The NVIDIA Normal Map Filter.

- The first column of options should be left all deselected.
    - In the second column, select a Filter Type of 4 Sample because this is a required option for the game. Also select the Wrap check box because this image is a tileable image that can be wrapped. Your MinZ should be set to 0, indicating that your lowest height value is 0. Your Scale should be set to 16. (I explain the Scale in more detail shortly.) Select the Animate Light check box because this helps display your changes in the 3D Preview.
    - In the third column, select Average RGB, which should be the only available option. Don't select any of the Alternative Conversions. Then select an Alpha Field of Height.

You might want to experiment with the Scale option. This setting adjusts how much bumping occurs with your image. The lower the number, the less of a bumping effect your image has. I suggested that you set this to 16, but you can try other values. Play with the number if you want and then click the 3D Preview button to see what it might look like in the game.

11. **When you're ready, click OK to apply the filter changes to your image.**

When you do, your image turns mostly blue, something like Figure 13-23. (Trust me, it's blue.) This is typical because the normal map uses color information to create the effect of bumpiness. This is your height map, and other than requiring a resize and save, it's ready for the game.

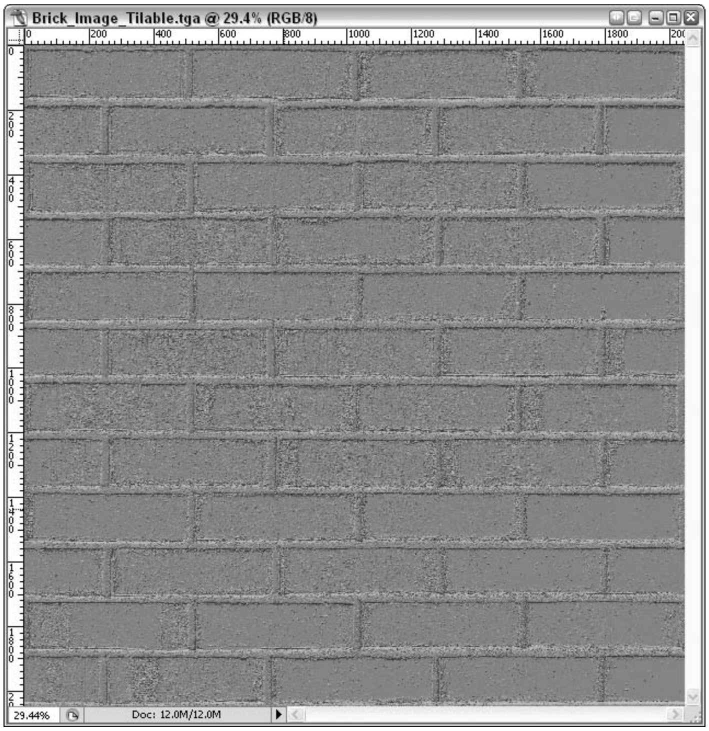


**Figure 13-23:** Apply the filter changes to your normal map. It will look blue.

12. **Resize the image to 256 pixels square.**

Your normal map image is created and ready to be saved. However, because you want to also use a specular map with your texture, this is the point at which it needs to be inserted into your normal map. Remember, specular maps are integrated with normal maps when normal maps are present.

To assign the specularity to the normal map, you must insert your specular map image into the alpha channel of the normal map image. This image that you're creating currently has four color channels: RGB, Red, Green, and Blue. These color channels can be seen individually in Adobe Photoshop by opening the Channel tab located behind the Layers tab.

One more channel, however, could be added to this image. That is the alpha channel, and it contains transparency information. Anything that is white is visible and anything that is black is transparent. *Half-Life 2* uses this information to determine which parts of the image are shiny and which parts are not. I think this is pretty smart of Valve Software.

To insert your specular map image as the alpha channel image, you need to create a new channel layer, copy the specular map image, and paste it into the alpha channel, as I outline here:

1. **If you haven't selected the Channels tab in the lower-right corner of the Adobe Photoshop editor, do so now.**

   The Channels tab looks like Figure 13-24.

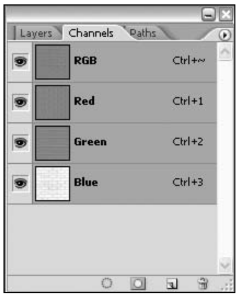


**Figure 13-24:** Select the Channels tab and create a new layer.

2. **Create a new channel by clicking the New Layer icon at the bottom of this window.**

   The default new channel is an alpha channel because all the color channels are already represented.

3. **Open the** `brick_spec.tga` **image saved earlier.**

   Choose File➪Open, navigate to the file, and click Open. The image should have been saved in the folder `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materialsrc\chapter13`, where *`username`* is your Steam user name and *`Valve`* is optional based on your installation.

4. **Copy all of** `brick_spec.tga` **by selecting the entire image and copying it.**

   Press Ctrl+A to select all the image. Then press Ctrl+C to copy it onto your clipboard.

5. **Paste the copy of your specular map image into the new alpha channel within the normal map image.**

   When you switch back to the specular map image, the new alpha channel should still be selected. If it isn't, select it now. Then press Ctrl+V to paste your copy into this layer, as I have done in Figure 13-25.

6. **Select the RGB channel to view your normal image without the alpha channel.**

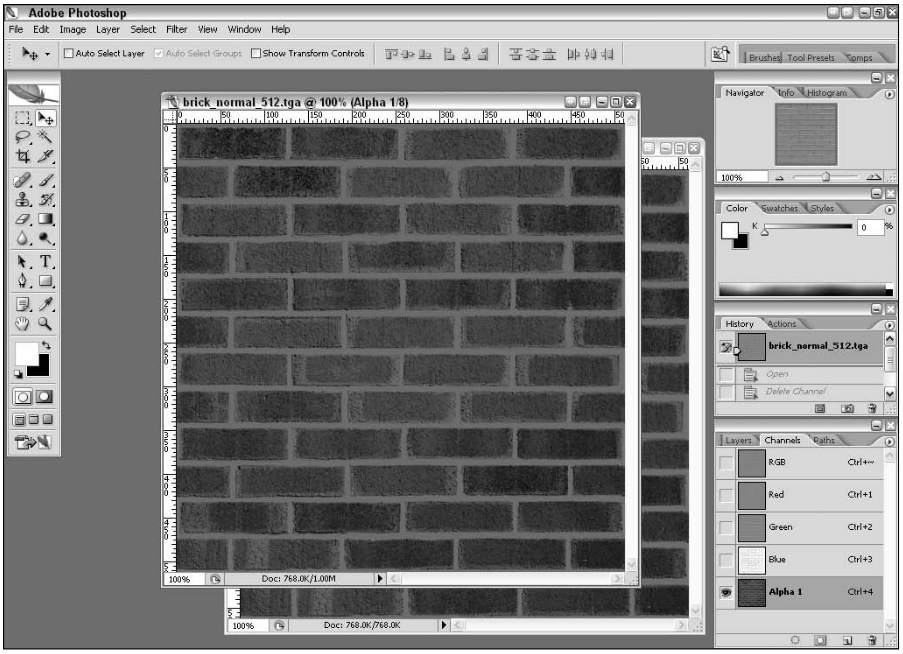


**Figure 13-25:** Paste the copy of your specular map image into the alpha channel.

Now your normal map image is ready and can be saved. Save this image in your custom materials folder as `brick_normal.tga`, making sure that `_normal` is at the end of the image name. However, this time, when prompted to save your Targa image resolution as 16, 24, or 32 bits/pixel, select 32 bits/pixel. This resolution contains the alpha channel image that you just created.

When saving your normal map image, it is very important to include the `_normal` at the end of the image name. Later, when you convert this Targa file into a Valve Texture Format file, this name will be required so the converter knows how to handle your image file.

TECHNICAL STUFF

The difference between 24-bit and 32-bit Targa images is that the 32-bit image contains the alpha channel and the 24-bit image does not.

## Creating a normal map without the filter

If you can't use the Photoshop plug-in detailed above, you have another option. ATI has released a tool called *Normal Map Generator,* which I have included on this book's CD as well. This image converter is not as precise and doesn't offer as many options as the Photoshop plug-in from NVIDIA, but it gets the job done.

To install this alternative tool, open the *NormalMapGenerator.zip* file and extract the contents to any folder on your hard drive.

Using Windows Explorer, locate the files you extracted from `NormalMapGenerator.zip` and run the file `TGAtoDOT3.exe`. After it's running, it will prompt you to select a TGA image file. Select the file you saved in Photoshop titled `brick_normal.tga` and click Open. Almost instantaneously, you will see a message that reads, "`Success! New TGA file: C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materialsrc\chapter13\brick_normal_preDOT3.tga`."

Rename that file for use within the game. Just right-click the file, select Rename, and change the name to `brick_normal.tga`.

## Building the VTF

Before converting your images into Valve textures, you can choose to set up some special parameters for your images that affect the image conversion. These special parameters can be turned on by setting a value of 1 next to a parameter's name or turned off with a value of 0 or by excluding it from your settings file. Some of these options are:

- **nolod:** This tells the game that no level of detail (LOD) is associated with this image. This is commonly used in textures that aren't found in any maps, but rather for locations in the game, such as the HUD (Heads Up Display).
- **nomip:** Mip-levels are not created with this setting turned on. *Mips* are different-sized versions of the same image to control quality seen and memory used. A 512 × 512–pixel image might have mip-levels of a 512 × 512–pixel version, a 128 × 128–pixel version, a 64 × 64–pixel version, and so on down the line.
- **clamps or clampt:** These settings prevent the texture from wrapping or tiling in either direction. This is often used with *sprites,* which are images that might be displayed by themselves without tiling.
- **skybox:** This setting makes sure that the edges of the skybox images don't show a seam. A *skybox* is a six-sided box with one image displayed on each of its six sides. You don't want to see where the corners meet in your box, and this setting helps with that.
- **startframe and endframe:** These two options are used for creating animated textures. Your textures must be named in sequence, such as texture000, texture001, and so on. The startframe is the first image and the endframe is the last image in the animation.

- **nocompress:** When the VTF is created, the image is compressed for the game to save file space and memory. You can turn off this compression if you want to ensure the highest quality image. However, your image will use up more memory and could slow down the game.
- **nonice:** *Nice* is a filter that is used for smaller mip-levels of a texture. This filter helps to make the smaller version look better for the most part. However, sometimes, you might prefer to turn this feature off in order to retain the look of the lower mip-level without filtering. Turn on nonice to turn off this feature.

These settings, if they are to be used, must be written in a plain text file. Then this file is saved as `*imagename*.txt`, where *`imagename`* is the name of the Targa image being converted. Save this file in the same location as your Targa file, and the Vtex converter finds it automatically. However, for the images that you're creating here, there is no need for the Vtex settings file because the default settings are fine.

It's time to pull all your images together into a single texture file for the game. As I mention earlier, *Half-Life 2* requires that textures be in this VTF format. The format contains not only the image data from the Targa files that you created, but also a few specific settings that you will have a chance to adjust momentarily.

To begin, open your file explorer program and navigate to the folder where all your texture files are saved. This should be `C:\Program Files\*Valve*\Steam\SteamApps\*username*\half-life 2 deathmatch\hl2mp\materialsrc\chapter13`, where *`username`* is your Steam user name, and *`Valve`* is optional based on your installation.

When you saved your first Targa image, you were instructed to create a folder unique to your map. This is the folder you just navigated to. I also recommended that you create another folder located at `C:\Program Files\*Valve*\Steam\SteamApps\*username*\half-life 2 deathmatch\hl2mp\materials\chapter13`. If you didn't create it before, create it now.

Now make some room on your screen so that you can see both the Targa images and the shortcut to `vtex.exe` that you created at the beginning of this chapter (see Figure 13-26). You need access to both this shortcut and your images so that you can easily select and drag your images over top of your shortcut. This is how you can convert them to the required VTF textures.

Select all your Targa images from the `chapter13` folder. Then drag your selection over the `vtex.exe` Desktop shortcut to convert the images. Don't open or launch `vtex.exe` from the shortcut, but simply drag your image files over top of the vtext.exe shortcut. You can convert each image individually if you wish, but it's often easier to just convert them all at once.

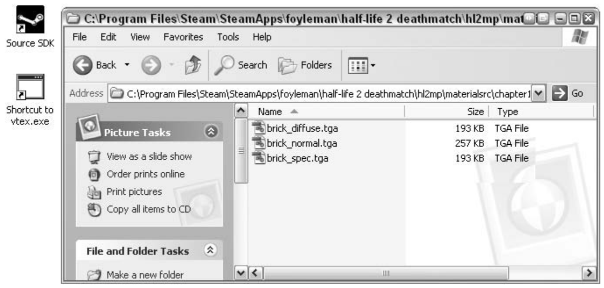
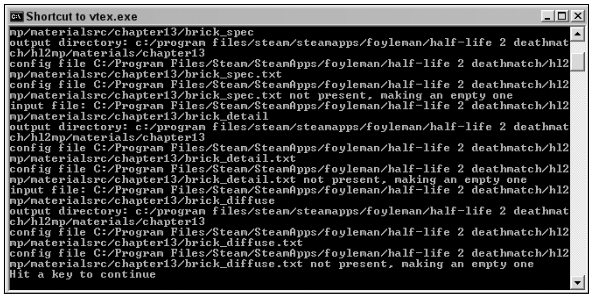

Source SDK
Shortcut to vtex.exe
C:\Program Files\Steam\SteamApps\foyleman\half-life 2 deathmatch\hl2mp\mat
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Program Files\Steam\SteamApps\foyleman\half-life 2 deathmatch\hl2mp\materialsrc\chapter1 Go
Picture Tasks
View as a slide show
Order prints online
Print pictures
Copy all items to CD
File and Folder Tasks
Make a new folder
Name Size Type
brick_diffuse.tga 193 KB TGA File
brick_normal.tga 257 KB TGA File
brick_spec.tga 193 KB TGA File

**Figure 13-26:** Arrange your workspace to see both your images and the Vtex shortcut.

TIP

In order for `vtex.exe` to work, you must have Steam running. If you've been following along from previous chapters of this book, you already have it running, as indicated by the Steam icon in your task tray.

After you drag the images over the `vtex.exe` shortcut and release the mouse button, a window pops up displaying the progress of the conversion, as shown in Figure 13-27. This window lets you know of any errors that come up and also lets you know when the conversion process is complete. As prompted within this window at the end of the conversion process, press any key to continue. This window will close.



```
mp/materialsrc/chapter13/brick_spec
output directory: c:/program files/steam/steamapps/foyleman/half-life 2 deathmat
ch/hl2mp/materials/chapter13
config file C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_spec.txt
config file C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_spec.txt not present, making an empty one
input file: C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_detail
output directory: c:/program files/steam/steamapps/foyleman/half-life 2 deathmat
ch/hl2mp/materials/chapter13
config file C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_detail.txt
config file C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_detail.txt not present, making an empty one
input file: C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_diffuse
output directory: c:/program files/steam/steamapps/foyleman/half-life 2 deathmat
ch/hl2mp/materials/chapter13
config file C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_diffuse.txt
config file C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2
mp/materialsrc/chapter13/brick_diffuse.txt not present, making an empty one
Hit a key to continue
```

**Figure 13-27:** This window displays the progress of the image conversion process.

You'll notice that you now have a text file for each of your Targa files in the `materialsrc\chapter13` folder. If the Vtex settings file is missing, Vtex will create a blank one during the conversion process. This is a blank file and can be deleted if you wish. It's up to you because it won't be needed at any time again.

If you switch to your `materials\chapter13` folder, what was once empty now contains your converted VTF files. This confirms that the conversion was a success, and you're ready to go to the final step of creating your custom texture.

## Making Everything Work Together

Now that you have all the pieces of the texture puzzle, it's time to assemble them. In a text file that looks just like a script, you need to tell the game where to find each of these images and how to assemble them. This bit of script in your text file is a *material.* It is saved as a Valve Material File (VMF).

### Building a material

To create a material for this new texture, follow these steps:

1. **Open Notepad.**

   You can find it at Start➪All Programs➪Accessories➪Notepad.

2. **In the text editor, type the following line of text:**

   ```
   "LightmappedGeneric" {
   ```

   This defines the texture you're creating as a commonly used, generic texture that is lit in the world by normal lighting methods. This definition is one of many *shader* definitions available for use in textures, but is also the most common.

3. **The second line tells the game what the defuse map texture is going to be. Enter it as such:**

   ```
   "$basetexture" "chapter13/brick_diffuse"
   ```

   This line defines the primary image to be displayed for this texture. First, you tell the game that this is the base texture that is to be displayed. Then you define it as being located in the folder `chapter13` and having the filename `brick_diffuse`. The file extension is left off because it's not needed.

   You could actually close off your material file here with a closing curly bracket, save it, and use it in the game. However, all the other detail images that you created would be ignored. Continue on with the additional definitions to enhance the final texture.

4. **The next line you need to type is**

   ```
   "$surfaceprop" "brick"
   ```

   Here you're telling the game that this texture acts like brick. It should look like brick when shot and sound like brick when walked upon. It defines the surface properties of this texture. You can choose from many other surface properties, including those in Table 13-1.

**Table 13-1 Textures and Surface Properties**

| *Texture* | *Surface Properties* | | | |
|---|---|---|---|---|
| **Concrete and Rock** | baserock | boulder | brick | concrete |
| | concrete_ block | gravel | rock | |
| **Liquid** | slime | water | wade | |
| **Metal** | canister | chain | chainlink | combine_ metal |
| | crowbar | floating_ metal_barrel | grenade | gunship |
| | metal | metal_barrel | metal_bouncy | Metal_Box |
| | metal_ seafloorcar | metalgrate | metalpanel | metalvent |
| | metalvehicle | paintcan | popcan | roller |
| | slipperymetal | solidmetal | strider | weapon |
| **Miscellaneous** | braking rubbertire | cardboard | carpet | ceiling_tile |
| | combine_ glass | computer | default | default_ silent |
| | floating standable | glass | glassbottle | item |
| | jeeptire | ladder | no_decal | paper |
| | papercup | plaster | plastic_barrel | plastic_ barrel_ buoyant |
| | Plastic_Box | plastic | player | player_ control_clip |
| | pottery | rubber | rubbertire | sliding rubbertire |
| | slidingrubber tire_front | slidingrubber tire_rear | | |
| **Organic** | alienflesh | antlion | armorflesh | bloodyflesh |
| | flesh | foliage | watermelon | zombieflesh |

| *Texture* | *Surface Properties* | | | |
|---|---|---|---|---|
| **Snow** | ice | snow | | |
| **Terrain** | antlionsand | dirt | grass | gravel |
| | mud | quicksand | sand | slippery slime |
| | tile | | | |
| **Wood** | wood | Wood_Box | Wood_Crate | Wood_ Furniture |
| | Wood_ lowdensity | Wood_Plank | Wood_Panel | Wood_Solid |

5. **To insert your normal map into the material file, enter the following line:**

   ```
   "$bumpmap" "chapter13/brick_normal"
   ```

   The material now knows which image is to be used when creating the bumping effect in the game.

6. **Because you included the specular map with your normal map, you need to activate the effects of your specular map. Add these three lines next:**

   ```
   "$envmap" "env_cubemap"
   "$envmaptint" "[.07 .07 .07] "
   "$normalmapalphaenvmapmask" 1
   ```

   The first line tells the game that this texture has the ability to reflect light. The next line tells the game how much it should reflect. The highest level of reflection is 1, and it should be reserved for polished metal and mirrors. The lowest level is 0 so .07 makes a good setting for brick or concrete. The three digits represent red, green, and blue, respectively. Usually, you will want the numbers to be the same so the color of your image doesn't change when light in the game bounces off of the image.

   The last line tells the game that the reflective areas of the image are defined in the alpha channel of your normal map. You copied the specular map into the alpha channel of the normal map image earlier in this chapter.

7. **(Optional) You can add some other options to your texture:**

   Although none of these additional options are appropriate for your brick texture, you might want to use one or more of the following in one of your future custom textures:

   ```
   "$model" 1
   ```

This line is required when creating textures that are applied to models, such as the crates that you added inside the buildings.

```
"$decal" 1
```

This line is required when creating textures that will be used as decals within the game. A *decal* is a texture placed over a brush in the game. This texture lays over whatever it is applied to.

```
"$alpha" 0.5
```

Containing a value between 0 and 1, this setting defines the translucence of the texture, 0 being completely transparent. This is useful for glass or fence textures.

```
"$color" [1 1 1]
```

With this setting, you can multiply this RGB color value with the texture. So, if you have a white light that you want to look red, adjust the color to [1 0 0]. The values represent percentages of the RGB color scale.

```
"$nocull" 1
```

This line allows the texture to be visible from both sides of a brush when applied to a flat brush.

```
"$selfillum" 1
```

This setting causes the texture to give off light based on the alpha texture in your diffuse map.

8. **Finally, to close these lines of scripting in your material file, add the closing curly bracket.**

   The result, when fully assembled in your text file, should read like this:

```
"LightmappedGeneric" {
        "$basetexture" "chapter13/brick_diffuse"
        "$surfaceprop" "brick"
        "$bumpmap" "chapter13/brick_normal"
        "$envmap" "env_cubemap"
        "$envmaptint" "[.07 .07 .07]"
        "$normalmapalphaenvmapmask" 1
}
```

   With your material file written, you need to save it someplace that the game can find and read it.

9. **Choose File➪Save As in Notepad. Navigate to the folder** `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\materials\chapter13` **and save your new material file as** `brick.vmt`**.**

   The name of your VMT file becomes the name of your texture, so name it to whatever you feel is appropriate. In this case, `brick` seems to make the most sense.

After you save your file as `brick.vmt`, open the folder by using Windows Explorer and double-check that the file has the correct name. Sometimes, when you save a file with Notepad and you add a file extension other than `.txt`, the program adds `.txt` to the end. The result as a file named `brick.vmt.txt`. Check that this hasn't happened and, if it has, correct the error by renaming the file.

The name of the file isn't crucial, but again, you don't want it to be confused with a file that might have been created by someone else. Therefore, naming it in relation to your map makes sense.

Now you're done and ready to show off your work.

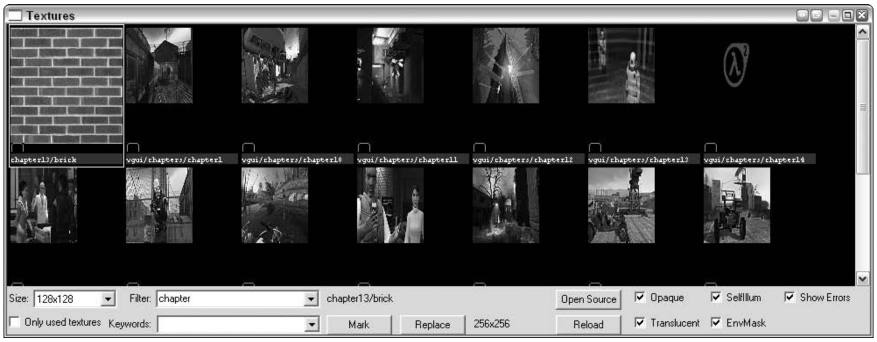
## Applying your material

Construction time is over, and now it's time to play. This is when you get to apply your texture in the editor and then see how it looks in the game.

1. **Load the editor, and then load your last map version, which should be** `dm_chapter12.vmf`**.**
2. **So as not to mess up the previous map, save this map with the new name** `dm_chapter13.vmf`**.**
3. **With your map loaded in the editor, move your camera in the camera viewport to the outside area and center on the outside of your building.**

   You apply your new brick texture to the outer walls of the building.
4. **To load your new texture, select the Texture Application Tool. Then click Browse to look through your texture.**

   Because your texture is saved in the `chapter13` materials folder, use "chapter13" to filter your textures. You should easily spot your new texture, as I have in Figure 13-28.



Size: 128x128 Filter: chapter chapter13/brick [✓ Opaque] [✓ SelfIllum] [✓ Show Errors]
☐ Only used textures Keywords: 256x256 [✓ Translucent] [✓ EnvMask]

**Figure 13-28:** Find the texture set that you created with your material file.

5. **Select this texture by double-clicking it.**
6. **Apply it to the outer walls of your building by right-clicking the walls in the camera viewport.**

   TIP

   Leave the top edge of the previous texture applied. You can see in Figure 13-29 that you can leave the top border of the original texturing in place around the building. This border looks good and does not interfere with the new texture.

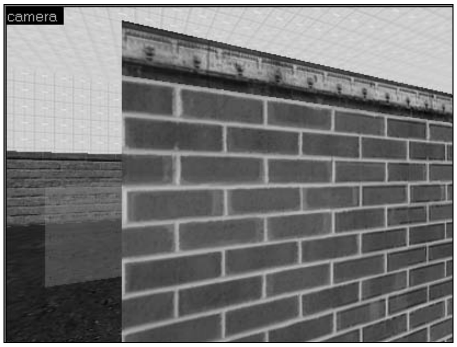


**Figure 13-29:** Apply the new texture to the walls but leave the top edge texture in place.

7. **Continue to apply this texture to the outer walls of your building until you have covered it entirely.**

   You might also want to apply this texture to the inner walls of your building for consistency.

   When you do this, you notice that you can't see where one square of the texture ends and the other begins. This means your attempt to make it tileable was successful, as shown in Figure 13-30.

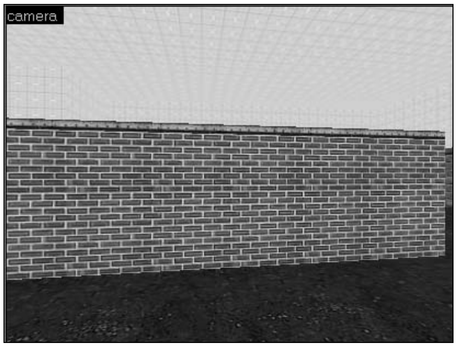


**Figure 13-30:** When applied to the wall surrounding your map, your texture looks clean and seamless.

8. **Now that you applied your new texture, save your map again. Then compile and play it from the Run Map window by pressing F9.**

   Press F9, check that all options are set to Normal, and then click OK. See Chapter 8 for details of this process.

   When the level loads, you see a beautiful-looking brick wall, like the one shown in Figure 13-31. Regardless of whether you're up close or far away, it will look good and run well in the game.

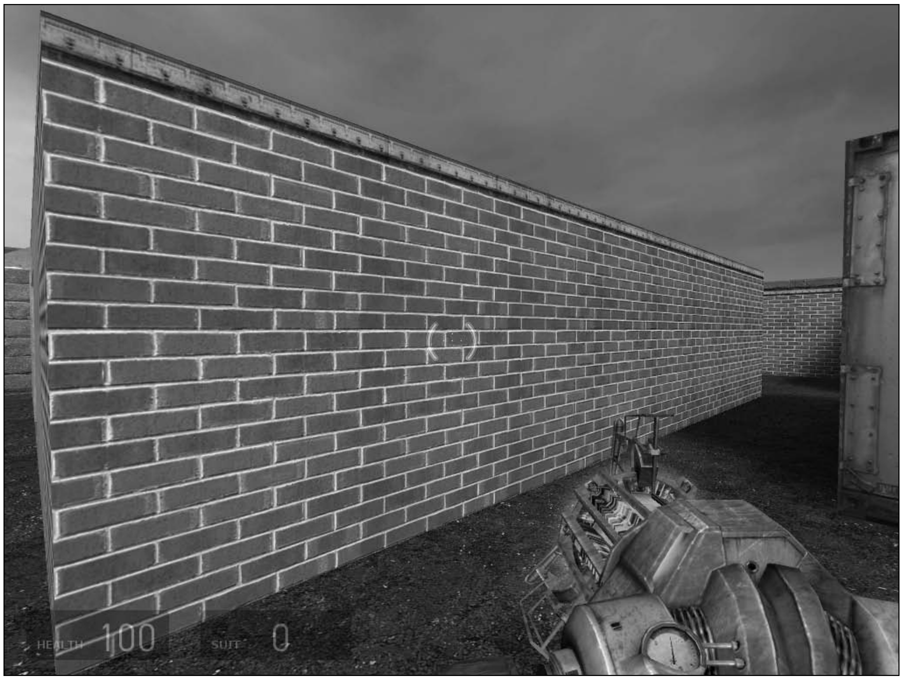


**Figure 13-31:** In the game, your texture looks well detailed regardless of your distance from it.

# Chapter 14
# Finishing Up Your Map

***In This Chapter***

▶ Getting more players into your game

▶ Adding cubemaps for reflections and specularity

▶ Dark and light shadows thanks to cubemaps

At this point, you have a nearly completed map. You have a couple rooms that lead to an outdoor area, and crates and weapons for defensive and offensive strategies are available for the players. However, you have only one player in the map so far. Playing a level intended for deathmatch doesn't make sense unless you allow other players to join. To do that, you have to give other players a place to spawn into your level.

Along with additional player spawn points, one other element is missing. Regardless of whether you are making a deathmatch game type or a single player game type, you should include an entity commonly referred to as the *cubemap.* The cubemap provides the effect of reflections on the surfaces that can utilize this feature. In your map, this surface is your custom texture, and it's utilized by the specular map that you incorporate into the normal map of your custom texture.

More details on the cubemap are to come. First, it's time to add more players to the map. Open up your map in the Hammer Editor so that you can start to make your changes.

## Multiple Spawns for Multiplayer Games

Up to this point, I've been showing you how to design a map for a death-match game. It's too small for players to team up, and it has been designed so that the players can basically run in a circle either away from their opponents or toward them. This design makes it easy to place the spawn points for the deathmatch game type.

A spawn point already exists in the South building, so start by moving this spawn point to a better location in the room. Press Shift+S to make sure the

Selection Tool is active. Then click-select your `info_player_deathmatch` entity by clicking it from inside the camera viewport. Then, click and drag it from within the top viewport and place it behind the crates in the bottom-right corner of the room, as shown in Figure 14-1.

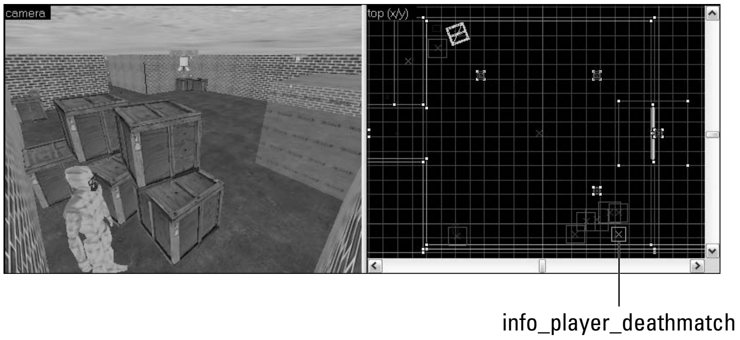


**Figure 14-1:** Move the existing player spawn point behind the crates in the same room.

What makes this a great location for the player to spawn is that it's not out in the open. Somewhat hidden by the crates, the player can spawn in without being immediately seen by another player, allowing the player to figure out where he is before jumping into battle.

The spawn entity is already facing the correct direction, which is out of the entrance of this stack of crates. So, no need to make any other adjustments here. You can move on to the next spawn point placement.

You could add the rest of the spawn points individually with the Entity Tool by doing the following:

1. Select the tool by pressing Shift+E.
2. Select the info_player_deathmatch entity as the object in the New Objects group on the right side of the editor.
3. Place the entity within the map.
4. Adjust the positioning of the entity so that it's not stuck in or hanging above the ground.
5. Adjust the angle at which the entity is facing.
6. Follow the same steps again with the next spawn point.

However, you can shorten this process a bit. Instead of creating a new point for each player, simply press Shift+click and drag a copy of the existing player spawn. Then all that's left is to adjust the angle at which the player is facing, and you can move onto the next spawn point. Do this for the next spawn point placement.

TIP

How many spawn points should you make? The maximum number of players allowed in a *Half-Life 2: Deathmatch* game is 32. In larger maps, I recommend that you place at least 32 spawn points in the event that 32 players try to spawn in at the same time. (Hey, it's possible.) However, you're allowed to place any number that you want, more or less, and the game will use what it has available.

Another good location for a player spawn point is behind the crates in the North building. Hold Shift, click, and drag your existing player spawn point into the other room behind the crates (see Figure 14-2). When it's in position, release the mouse button and press Alt+Enter to open the Object Properties window. In this window, adjust the angle so that the player is facing the open area of the crates. It's no fun to spawn into a game and find yourself disoriented because you are facing the wall.

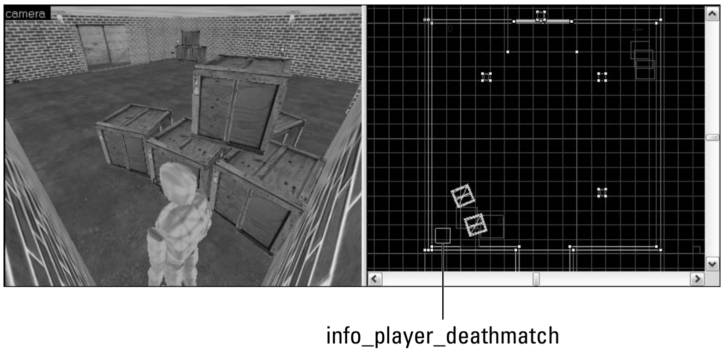


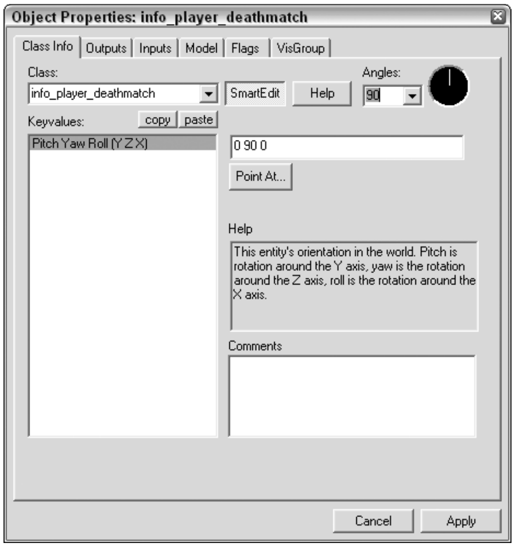
**Figure 14-2:** Move a copy of the spawn point behind the crates in the upper room.

In the top-right corner of the Object Properties window is a black circle with a line. This line points in the direction the player entity is facing. Click and drag this line to adjust your player's facing angle. The result, as shown in Figure 14-3, should read Angles: 90 and Pitch Yaw Roll: 0 90 0. This means the player is facing at a 90-degree angle to Y axis plane of your map.

Click Apply to apply the changes to your entity. Then close the window.

Now create another duplicate of this player spawn point just as you did above. Drag this to the nook between the two buildings in the outside area. This position isn't as safe as the others, but it's not in plain sight, which makes it an acceptable spawn location.

You will notice in Figure 14-4 that I've placed the player spawn point in the lower-right corner of this nook. Again, this location is generally out of sight from most of the map. The player should be able to spawn here without catching the eye of an enemy. As a bonus, when the player spawns into the level, a shotgun is available for immediate pickup. Just make sure that the player isn't facing the other wall in that corner.


Object Properties: info_player_deathmatch

Class Info | Outputs | Inputs | Model | Flags | VisGroup

Class: info_player_deathmatch

Angles: 90

Keyvalues:

Pitch Yaw Roll (Y Z X): 0 90 0

Help

This entity's orientation in the world. Pitch is rotation around the Y axis, yaw is the rotation around the Z axis, roll is the rotation around the X axis.

Comments

**Figure 14-3:** Adjust players' angles so they are facing away from any walls.

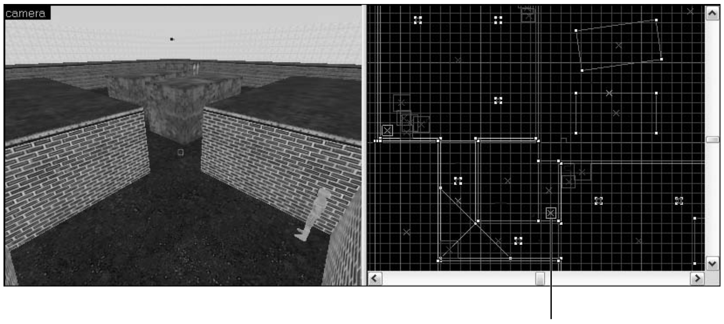


**Figure 14-4:** Place a spawn point in the nook between the two buildings.

One last spawn point should be placed behind the large crate in the upper-right corner of the outside area. It's another safe area for the player to spawn. Press and hold Shift while you click and drag one of your existing player spawn points into this corner.

When it's in the corner, adjust the facing angle so the player is facing away from the walls of the corner. A good angle in this corner would be 225 degrees, resulting in what you see in Figure 14-5.

That's enough spawn points for this map. It's a small map, and four players are plenty for a map this size. Save this map as `dm_chapter14.vfm`.

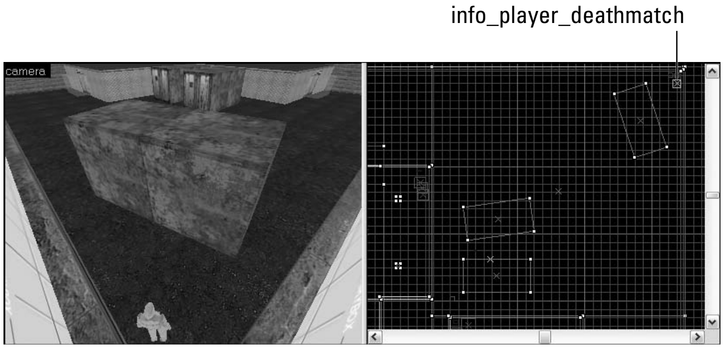


**Figure 14-5:** Place the last spawn point behind the crate in the upper-right corner of the map.

## Reflecting on Your Surroundings

Before you can consider this level complete, you need to add at least one cubemap. Your map may contain textures or other entities that have some capacity for reflecting either directly, like that of a mirror, or indirectly, like that of a specular map. In order for the game to be able to reflect its surroundings, it must calculate what those surroundings are. This is where the cubemap comes into play.

### Dropping in cubemaps

Placing a cubemap into your map is as simple as placing any other entity. That's because the cubemap is an entity; the `env_cubemap` entity. However, knowing where to place each cubemap is very important so that your reflections not only look good but also don't hinder the performance of game play.

For each cubemap that is placed in your map — yes, you can place many and you will soon — the game essentially takes a picture — actually *six* pictures — of its surroundings. These pictures are extremely similar to the images used in creating your sky in that there is one picture for each side of a cube (hence the *cube* in *cubemap*). These pictures, when the game is being played, are then projected back from this cubemap entity and reflected off any surface that allows for reflection. This gives the effect of real reflections in the game.

TECHNICAL STUFF

The game uses pre-rendered images to project from the cubemap because it helps to make the game play smoother. If the game had to calculate environmental reflections in real time, your computer wouldn't have enough processing power to play the game. It's that much work for your computer. So, the game uses projected images that closely resemble the environment instead. These projected images are 64 × 64 pixels each in width and height.

I explain the reasoning for each placement of a total of eight cubemaps as I go through each of the following steps:

1. **Select the Entity Tool by pressing Shift+E and select the** `env_cubemap` **entity as your object within the New Objects Group on the right side of the editor.**

   With the entity tool turned on and the cubemap entity selected, you're ready to start inserting your cubemap.

2. **Place the first cubemap right next to the player in the South room.**

   Click in the Top 2D viewport in the corner of the South room and near the player. Make sure you place the entity within the room and not above or below it. Then press Enter to place the new entity.

3. **Switch to the Selection Tool by pressing Shift+S. Then adjust your ports so you have a good view of the new cubemap and the player spawn point next to it.**

   You need a close view of the two entities together for the next step, so position your camera and zoom in as I have done in Figure 14-6.

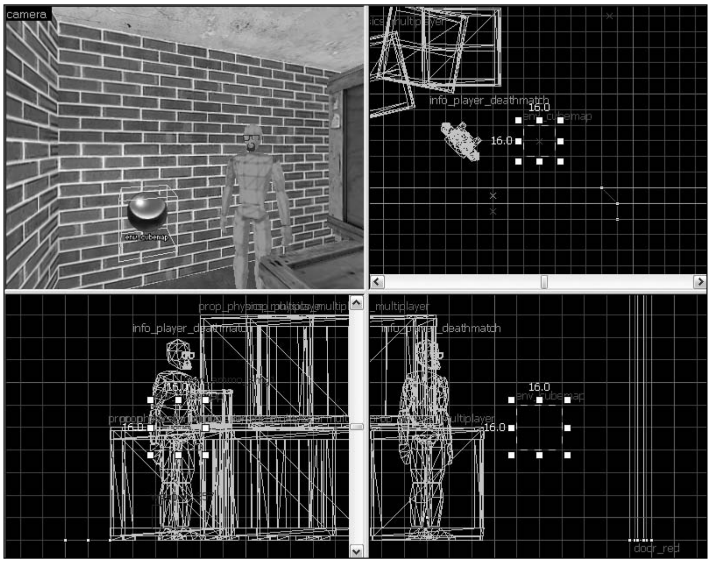


**Figure 14-6:** Place the cubemap next to a player spawn and zoom in on both entities.

4. **Adjust the height of the** `env_cubemap` **entity so that it is at eye level of the player spawn point (see Figure 14-7).**

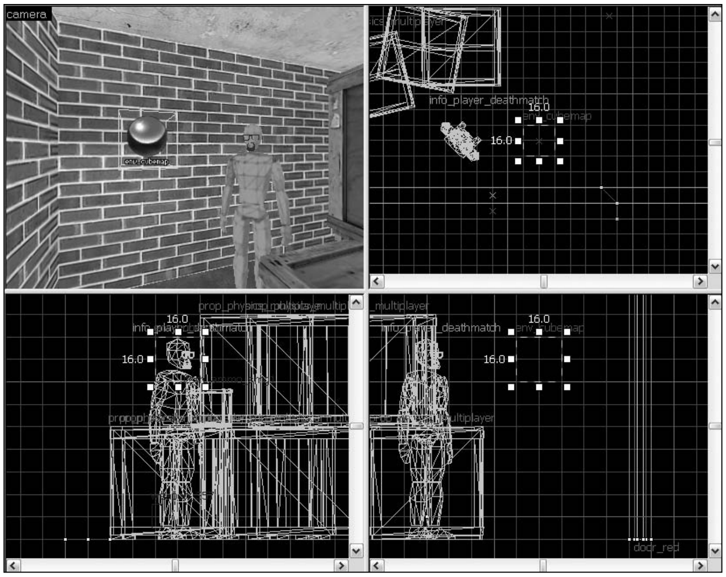


**Figure 14-7:** Adjust the height of the cubemap to put it in the player's line of sight.

You zoomed in close in Step 3 so you could see the player spawn's eye level within the map. When you zoom in, you can see what looks like a simple model of the character Gordon from the story of *Half-Life 2*.

Regarding the description above of the workings of cubemaps within the game, you can understand why you would want the images of these cubemaps to be projected as if from the player's line of sight. Placing the cubemap at the player's eye level accomplishes this objective. So, when the player runs through the level, the reflections appear from the perspective of the player's line of sight.

5. **Finally, move the cubemap entity to the center of the room, leaving its height unaffected (see Figure 14-8).**

   For now, this is the only cubemap to place in this room. The room is small, and adding more cubemaps to this room may not improve the effect. As a matter of fact, adding more cubemaps would increase the memory used by the game because the additional cubemap images must be loaded for display, which would make your computer work harder and possibly slow game play. Placing a single cubemap in the middle results in full coverage and even distribution of the reflection images. However, testing the level during game play will determine if more cubemaps will make the level look better without slowing game play.

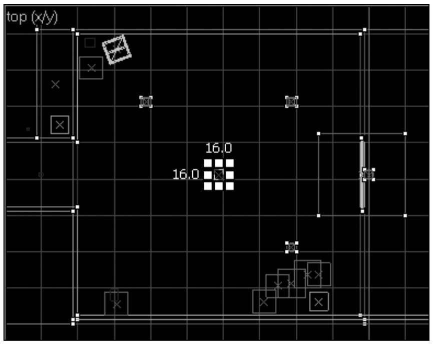


**Figure 14-8:** Move the cubemap to the center of the room for even coverage.

6. **Copy and place a cubemap in the remaining areas of the map.**

   See Figure 14-9 for placement of these cubemaps.

   Now that the first cubemap is placed, the rest are easy. With the first placement, you needed to get the height correct. Now you can simply copy the existing entity for placement in the rest of the map. By holding Shift, clicking, and dragging the first cubemap entity, you can create copies of it for placement elsewhere in the map.

   A cubemap should be placed in each sectioned area of playable map. By *sectioned*, I mean each square or rectangle of the map that essentially could be its own room. This includes

   - The two rooms (2)
   - Each leg of the hallway (2)
   - The nook between the rooms (1)
   - The large outdoor area (1)
   - The areas outside the doors of each room (2)

   That makes a total of eight cubemaps that need to be placed.

TIP

Visual errors may occur if a cubemap is too close to a wall or other world brush. To avoid this possible error, make sure that you don't place any cubemaps closer than 16 units from any world brush in the map.

TIP

In the hallways, place the cubemap right next to the lights that are also in the center of the hallway. Although there is no rule that states that these two entities should not overlap, I recommend that you don't overlap them, to prevent the possibility of visual errors in these areas of the map.

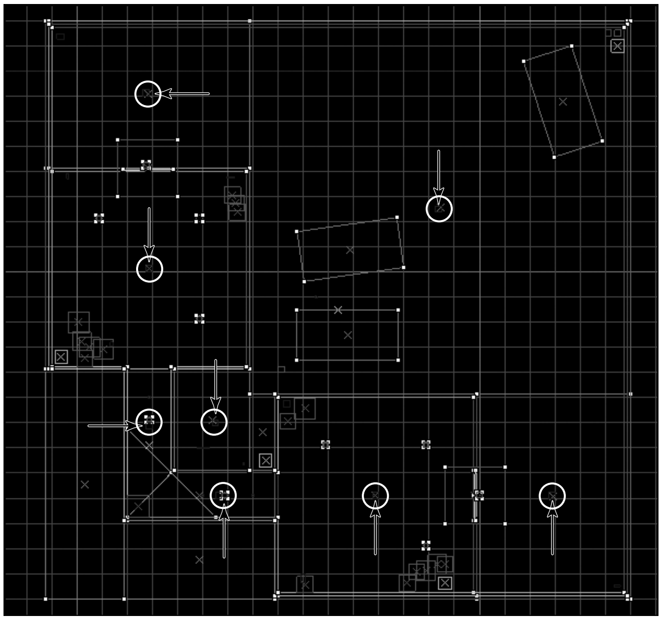


**Figure 14-9:** Place a cubemap in each section of map that could be its own room.

That takes care of the general cubemap placement. However, all this does is accurately project the reflective images from that specific point in the map. The reflection isn't going to be correctly represented in those areas of the map that are either brighter or darker than the area around the cubemap nearest that location.

For example, if you jump into the map now with your pistol in hand and walk into one of the dark corners of the building, the weapon will reflect more light than it should. You can see this in Figure 14-10. This is because the reflections on the weapon are based on the cubemap closest to the player, which happens to be in a moderately well-lit area of the room.

To fix this problem, you need to place another cubemap in the darker corners of your map. Each of the rooms has one corner that doesn't have a light, and the upper-right corner in the outside area also has a lot of shade. Place an `env_cubemap` in each of these three corners as outlined in Figure 14-11; remember to keep it at least 16 units away from the surrounding walls.

Now, when you generate your cubemaps and take another look at your weapon in one of these darker corners, the weapon won't shine so brightly.

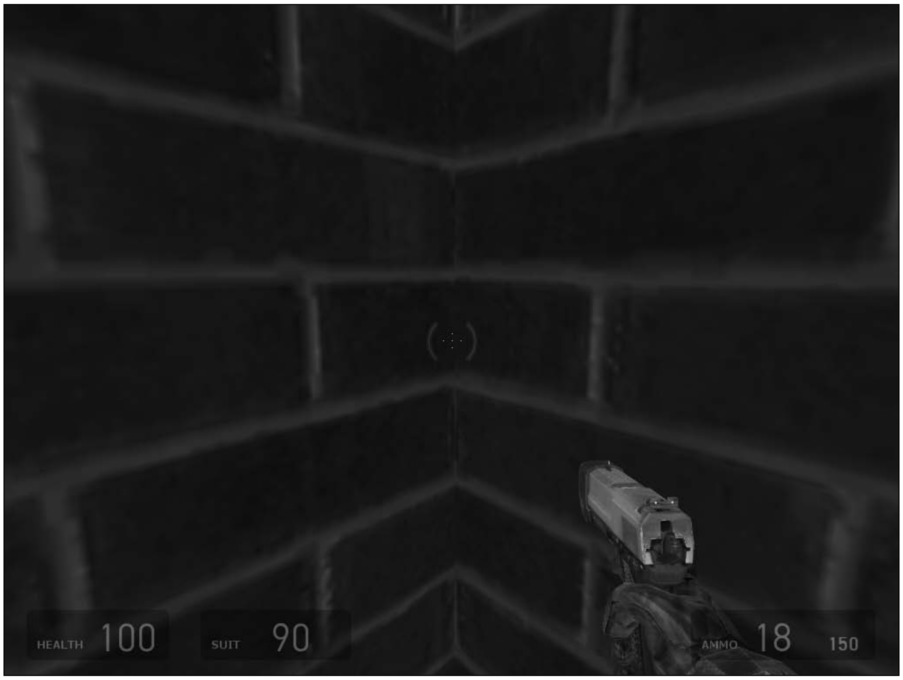


**Figure 14-10:** The weapon in this dark corner reflects an unrealistic amount of light.

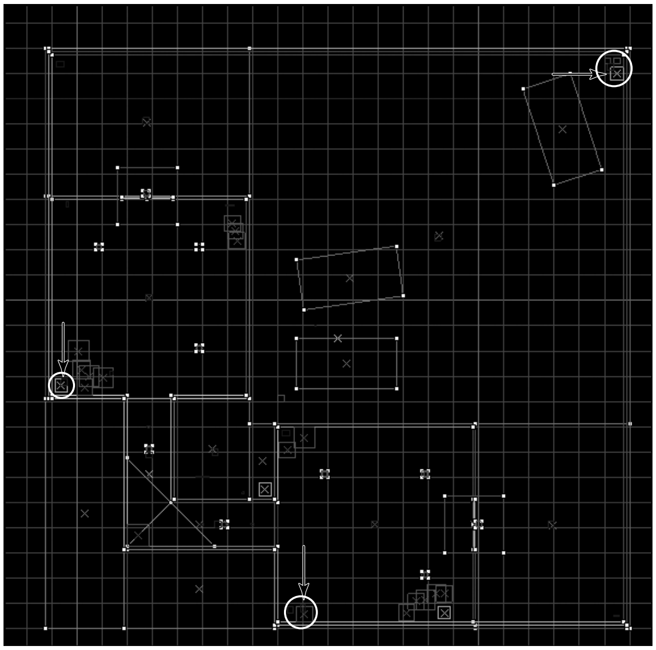


**Figure 14-11:** Place a cubemap in each dark corner of the map.

## Generating reflections

You have your cubemaps placed in the map, but all this does is tell the game where the reflective images will project from. You still need to create these reflective images. Thankfully, this is another simple process that begins with compiling your map.

Make sure you've saved the map by pressing Ctrl+S. Press F9 to open the Run Map window, make sure all the BSP options are set to Normal, and then press OK to compile and run your level in the game.

When the map loads in the game, it may not look exactly as it did. It may be darker or brighter than it was in previous tests. Don't let this worry you; it will soon be fixed and look even better than before.

TECHNICAL STUFF

The reason the level may look different is that the cubemap is using the sky textures as images. When no other cubemap images are available, the sky textures are used by default, and this may cause things to look slightly different.

Open the console by pressing the ~ key and then type the command **buildcubemaps**. Then press Enter and wait. The screen shrinks and flashes a bunch of images. This is the game taking the six images for each cubemap that you placed in the map. Then, the game saves all of these images right inside the compiled map file. This makes it less of a hassle to deal with when distributing your file later. When you see the console again, the process is complete.

To see the changes to your level, you must restart it in the game. To do this, open the console (if it was closed) and enter **map dm_chapter14** to reload the map. The map loads, and you can look around at the final changes. Figure 14-12 shows you how the light now affects your weapon more realistically.

REMEMBER

Now, if you decide to make any changes at all to your map file and recompile the map, you also have to rebuild the cubemap images. Just follow the same method above to generate these images, and you'll be set.

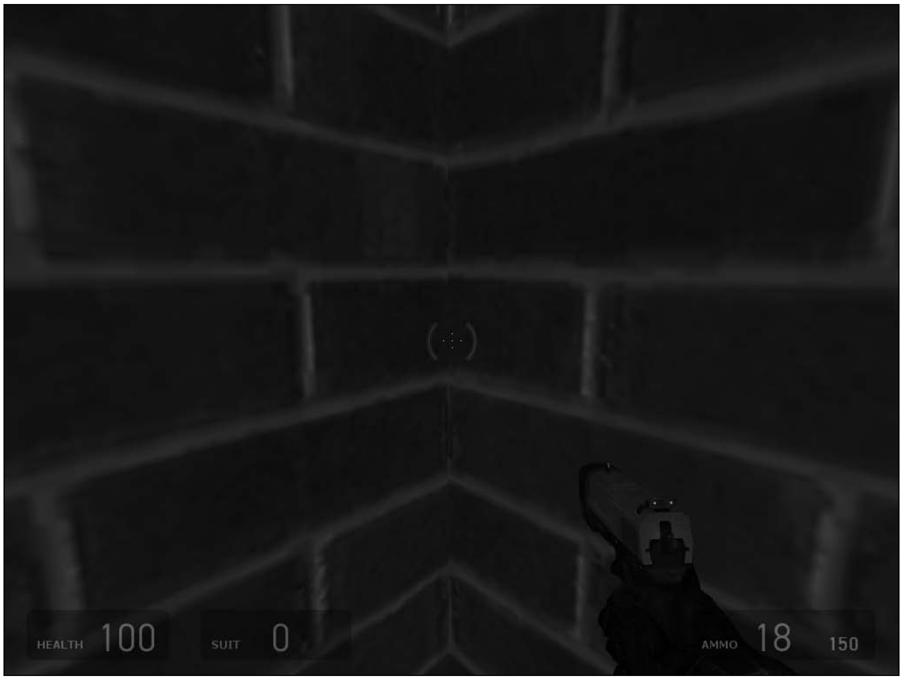


**Figure 14-12:** The weapon on this dark corner is now dark itself.

# Chapter 15
# Showing the World

*In This Chapter*

- Defining what's in the package and what should be done with it
- Putting together your BSP package
- Zipping up all the final files for release
- Getting your mod out to the public

After you finish creating all of your custom content, you're going to want to show it off. You could just put all the files together and start handing it out, but this can lead to confusion and other problems. Practiced modders commonly use a preferred method of distribution that I outline for you here. I also show you how to find places to put your package online and get it out to the community where it can be enjoyed by all.

## Packing Up for Release

One of the things that Valve Software has done to make distribution easier is to create a method of packing up all your files into a single file. This way, when someone downloads your mod, he has to place only one file — instead of 20 — in the proper directory, and then he's ready to go. I take you though the process of putting together your custom map — after it has been compiled for the last time — and your custom texture.

### Explaining yourself

After you have all your content built and tested to make sure it's complete, you need to write your `readme.txt` file. This is a text file, also called just README, that accompanies all modifications. It tells the user who made the mod, what it contains, how to install it, what (if any) bugs exist, as well as any other information you want to include.

## The origins of README

Many years ago, before game modifications, program writers would include a README file with their distributions. The file had the same purpose as it does for mods; however, the files were named just README rather than `readme.txt`. Because this file was generally viewed only by other programmers, a file extension wasn't required and so was often left off. It was named in all capital letters to draw more attention to the file.

TIP

Although the README file doesn't need to be included to make your files work with the game, it keeps you on friendly terms with the community. When you give your mod to the different distribution Web sites, you don't want to turn the site administrators into a support team for your mod. That's a sure way to have your file deleted from the Web site. This README offers the users basic answers to questions they might have as well as a way of asking questions directly of you, the author.

A good README outline supplies the users with as much information about your map as they need to know and, if necessary, with information about how to contact you with questions. An outline of a README looks like this:

```
============================================================

Map Title          :
Map Version        :
Author             :
E-mail             :
Website            :

============================================================

Game               :
Supported Gametype :
Map Size           :
Map Rating         :

============================================================

Contents of this Package :

============================================================

Installation Instructions :

============================================================

Construction Date  :
Construction Time  :
```

```
Computer Configuration :
Custom Content     :
Known Bugs         :

============================================================

Credit to Authors  :
Special Thanks     :
Additional Notes   :

============================================================
```

All this information is contained within a plain text file. This way, it can easily be read in Notepad or any other text editor out there because they all can read plain text.

The file should begin with map and author information. Give yourself credit for a job well done. Let people know who you are and possibly where they can find more custom content that you created.

Then tell the user and distributor what this content is for. As a user, I sometimes find files that I downloaded months or years ago, but I can't remember which game they go with. The same has happened to me as a Web site administrator. If the users or site administrators don't know within which game to install your content, they're just going to delete it.

Next, tell the users what's in the package. Let them know that it's a map, custom texture, or whatever you might have put together. Then, when they load this content in the game, they know what to look for.

A very important part of this README file is the installation instructions. You might know what you're doing when it comes to installing game files. However, thousands of other people out there have no clue what to do. Try to include detailed instructions about how to make your files available in the game.

Providing construction information might not seem important, but if you continue modding games, it will be something that you will want to know in the future. As I wrote this chapter, I received an e-mail from someone looking for information on compile times for his map. I referred him to a map I made about five years ago. I wouldn't have remembered the details of this project on my own, but because the details were in the README, I was able to help him with his questions.

Finally, just as important as telling the user what's contained in your package, you must give credit to those who deserve it. If you found inspiration in someone else's work or received help from someone or some group, give credit for it. This is how the gaming community remains a community; it avoids accusations of theft and provides respect to those in the community who help and support it the most.

To create a README file for your custom level, start by opening your plain text editor. I recommend using Notepad because it comes with every installation of Microsoft Windows and is therefore readily available. Also, since Notepad is a plain text editor, using it ensures that anyone can read the file because there has been no special formatting applied to the text; it's just plain text.

You can find Notepad by choosing Start➪All Programs➪Accessories➪Notepad. This opens a new Notepad window, as shown in Figure 15-1.

**Figure 15-1:** Create your README using Notepad or another plain text editor.

Type the contents of your README as outlined above. Fill in the blanks with information about your custom level and its contents. When you are done, it should read something like this:

```
============================================================

Map Title          : Chapter 14
Map Version        : v2.0
Author             : foyleman
E-mail             : foyleman@modsonline.com
Website            : http://www.modsonline.com

============================================================

Game               : Half-Life 2
Supported Gametype : deathmatch
Map Size           : 4 players
Map Rating         : for anyone, no adult content
```

```
============================================================

Contents of this Package : This is a custom map made for the Half-Life 2 Mods
               For Dummies book. It contain a custom brick texture.

============================================================

Installation Instructions : Save dm_chapter14.bsp to the folder C:\Program
               Files\Valve\SteamApps\username\half-life 2 deathmatch\hl2mp\maps
               where Valve is you installation directory and username is your
               Steam account username. Then load the map dm_chapter14 in a
               deathmatch game.

============================================================

Construction Date  : 08/2006
Construction Time  : 2 days
Computer Configuration : AMD 4200 X2, 2Gb Ram
Custom Content     : textures, map
Known Bugs         : NA

============================================================

Credit to Authors  : NA
Special Thanks     : Half-Life 2 Mods For Dummies
Additional Notes   : Check out MODSonline.com for more maps and tutorials for
               Half-Life 2 and more

============================================================
```

Save this file as `dm_chapter14.txt`, which is the same as your map file's name but with a different suffix. Save the file in the same folder on your computer as the BSP file — that is, `C:\Program Files\`*`Valve`*`\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\maps`, where *`Valve`* is your installation folder and *`username`* is your Steam account user name.

Some people like to save their file as `README.txt`. However, that means a lot of different maps' text files have the same name. By naming this file the same as your map file or perhaps as `yourmapname_README.txt`, you are better assured that it won't get lost or overlooked.

That's it! Your README is all set and ready for inclusion with your distribution package.

## Picturing your level

When you submit your final package, the distribution Web sites will ask you for a screenshot of your content. A *screenshot* is a picture of what you see on your screen. Often, this means a screenshot from within your level. However, this image can be anything that you want. The game allows you to take screenshots automatically and saves them in JPG image format in a folder on your computer. By default, the game sets F5 as your screenshot shortcut key. This can be changed from within the game's keyboard options.

REMEMBER

If F5 is no longer your screenshot command, check your game options to see what the new keyboard command is. You can do this by pressing Esc to get to the game menu, choosing Options, clicking the Keyboard tab, and scrolling down to the option labeled *Take Screen Shot*.

The distribution Web sites ask you for this screenshot so that they can include a visual image of your content on the Web page. By supplying the Web site administrators with that image, you save them time and effort. Less work for Web site administrators means a greater likelihood that your package will be posted without delay.

To take this shot, load your level in the game. When it's loaded, move about your level by using your standard keyboard controls, looking for the picture you want to use to represent your level. When you're ready, press F5 (or whatever your assigned keyboard shortcut is) to take a screenshot, as I have done in Figure 15-2.

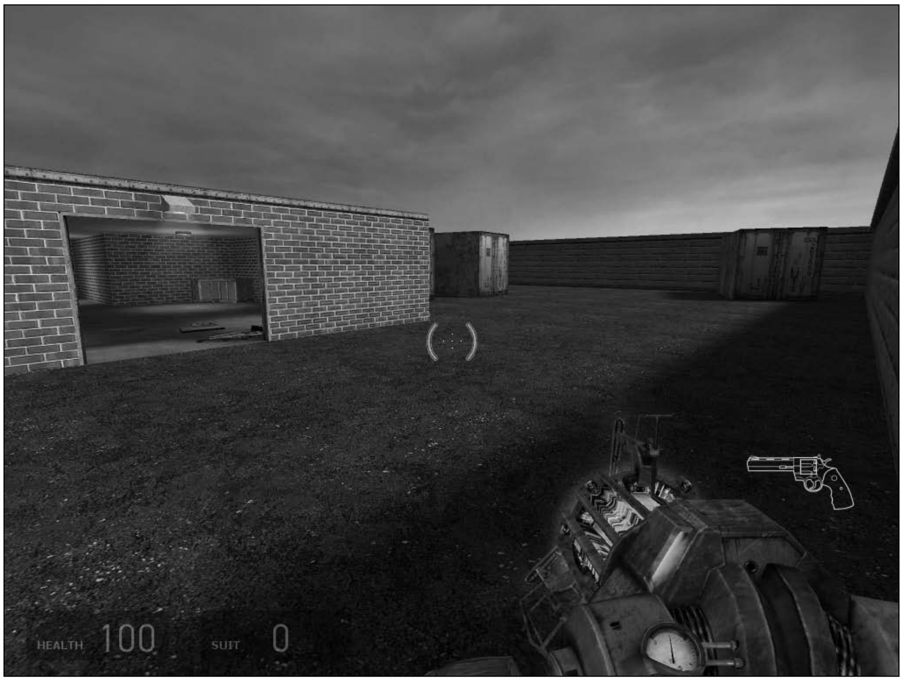


**Figure 15-2:** Take a screenshot of your level from within the game.

Then you can exit from the game and retrieve your screenshot. Here's how to find it:

1. **Open Windows Explorer and navigate to the folder** `C:\Program Files\`*`Valve`*`\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\screenshots`**.**

   This folder may not have been here before, but when you took a screenshot from within the game, the folder was automatically created and the screenshot was saved here. In the path above, *`Valve`* is your installation directory and *`username`* is your Steam account user name.

2. **Select the most recent screenshot in this folder and copy it by pressing Ctrl+C.**

   When you took your screenshot, the game created a JPG file. The file was named the same thing as your custom map but followed by a series of four numbers, starting with 0000. Each time you take a screenshot, the game creates a new JPG file and increments this four-digit number by 1. This means that you can take up to 10,000 screen shots of a single level before you start overwriting existing screenshots.

   In my case, I found the screenshot named `dm_chapter140000.jpg` listed here, and it was the most recent file added to this folder.

3. **Now change folders to** `C:\Program Files\`*`Valve`*`\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\maps` **and paste your copied file here by pressing Ctrl+V.**

   You want to paste your screenshot in the same folder as your BSP file so that later it can be included in a package for distribution with your BSP.

TIP

You can now rename your screenshot. However, sometimes you want to include more than one screenshot with your distribution. The screenshot naming scheme provided by the game is fine, and you don't need to rename them.

Now move on to the next section to create your BSP package, and later you will include your custom screenshot for distribution.

## Packaging additions to the game

The less confusing it is for people to install your level, the more likely it is that they will install it and enjoy it. For that reason, you want to package your custom maps and other content into a single file that you can easily hand out to the world.

With that in mind, the package you put together will contain:

- The map and related items for that map as constructed in earlier chapters.
- The custom texture, consisting of a few images and a material file.
- The README that you create earlier in this chapter.

The last chapter that required working with your custom map and related elements was Chapter 14. The files from this map, including the README created earlier in this chapter, are what you put together in your package. Those files are

- `hl2mp\maps\dm_chapter14.bsp`: This is your compiled map.
- `hl2mp\maps\dm_chapter14.txt`: This is the README file that you created to describe the contents of your level.
- `hl2mp\maps\graphs\dm_chapter14.ain`: This is a file created by the game that helps with the game's artificial intelligence (AI). This file was automatically created for you when you first played your level in the game. If this file doesn't exist, the game will automatically create it for you, so it isn't critical.
- `hl2mp\maps\soundcache\dm_chapter14.cache`: This is the list of sounds that are pre-loaded for the game to use in your level. This file was automatically created for you when you first played your level in the game. If this file doesn't exist, the game will automatically create it for you, so again, it isn't critical.
- `hl2mp\materials\chapter13\brick.vmt`: Your custom brick material file, as created in Chapter 13.
- `hl2mp\materials\chapter13\brick_detail.vtf`: One of your custom texture images, as created in Chapter 13.
- `hl2mp\materials\chapter13\brick_diffuse.vtf`: One of your custom texture images, as created in Chapter 13.
- `hl2mp\materials\chapter13\brick_normal.vtf`: One of your custom texture images, as created in Chapter 13.

A package is a single, compressed file that contains your compiled level and all of the other files required for your level to work in the game without any errors. You compress these files into a single BSP file by using a method similar to other compression utilities, such as WinZip, Winrar, and so on. The only thing you absolutely need to distribute is this single BSP that contains everything needed to run your level in the game.

To create this package, you need a program that can compress and insert these files into the BSP file. The Source SDK, which you should already have downloaded and installed in Chapter 4, includes Bspzip for this purpose.

Alternatively, I review a third-party program that you may prefer to use over Bspzip. Pakrat is a stand-alone utility with a GUI, making it easier to see what you are doing rather than having to work from the command line. Information on how to use Pakrat can be found on the CD media in the back of the book in a PDF entitled Chapter 18.

### Bspzip

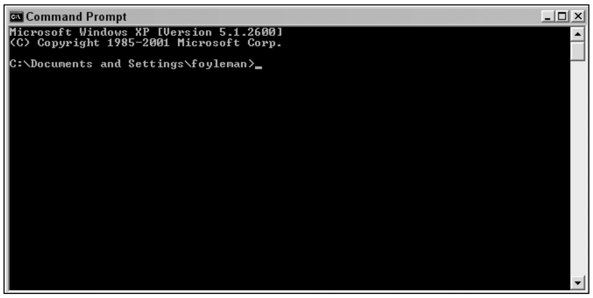
Bspzip is the program that came with the Source SDK. When you installed the modding tools for *Half-Life 2*, you automatically installed this program as well.

As I note earlier, some people find this program difficult to work with. This is because in order to use the program, you have to manually type all the commands into a Command Prompt window. Depending on how many custom files you need to include with your package, this can be time consuming.

The level and custom content created in the previous chapters doesn't add up to a lot of different files, so using Bspzip in this instance is not all that troublesome. I break down the compression process for you here:

1. **Open a Command Prompt window by choosing Start➪All Programs➪Accessories➪Command Prompt.**

   This window, pictured in Figure 15-3, is where you type your commands to import each file into the BSP.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\foyleman>_
```

**Figure 15-3:** You type Bspzip commands in the Command Prompt window.

2. **In the Command Prompt window, type the following command to change directories to your maps folder where the BSP and Bspzip files are located and then press Enter:**

   ```
   cd \Program Files\Valve\SteamApps\username\half-life 2
           deathmatch\hl2mp\maps
   ```

   In this command, replace `Valve` with your installation folder and `username` with your Steam account username.

   This line begins with the command `cd`, which means *change directory*. The directory that you want to change to is listed after the command.

3. **Referring to the list of custom files for your map, enter the command to import a file into the BSP, pressing Enter after each command:**

```
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp
        "materials/chapter13/brick.vmt"
        "%vproject%/materials/chapter13/brick.vmt"
        dm_chapter14.bsp
```

   The command above is broken down by spaces.

   With the first statement,

```
"%sourcesdk%\bin\bspzip"
```

   you launch the Bspzip executable, which is located in the `sourcesdk\bin` folder. The inclusion of `%sourcesdk%` tells your computer that it should look in the Steam-defined `sourcesdk` folder.

   In order for this command to work, you must have the Steam application running in the background. If Steam isn't running, an error message will appear in the Command Prompt window telling you that Steam must be running.

   The next command,

```
-addfile
```

   tells Bspzip what function it is going to run. In this case, the function is to add a new file.

   The next three entries,

```
dm_chapter14.bsp "materials/chapter13/brick.vmt"
        "%vproject%/materials/chapter13/brick.vmt"
        dm_chapter14.bsp
```

   are details about the files to be added. They are, in order:

   - The path name of the file you are adding; relative to the game installation.
   - The full path name of the file. The inclusion of `%vproject%` tells your computer that it is to automatically insert the full folder path up to the location of your BSP file.
   - The resulting BSP filename after adding this file to the original BSP. Usually, this is going to be the same name as the original BSP file.

4. **Repeat Step 3 for the remaining files that need to be added to your BSP.**

   Those remaining files should be added with the following commands, pressing Enter after each line:

```
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp
        "materials/chapter13/brick_detail.vtf"
        "%vproject%/materials/chapter13/brick_detail.vtf"
        dm_chapter14.bsp
```

```
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp
           "materials/chapter13/brick_diffuse.vtf"
           "%vproject%/materials/chapter13/brick_diffuse.vtf"
           dm_chapter14.bsp
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp
           "materials/chapter13/brick_normal.vtf"
           "%vproject%/materials/chapter13/brick_normal.vtf"
           dm_chapter14.bsp
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp "maps/dm_chapter14.txt"
           "%vproject%/maps/dm_chapter14.txt" dm_chapter14.bsp
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp
           "maps/graphs/dm_chapter14.ain"
           "%vproject%/maps/graphs/dm_chapter14.ain" dm_chapter14.bsp
"%sourcesdk%\bin\bspzip" -addfile dm_chapter14.bsp
           "maps/soundchache/dm_chapter14.cache"
           "%vproject%/maps/soundchache/dm_chapter14.cache"
           dm_chapter14.bsp
```

That's all of the custom files that were made for this map. So, you're now done importing your files into the BSP.

If you want to confirm that your files imported correctly, you can run another Bspzip command to view the list of files contained within your BSP file:

```
"%sourcesdk%\bin\bspzip" -dir dm_chapter14.bsp
```

In this command, you are telling Bspzip to list the directory of files that have been compressed into the file `dm_chapter14.bsp`. When you enter this command and press Enter, you should get a list of files that reads something like this:

```
maps/dm_chapter14.txt
maps/graphs/dm_chapter14.ain
maps/soundcache/dm_chapter14.cache
materials/maps/dm_chapter14/chapter13/brick_248_-56_72.vmt
materials/maps/dm_chapter14/chapter13/brick_408_672_72.vmt
materials/maps/dm_chapter14/chapter13/brick_32_-272_72.vmt
materials/maps/dm_chapter14/chapter13/brick_-328_520_72.vmt
materials/maps/dm_chapter14/chapter13/brick_-544_304_72.vmt
materials/maps/dm_chapter14/chapter13/brick_-152_-56_72.vmt
materials/maps/dm_chapter14/chapter13/brick_-160_128_72.vmt
materials/maps/dm_chapter14/chapter13/brick_-328_120_72.vmt
materials/maps/dm_chapter14/chapter13/brick_696_-56_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_248_-56_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_696_-56_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_-336_968_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_-328_520_72.vmt
materials/maps/dm_chapter14/chapter13/brick_-336_968_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_32_-272_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_-160_128_72.vmt
```

```
materials/maps/dm_chapter14/metal/metalwall031a_-544_304_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_-152_-56_72.vmt
materials/maps/dm_chapter14/metal/metalwall031a_-328_120_72.vmt
materials/maps/dm_chapter14/cubemapdefault.vtf
materials/maps/dm_chapter14/c248_-56_72.vtf
materials/maps/dm_chapter14/c408_672_72.vtf
materials/maps/dm_chapter14/c32_-272_72.vtf
materials/maps/dm_chapter14/c-328_520_72.vtf
materials/maps/dm_chapter14/c-544_304_72.vtf
materials/maps/dm_chapter14/c-152_-56_72.vtf
materials/maps/dm_chapter14/c-160_128_72.vtf
materials/maps/dm_chapter14/c-328_120_72.vtf
materials/maps/dm_chapter14/c696_-56_72.vtf
materials/maps/dm_chapter14/c-336_968_72.vtf
materials/maps/dm_chapter14/c848_1104_72.vtf
materials/chapter13/brick.vmt
materials/chapter13/brick_detail.vtf
materials/chapter13/brick_diffuse.vtf
materials/chapter13/brick_normal.vtf
```

Notice that the files you specifically imported are listed above. As for all of those additional `vtf` texture files, they are the cubemap images that you generated in Chapter 14. The numbers for some of those `vtf` texture files listed above may be different because they were automatically generated by the game, but they should be similar.

One other command that you can use with Bspzip can extract the files from the BSP file. So, not only can you add new files, but you can also take out existing files. This can be useful if you want to include a custom map element into your map that was used by someone else. You can extract all the files and include an individual file with your BSP file. Just remember to give the original author credit for his hard work.

To extract all the files from a BSP file, enter the following command into the Command Prompt:

```
"%sourcesdk%\bin\bspzip" -extract dm_chapter14.bsp
```

This extracts all the files from the BSP file into folders that are relative to the original files location. These extracted files are all of the files that were compressed into `dm_chapter14.bsp`.

When you are done working with the Bspzip program, close the Command Prompt window.

### Pakrat

Pakrat is a stand-alone Bspzip utility that has a graphical user interface (GUI). This means you are working with a program that has buttons and readouts that are easier to follow than the command-line processes of the Bspzip utility.

Version 0.95 of this utility is included on this book's CD. The installation file includes a `readme.txt` file that shows you how to install the utility on your computer. However, I quickly walk you through the process here as well in case those instructions are not clear.

As indicated in the installation notes for this file, Pakrat is a Java application, and it requires that you have the Java Runtime Environment (JRE) installed on your computer. You can download the JRE from `http://java.sun.com/`. From the Java Technology Web site, choose Downloads➪Java SE and follow the links for the download of the installation file for your operating system.

After you've installed the JRE, follow these instructions to install Pakrat:

1. **Open the file** `pakrat-095.zip` **with your file compression utility.**

   If you don't have a file compression utility that recognizes this file when you double-click it, I recommend installing WinZip as found on the CD media in the back of this book. It comes with a free trial.

2. **Extract the file** `Pakrat.jar` **to your** `sourcesdk/bin` **folder.**

   Select `Pakrat.jar` and click Extract within your compression utility. Navigate to the folder `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\sourcesdk\bin`, where *`Valve`* is your installation folder and *`username`* is your Steam account username. Press Extract.

   You can extract `Pakrat.jar` anywhere you want. It just makes the most sense to place this file in the same place as all the other modding executable files. If it's easier for you, go ahead and extract this file to your Desktop. Then it will be readily available for use.

3. **Close your compression utility.**

Now that Pakrat is installed, you can launch the utility:

1. **Navigate to the** `Pakrat.jar` **file.**

   If you installed the file where I recommended, then open Windows Explorer by choosing Start➪All Programs➪Accessories➪Windows Explorer. Then navigate to the installation folder `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\sourcesdk\bin`, where *`Valve`* is your installation folder and *`username`* is your Steam account username. Here you will find `Pakrat.jar`.

If you installed the game by downloading it via the Steam client, then your installation folder will be *Steam*. However, if you installed the game from CD or DVD disks, then your installation folder will be *Valve*.

2. **Launch** `Pakrat.jar` **by double-clicking it.**

   You should see two windows, as shown in Figure 15-4.

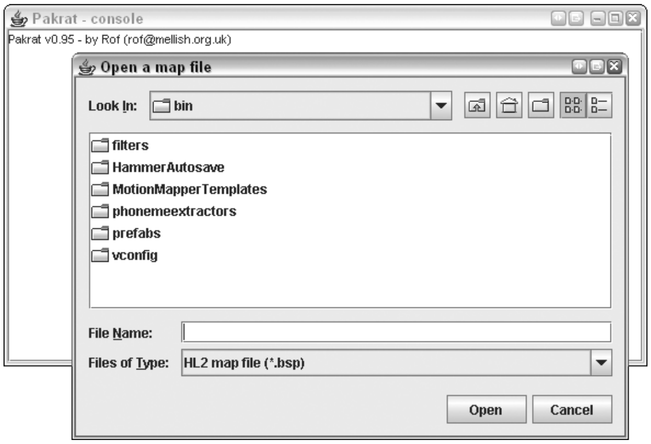


**Figure 15-4:** Launch Pakrat and select your BSP file.

   The first window that opens prompts you to select your map file. This window wants you to select the BSP file into which you will be importing your custom files. This window works much like Windows Explorer, so navigating through the different folders and files is fairly simple.

3. **In the Open a map file window, navigate to** `dm_chapter14.bsp`**.**

   You should be able to find it in `C:\Program Files\`*`Valve`*`\Steam\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\maps`, where Valve is your installation folder and *`username`* is your Steam account username.

4. **Select** `dm_chapter14.bsp` **and click Open.**

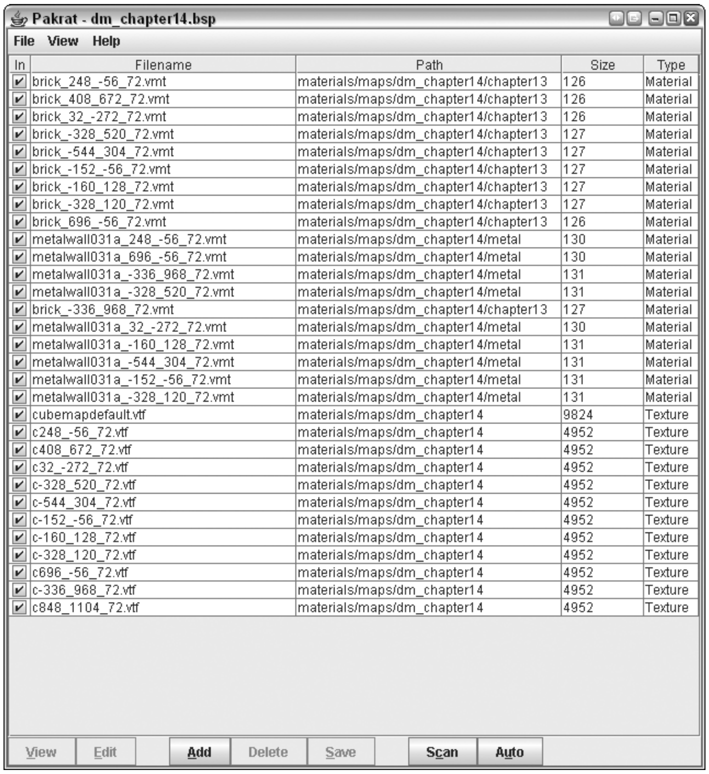
   The file opens and a new window displays all of the files already included in your BSP, as shown in Figure 15-5.

At the bottom of the window are number of buttons that can help you work with your BSP:

- **View:** Select a file and click this button to view the contents of the selected file. The contents displayed will either be the actual content like that of a text file, as in the case of the VMT file, or be the details of the file if it's an image, as in the case of the VTF file.

- **Edit:** Select a file and click this button to edit the file, such as changing its folder location or name. The contents of a selected file cannot be changed.
- **Add:** Add a new file to the BSP.
- **Delete:** Remove a file from the BSP.
- **Save:** Save the changes made to the BSP.
- **Scan:** Scan the map information within the BSP for custom files that might need to be included.
- **Auto:** Automatically scan your BSP and add the custom files.

Your first thought might be to click Auto and let Pakrat import all the files for you automatically. However, doing so is problematic. For some reason, when you click the Auto command, Pakrat inserts your first file in all capital letters. In this case, that file is the Brick material file, and it would be inserted as `materials/CHAPTER13/BRICK.vmt`. This is obviously incorrect and would cause your custom texture to be missing from the game when played.



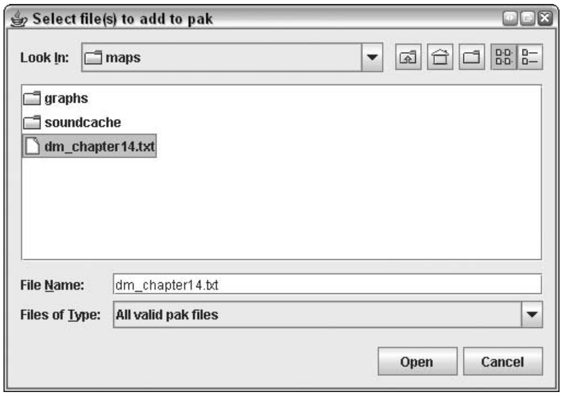
| In | Filename | Path | Size | Type |
|---|---|---|---|---|
| ✓ | brick_248_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_408_672_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_32_-272_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_-328_520_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-544_304_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-152_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-160_128_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-328_120_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_696_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | metalwall031a_248_-56_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_696_-56_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_-336_968_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-328_520_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | brick_-336_968_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | metalwall031a_32_-272_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_-160_128_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-544_304_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-152_-56_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-328_120_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | cubemapdefault.vtf | materials/maps/dm_chapter14 | 9824 | Texture |
| ✓ | c248_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c408_672_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c32_-272_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-328_520_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-544_304_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-152_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-160_128_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-328_120_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c696_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-336_968_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c848_1104_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |

**Figure 15-5:** Pakrat displays all of the files already included in your BSP.

So, instead, the solution is to add each file individually. This might sound tedious, but it's still easier than adding them from the command prompt line-by-line. Here's how to add new files to your BSP:

1. **Click Add.**

   Doing so opens a file explorer window (see Figure 15-6), which lets you navigate the files on your computer and select the file to be added.



File Name: dm_chapter14.txt
Files of Type: All valid pak files

**Figure 15-6:** Select and add each custom file created for your map.

2. **Navigate to the maps folder where your BSP file is located, select the README file `dm_chapter14.txt`, and click Open.**

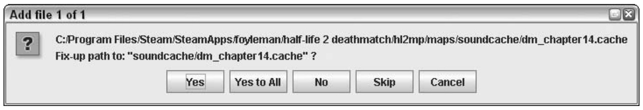
   Your README file is located in `C:\Program Files\`*`Valve`*`\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\maps`, where *`Valve`* is your installation directory and *`username`* is your Steam account username.

3. **Repeat Step 2 for all the files you want to add to your BSP.**

   Refer to the list of files outlined earlier to locate and add to your BSP package.

   If at any time you are asked if you want to *Fix-up path* for any of the files you are adding, as shown in Figure 15-7, click Yes. This question from Pakrat means that Pakrat is trying to include the full file path into your package when the file path should actually be relative to your installation of *Half-Life 2*. Clicking Yes fixes the issue.

**Figure 15-7:** Click Yes any time you are asked to fix up a path by Pakrat.



C:/Program Files/Steam/SteamApps/foyleman/half-life 2 deathmatch/hl2mp/maps/soundcache/dm_chapter14.cache
Fix-up path to: "soundcache/dm_chapter14.cache" ?

TIP

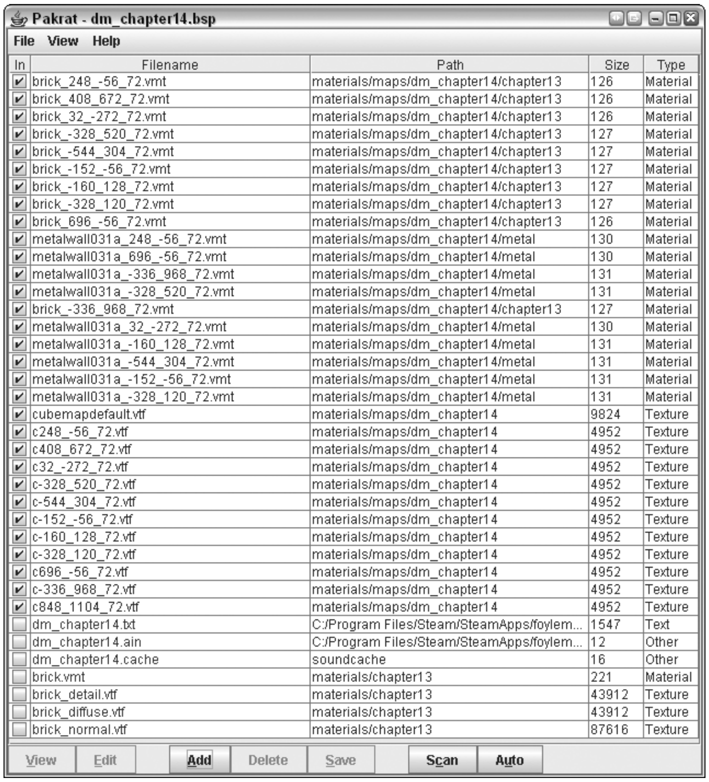
If you don't see the file you need to add, then change the Files of Type to All Files in the Select Files window. This will most likely happen when you attempt to add the `soundcache\dm_chapter14.cache` file. Changing the type of viewable file to All Files will make the hidden file visible, and you'll be able to add it to your BSP package.

When you're done adding files, they should all be listed at the bottom of your Pakrat files list. They are listed in blue and without a check mark next to their names, as shown in Figure 15-8. The check mark is an indication that the file has been added to your BSP and saved as such. Because you haven't saved your changes yet, the files are not checked off.

4. **To save your changes, choose File➪Save BSP.**

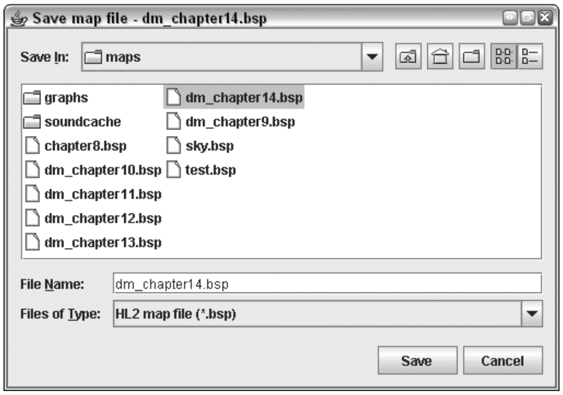
   This opens a save window (Figure 15-9) asking you the name of the BSP you want to save. This is an opportunity for you to save the final BSP with a new name. However, renaming isn't necessary in this case.

5. **Select your existing file `dm_chapter14.bsp` and click Save.**


Pakrat - dm_chapter14.bsp
File View Help
View Edit Add Delete Save Scan Auto

| In | Filename | Path | Size | Type |
|---|---|---|---|---|
| ✓ | brick_248_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_408_672_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_32_-272_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_-328_520_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-544_304_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-152_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-160_128_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-328_120_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_696_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | metalwall031a_248_-56_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_696_-56_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_-336_968_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-328_520_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | brick_-336_968_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | metalwall031a_32_-272_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_-160_128_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-544_304_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-152_-56_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-328_120_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | cubemapdefault.vtf | materials/maps/dm_chapter14 | 9824 | Texture |
| ✓ | c248_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c408_672_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c32_-272_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-328_520_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-544_304_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-152_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-160_128_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-328_120_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c696_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-336_968_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c848_1104_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
|  | dm_chapter14.txt | C:/Program Files/Steam/SteamApps/foylem... | 1547 | Text |
|  | dm_chapter14.ain | C:/Program Files/Steam/SteamApps/foylem... | 12 | Other |
|  | dm_chapter14.cache | soundcache | 16 | Other |
|  | brick.vmt | materials/chapter13 | 221 | Material |
|  | brick_detail.vtf | materials/chapter13 | 43912 | Texture |
|  | brick_diffuse.vtf | materials/chapter13 | 43912 | Texture |
|  | brick_normal.vtf | materials/chapter13 | 87616 | Texture |

**Figure 15-8:** Your files are in the Pakrat list, but not yet listed as saved.



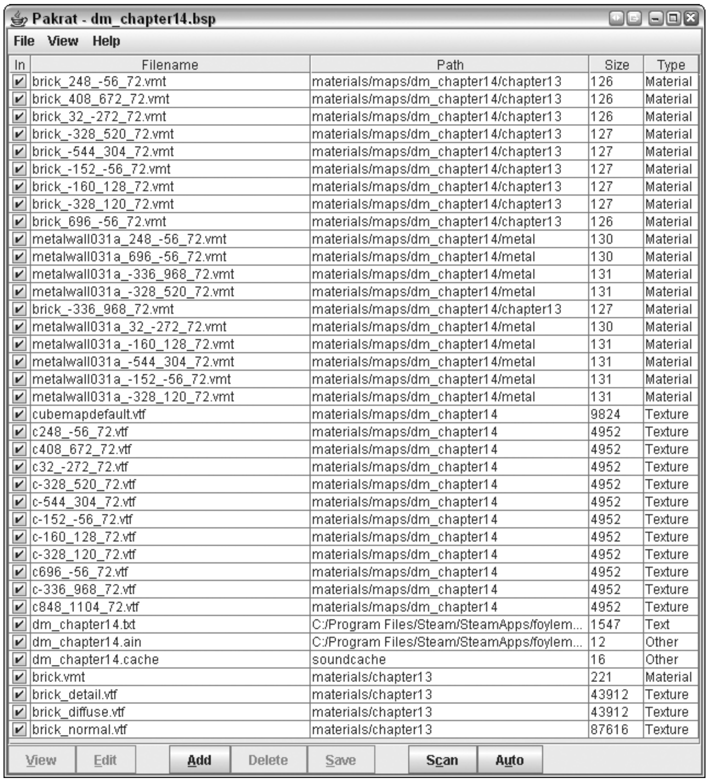
File Name: dm_chapter14.bsp

Files of Type: HL2 map file (*.bsp)

**Figure 15-9:** Save your altered map file over the existing BSP file for your map.

6. **When prompted about whether you want to overwrite the existing BSP, click Yes.**

   When the new BSP is saved, Pakrat reloads your BSP and shows that your new files have been included and saved. As Figure 15-10 shows, your BSP is now ready for distribution.



| In | Filename | Path | Size | Type |
|---|---|---|---|---|
| ✓ | brick_248_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_408_672_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_32_-272_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | brick_-328_520_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-544_304_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-152_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-160_128_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_-328_120_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | brick_696_-56_72.vmt | materials/maps/dm_chapter14/chapter13 | 126 | Material |
| ✓ | metalwall031a_248_-56_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_696_-56_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_-336_968_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-328_520_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | brick_-336_968_72.vmt | materials/maps/dm_chapter14/chapter13 | 127 | Material |
| ✓ | metalwall031a_32_-272_72.vmt | materials/maps/dm_chapter14/metal | 130 | Material |
| ✓ | metalwall031a_-160_128_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-544_304_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-152_-56_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | metalwall031a_-328_120_72.vmt | materials/maps/dm_chapter14/metal | 131 | Material |
| ✓ | cubemapdefault.vtf | materials/maps/dm_chapter14 | 9824 | Texture |
| ✓ | c248_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c408_672_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c32_-272_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-328_520_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-544_304_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-152_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-160_128_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-328_120_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c696_-56_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c-336_968_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | c848_1104_72.vtf | materials/maps/dm_chapter14 | 4952 | Texture |
| ✓ | dm_chapter14.txt | C:/Program Files/Steam/SteamApps/foylem... | 1547 | Text |
| ✓ | dm_chapter14.ain | C:/Program Files/Steam/SteamApps/foylem... | 12 | Other |
| ✓ | dm_chapter14.cache | soundcache | 16 | Other |
| ✓ | brick.vmt | materials/chapter13 | 221 | Material |
| ✓ | brick_detail.vtf | materials/chapter13 | 43912 | Texture |
| ✓ | brick_diffuse.vtf | materials/chapter13 | 43912 | Texture |
| ✓ | brick_normal.vtf | materials/chapter13 | 87616 | Texture |

**Figure 15-10:** Your BSP has now been updated with your custom files and is ready for the public.

7. **Close Pakrat by choosing File➪Quit and then confirm that you want to quit Pakrat by clicking Yes.**

## Zipping It Up

All of your files are now ready to send out to the Web sites so they can offer them to the public to download. However, you can't just give them a handful of loose files. You need to first compress them into a single file. In this case, that single file is a *zip file*.

Provided on this book's CD is a program called WinZip. This program compresses one or more files into a single, smaller file. This is perfect for offering files for distribution. If you haven't installed WinZip yet, install it now.

It's time to create your final distribution package. Here's how:

1. **Open Windows Explorer and navigate to the maps folder where your BSP file is located.**

   This folder is `C:\Program Files\`*`Valve`*`\SteamApps\`*`username`*`\half-life 2 deathmatch\hl2mp\maps`, where *`Valve`* is your installation folder and *`username`* is your Steam account username.

   This folder holds not only your BSP file, but also your README file and screenshot. These files will also be included in the final package.

2. **Select your BSP file, your README file, and your screenshot.**

   To select multiple files, hold down Ctrl and click each of these three files within Windows Explorer. Each file should be highlighted as it is selected.

   The files you want to select are

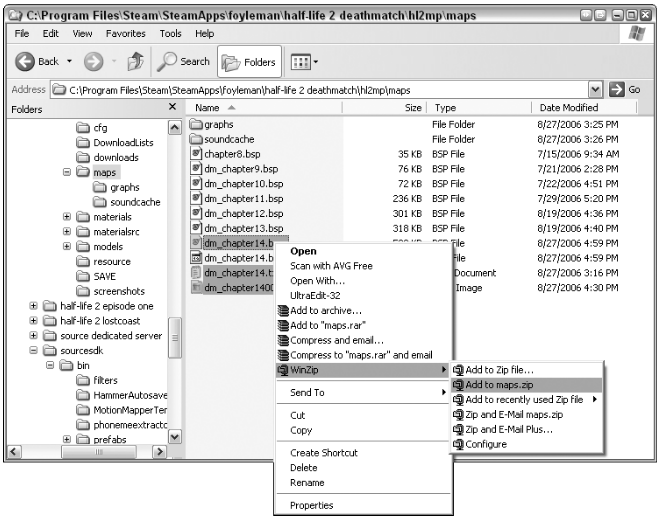
   - `dm_chapter14.bsp`
   - `dm_chapter14.txt`
   - `dm_chapter140000.jpg`

   TIP

   Although you have already included `dm_chapter14.txt` in the BSP package, not everyone is going to know that it's there. Including it in this zip file ensures that anyone can find it and read about what is contained within.

3. **Right-click one of these selected files and choose WinZip➪Add to maps.zip.**

   This new menu option is available after the installation of WinZip, as shown in Figure 15-11.



| Name | Size | Type | Date Modified |
|---|---|---|---|
| graphs | | File Folder | 8/27/2006 3:25 PM |
| soundcache | | File Folder | 8/27/2006 3:26 PM |
| chapter8.bsp | 35 KB | BSP File | 7/15/2006 9:34 AM |
| dm_chapter9.bsp | 76 KB | BSP File | 7/21/2006 2:28 PM |
| dm_chapter10.bsp | 72 KB | BSP File | 7/22/2006 4:51 PM |
| dm_chapter11.bsp | 236 KB | BSP File | 7/29/2006 5:20 PM |
| dm_chapter12.bsp | 301 KB | BSP File | 8/19/2006 4:36 PM |
| dm_chapter13.bsp | 318 KB | BSP File | 8/19/2006 4:40 PM |
| dm_chapter14.b | | File | 8/27/2006 4:59 PM |
| dm_chapter14.b | | File | 8/27/2006 4:59 PM |
| dm_chapter14.t | | Document | 8/27/2006 3:16 PM |
| dm_chapter1400 | | Image | 8/27/2006 4:30 PM |

**Figure 15-11:** Zip your selected files into a single zip file right from Windows Explorer.

Because `maps` is the folder you are currently in, the option to create a zip file from the selected files under the name of `maps.zip` is an option. This makes it very simple to create a zip file of multiple files at once. Later, you can rename this zip file to something more appropriate.

After you make this selection, a new file, `maps.zip`, appears in your maps folder. This is your distribution package.

4. **Right-click** `maps.zip` **and choose Rename. Rename this file to** `dm_chapter14.zip`**.**

   You want to rename this file to the same name as your map so it is easily identifiable.

   When you choose Rename from drop-down menu, the file becomes highlighted and the cursor is available so you can change the name of the selected file. Replace the name of the file and press Enter to apply the change.

That's it. You're now ready to get your map out to the world for everyone to enjoy. Just by sending this zip file to your friends and to Web sites, you can distribute your custom game content.

## Distributing the Goods

Getting your map out to the public might seem easy at first. However, when you start looking at free distribution Web sites, you'll find it isn't always so easy to figure out who to contact or where to go — unless you have a little basic direction.

The first place you want to start is with Web sites that already offer downloadable content for *Half-Life 2.* Because Web sites can change so frequently, I've set up a Web page where you can find a list of *Half-Life 2* map submission links at `www.half-life2modsfordummies.com`.

TIP

Internet search engines are a great place to search for Web sites that offer *Half-Life 2* custom content and maps. Use your favorite search engine to locate more Web sites by using keywords such as *Half-Life 2, custom content, custom maps, downloads, mapping,* and *editing.*

When you visit any sites, begin by looking for a link that refers to submissions of content. Look in the Downloads section as well as the Contact pages.

If you still can't find anything that looks like a submission link, contact the Web site administrator by using whatever contact links you can find. As a Web site owner myself, I don't mind being contacted with questions, especially if it means adding more content to my site. You must remember that these Web site owners and admins love to add content because more submissions usually means more visitors. As long as you've included your README file, screenshot, and game content files, your mod is sure to be listed on these sites.

When contacting Web sites, tell the administrators who you are and what you would like to provide them. Tell them that you're a mapper or modder and that you would like them to offer your custom game content for download on their Web site. Don't include the file with your e-mail at this time. Let the recipients get back in touch with you with instructions on how you should supply the file to them.

You're now well equipped to make maps and mods for *Half-Life 2* and get your custom material out to the public.

# Part V
# The Part of Tens

The 5th Wave By Rich Tennant

" Ms. Gretsky, tell the employees they can have Half-Life 2 on their computers again."

## In this part . . .

After getting this far, you must be raring to go further. In this part, I offer some tips that will keep you out of trouble and then help you discover more about what modding can do.

Don't be concerned that you are reaching the final pages of the book because your education doesn't stop here. Take a look online at all of the modifications you can download and enjoy. Then, take a look at how they were created. It's amazing what others can do with mods, and you can figure out how to do it, too.

# Chapter 16

# Ten Great Tips and Tricks

To help you further your knowledge and jump over some of the beginner hurdles, this chapter gives you the top ten tips and tricks for modding *Half-Life 2.* These tips answer some of the most commonly asked questions by people just starting out with mapping and modding.

## Coming Up with Original Ideas

You have the knowledge to get started on your own mod for *Half-Life 2.* Although you're ready to jump into editing, maybe you don't know what to create. This is a very common problem among modders and mappers. Either you start with an inspired idea to work, or you start inspired to work without an idea of where to start. It happens to professionals and beginners alike.

To come up with ideas, use what's around you. The first place you should look is right where you're sitting. Try building a map of your house, school, work, town, or other location with which you're familiar. Most of the best maps were created around actual places. If where you live doesn't inspire you, turn toward television, books, or the Internet. With so much technology at your disposal, you can look at environments and places located anywhere in the world without leaving your own home.

Another way I like to start my personal projects is to think of something that hasn't been done before. For example, I once made a map for *Solider of Fortune* that was the first for the game series to incorporate an elevator that could stop and start at the floor of the player's choosing. The player wasn't limited to a preset path for the elevator to travel. So, ask yourself, "What new thing can I add?" When you have the answer, add it.

Finally, when you have no more ideas to work on but you really have an urge to do something, try asking members of a *clan* (gaming group) or other gaming group if they would like a custom addition to the game. Let them know you can build it based on their requests. Clans often have a level or game type that they prefer playing more than any other. However, after hours of playing the same levels, the members yearn for variety. That's where you can step in and create that spectacular add-on they've been begging for.

## Planning Your Build

I don't care how experienced you are at building maps or mods: If you don't have some sort of plan for what you're constructing, you're eventually going to have to rebuild part of your map.

I once worked on a city map for a mod team. This map called for a couple of city blocks of buildings with streets and alleys that ran between them. I figured that I could build this map without planning distances between buildings, the number of buildings, and other items that would fill this map.

Each build resulted in streets that weren't the correct width, streets that couldn't be blocked off by any logical means, and other obstacles. It wasn't until a month later and my sixth version of this map that I started to get things right.

TIP

Maps can't just go on forever. Eventually, you must define boundaries, and streets are often the most difficult to block off. Sure, you could just stop the map, but you want to do something that's logical. Try considering roadblocks, a broken bridge, a disabled vehicle, a fire, or a similar impassible obstacle.

The solution to avoiding these errors is to plan your project before you begin. Draw a diagram of what you're trying to accomplish. Expand the diagram to include features, images, and other items that you specifically want added. If you're mapping, get yourself some graph paper and plot the building and other structures that you plan to build. Also consider taking or finding photos of what you want to build to further illustrate what you want to accomplish.

With everything properly planned out ahead of time, you'll find that your project goes together much easier and quicker. You won't have to rebuild as much, and you'll be done in no time.

## Designing Minimally

It's an awful thing when you construct an entire map, compile it, test it, and then realize that everything runs too slowly in the game because the level contains too much detail. Although game engines have rapidly come a long way in becoming powerful delivery agents of 3D worlds, they can't yet handle everything you might want to throw at them. This is where designing minimally comes in handy.

You can have all those details that you want in your level. However, you don't have to make them into solid objects or brushes that the game has to overwork to deliver to your screen. Instead, let the textures do the work for you.

Think of the large outdoor crates that I show you how to construct in Chapter 12. If you look at the crates within the game, they have dimension, indentation, and a realistic presence — things that make the crates look like they were constructed from a lot of tiny brushes. However, all you had to do for each crate was build a cube and apply a texture. This texture then did the rest of the work for you.

Where can you find textures for this purpose? You can use many different types of resources. For example, for the bricks used in Chapter 13, I took pictures of the exterior of a local building. Here are a few other good ways to acquire textures:

- **Purchase textures from the Internet.** Searching the Internet for "buy textures" turns up pages of results.
- **Buy images.** You can often find CDs of images at your local computer stores, or you can look on the Internet for sites that sell individual photos. Just make sure that you read the licensing information before distributing them to your friends.
- **Use textures from other games, such as earlier versions of *Quake, Half-Life,* or just about any other game.** Often, I choose games that support modifications, such as the *Quake* series, because I know it will be easier to extract those textures from the game. If a game doesn't support mods, you might not be able to get those textures out of the game.
- **If you have experience with modeling, you can model your own scenes and render the images required for your textures.** I have, on occasion, created my own models, arranged them in a scene, rendered an image of that model, and converted it into a texture for the game.

When you don't have what you need, think outside the box. Or, in this case, think outside the program. Look online or outside for ways to reduce your brush count in the game while still delivering details in your level.

## Following Examples

You might not have realized it yet, but the SDK download and installation already comes with many example maps for you to look at. Several levels and their compiled counterparts are supplied to you.

To find this example content, take a look through the folders and files that were installed with the game. Look through the folder

```
C:\Program Files\Valve\SteamApps\username\sourcesdk_content
```

where *Valve* is your installation folder and *username* is your Steam account username to find a ton of great things you can learn from.

Say that you are trying to provide your deathmatch level with more professional lighting, but you don't know how to lay out your lights for your situation. Try loading some of the example maps into the game and looking through them for similarities and things you like. Then open the source file in the editor and see how the developers did it.

To look through the deathmatch maps, try these steps:

1. **Open your file explorer program.**

   To use Windows Explorer, choose Start➪All Programs➪Accessories➪Windows Explorer.

2. **Navigate to the folder**

   ```
   C:\Program Files\Valve\SteamApps\username\sourcesdk_content\hl2mp\mapsrc
   ```

   where `Valve` is your installation folder and `username` is your Steam account username.

   Here you will find several example maps and their compiled BSP files.

3. **Copy the contents of this folder.**

   Select all the files in this folder and press Ctrl+C to copy them.

4. **Navigate to your deathmatch maps folder**

   ```
   C:\Program Files\Valve\SteamApps\username\
             half-life 2 deathmatch\hl2mp\maps
   ```

   where `Valve` is your installation folder and `username` is your Steam account username.

5. **Paste your copied files in this folder by pressing Ctrl+V.**

6. **Load and play these custom maps in the game.**

   Start the game and open the console. With the same `devmap` console command you use in earlier chapters, load each map one at a time. Explore the maps and take notes on what you do like and don't like.

   When you're done playing the maps and taking notes, load those same maps in the Hammer editor. Here, you can look at how the maps were constructed and see how the developers did what you like — and how not do those things you didn't like.

The same goes for other parts of the game. Scripts, GUIs, and other things used to make a level work are included in the `source_sdk` folders. Even the models that are using the game can be imported into a 3D modeling program. The possibilities are staggering.

Sometimes, checking out examples can inspire you or drastically reduce your build time. The maps that come with the game are perfect for this. The developers of the game have put a lot of neat tricks at you fingertips.

## Using Prefabs

If you're making a map or series of maps that reuse the same construction of brushes, turn it into a *prefab,* which is like a tiny map that can be imported into any map that you're working on. Usually, prefabs are made up of guard towers, buildings, or other commonly used structures. This way, you have to build them only once, and then you can easily duplicate them whenever you need to.

To create a prefab, select the group of brushes that you want to save. Choose Tools➪Create Prefab. Then save the file under a name of your choosing in your prefabs folder. The prefabs folder is located at

```
C:\Program Files\Valve\SteamApps\username\sourcesdk\bin\prefabs
```

where `Valve` is your installation folder and `username` is your Steam account username. The file is saved in the map format VMF.

To load a prefab is just as simple. Select the Entity Tool. Click and place your insertion point in your 2D viewports with the Entity Tool. From the New Objects Group on the right side of the editor, select Prefabs from the Categories list. From the Objects list, select your prefab of choice. The prefabs listed here are those available in the `sourcesdk\bin\prefabs` folder. Press Enter to insert the prefab just as if it were an entity. Then you can select, move, rotate and do just about anything you need to do with your new prefab.

TIP

When you insert a prefab, contents are grouped. If you want to make changes to the prefab, ungroup everything by selecting the prefab and pressing Ctrl+U. Then you can manipulate each piece of the prefab however you need.

You can also find prefabs available online, usually at the same places where you can find custom maps and mods. Sometimes, mappers like to offer their prefabs for download, just like their maps. This way, their great constructions can find their way into other maps. Just remember that if you do use a prefab in your map, you should give credit to the author in the README file (which I discuss in Chapter 15).

## Meshing Objects

If you're trying to create a more flexible brush in your map, such as terrain with hills and valleys, consider displacement creation. In this process, you take a seemingly simple block and divide it into a flexible brush that you can use to make your hilly terrain.

1. **Create a large brush for your terrain.**

   This will be your ground, so make it large enough to cover the area. Refer to Chapter 5 for creating brushes.

2. **Open the Texture Application Tool by pressing Ctrl+A.**

3. **Click the Displacement tab at the top of the Face Edit Sheet.**

   Here you can split your brush to make it flexible.

4. **Press Create under the list of Tools buttons.**

   This tells the editor that your selected brush is to be displaced with the settings you are about to enter. It also pops up another window asking you for the power of displacement.

5. **Enter a value between 1 and 4 in the Power attribute and then click Apply.**

   This Power attribute determines how many times the displacement brush will be divided. For now, enter a value of 3; you can experiment later with different values. You'll quickly get the hang of how these different values affect the displacement of your brush.

6. **Select the Paint Geometry tool within the Displacement window.**

   A new window comes up, offering you methods of changing your displacement brush. This tool allows you to click and drag points of your displaced brush to create those hills and valleys. For now, leave all the settings at default to create some hills.

7. **Left-click various points of your displacement brush to raise that area higher than the rest.**

   This creates a nice hill in your terrain. If you want to lower the terrain, change the effect from within the Displacement Paint Geometry window.

8. **Try different options and adjust your displacement brush to get a sense of what happens with the use of each function.**

   By experimenting, you quickly get the hang of how to create a great-looking terrain for your map.

If you're working with a large outdoor area, you'll find that the displacement brush isn't divided enough for you to create smooth terrain. The solution is to create several smaller displacement brushes and sew them together. After working your displacement brushes into the desired shapes, place these brushes next to each other. Click Sew in the Tools of the Displacement tab to force all the edges to meet and close any visible gaps.

With this option for your maps, you can really go wild and create some nice-looking levels. However, be careful not to overdo it: All this added geometry adds up. Eventually, the game starts slowing down because the game has so much to calculate that your processor can get bogged down, and the game won't run smoothly. So, as you go along with your terrain (or whatever you're creating), test it in the game to make sure things are progressing as expected.

## Putting on a New Skin

Skinning is a popular way to change existing models already in the game. Basically, *skinning* is the replacement of existing textures with new ones. You can look into the game files for the existing textures used on weapons, players, or any other entities and models, and you can replace these textures with your own creation. You can effectively change the look of the entire game.

One such example is to change a character skin so that it looks like you — the real *you* — in the game. This means locating the face texture for one of the player models. Load this texture into Photoshop and overlay your face in place of the original. Then just save that new texture and check it out — you're in the game!

## Measuring the Player

A very common question in all the mapping forums is, "What are the player dimensions?" Many people want to know how the game relates to real life in order to properly plan their maps. Perhaps they want a life-size room or building. These dimensions could help them achieve that.

The basic measurements of the player in the game are as follows:

- **Player's height:** 73 units
- **Player's height while crouched:** 37 units
- **Player's width:** 37 units
- **Player's maximum step without jumping:** 18 units
- **Player's maximum jumping height:** 56 units
- **Player's maximum jumping height while crouched:** 21 units
- **Player's maximum jumping distance:** 176 units
- **Player's maximum jumping distance while crouching:** 62 units
- **Player's highest fall without damage:** 240 units
- **Steepest angle that a player can climb:** 45 degrees

Before I list the conversion of units to inches, keep in mind that this is a game. The laws of nature do not apply in the virtual world. Therefore, you really should make your map so that it looks good — not so that it maps to real-world specifications.

With that in mind, the unit-to-measurement conversion has been averaged as follows:

3727 units = 100 meters

1 unit = .95 inches (You can round to 1 inch.)

## Finding More to Mod

If you're looking to further modify a game, you can do a lot more than what I tell you about in this book. I was able to cover only mapping, scripts, textures, and skins in the page space allotted. However, modding doesn't stop there.

### Listening in

You can add new audio or even change the audio that already exists in the game. Weapons, background music, pickups, and several other objects in the game create sound. These are just a few places to start.

The audio files in the game can be found in `sound` directory of your mod. The files are in WAV format and saved at 11/22/44 kHz 8/16 Bit PCM, which is a fairly normal audio format. Music on your audio CDs is saved as 44 kHz 16 Bit WAV files. All you need is an audio-editing program that can manage WAV files, and you are ready to go.

### Modeling and animating

If you've read this book to this point, you've seen the models in the game: the player, weapons, and several other entities. These models were created in modeling programs such as MilkShape 3D, Maya, and 3D Studio Max.

You can create your own models and import them into a game. This might take additional time and practice on your behalf, but if you can create a model, you will be on your way to mastering games.

After you discover creating custom models for the game, you can get into animating them. Think of what happens when your weapon runs out of ammo: The player reloads the weapon, and you see an animation of this reloading.

If you want to get into modeling and animating for games, I recommend that you start with the Internet. Locate some online tutorials specific to *Half-Life 2.* They can give you an idea what software you will need to pick up and what more you have to figure out.

## Looking for Help

The topics I don't cover in this book are most likely to be available by way of online tutorials. My education into the world of game mods started with online tutorials. When I surpassed the tutorials that were available, I started making my own.

You can find various Web sites that offer both written and video tutorials on a variety of modding subjects. For those tasks that aren't covered by tutorials, the forums offer a wealth of information and means to ask for more.

If you're looking for more help, start with your favorite Internet search engine. You're sure to come up with a few pages of results that give you what you are looking for.

I created an additional source for finding Web sites that offer tutorials information for modding games. Visit `www.half-life2modsfordummies.com` and follow the links provided there.

# Chapter 17

# Ten Great Mods to Learn From

A number of great mods are already available on the Internet. They're not only great fun, but they're also a great way to gain knowledge. You can download a mod, open the contents, and explore the changes that were implemented to make what someone created possible.

In this chapter, I selected ten great mods and dissected them. My goal is to give you ideas on what you can do and create. In some cases, I even tell you what changes the mod authors made to the game so you can experiment. However, providing inspiration isn't all I'm trying to do. I also want to tell you what the modders could have done to further improve the mods that they released to the public. Hopefully, the result will be a lot more mods for all to enjoy.

A portion of the mods listed in this chapter are included on this book's companion CD. Feel free to install them as they are discussed. Play around with them to see what they can do — and maybe feel inspired to create your own!

TIP

For more information on how these and other mods were created, examine their contents. Compare the files in the mod with those of the original game. This will truly help you to solve more riddles when creating your own mods.

## Crossfire 2 b3

| | |
|---|---|
| Mod name | *Crossfire2* |
| Version | b3, release date unknown |
| Author | Unknown |
| E-mail | Unknown |
| Web site | Unknown |
| Filename | `dm_crossfire2b2.bsp.rar` |

Inspiration for your creation can come from anywhere. In the case of this level, the author drew inspiration from a custom level that was created many years ago for the original *Half-Life* game. *Crossfire2* is a remake of *Crossfire,* one of my favorite deathmatch maps.

The map has received subtle updates while retaining its original look and feel. Additions, such as combine gates and transparent shield barriers, were added. The *Half-Life 2 Source* engine makes this level shine.

### Possible pitfalls

Remaking a level means that layout for your level is already done; however, it also makes room for more scrutiny. Because so many people have played the original countless times, nothing short of perfection will be expected. So, be careful when remaking levels. When done well, your re-creation will receive much praise.

## Urbanes Deathmatch 2

| | |
|---|---|
| Mod name | *DM Urbanes* |
| Version | v2, released 03/17/2005 |
| Author | DR@GONFLY |
| E-mail | voorkaj@hotmail.com |
| Web site | Unknown |
| Filename | `dm_urbanes_v2.rar` |

*Urbanes* is a beautiful deathmatch level with an original design. Elements that aren't normally found in deathmatch levels, such as plasma balls, have been used here to give game play a unique feel. The level appears to have no bottlenecks, which means that players won't be caught in areas of the level in which they cannot escape.

Another unique inclusion in this level is a teleporter. In one of the larger rooms, a beam of light shoots through the floor. When the player steps into this light, he is teleported to a platform that provides him with a sniping advantage over the other players. However, because there should be no advantage without disadvantage, the only escape from this platform is a painful jump to the floor.

### Possible pitfalls

Version 2 addresses some important issues that were overlooked with the first release. When adding so many custom elements to a level, your frame rates grow, resulting in slow, laggy game play. Optimization becomes extremely important. Based on such criticism to the first release, version 2 addresses these frame rate issues. The map was optimized, and the original lag is gone.

To avoid these problems, test your level before its release. Use the console commands explored in Chapter 10 and look for potential problems. If possible, play your new level with friends to help you look for these problems. Then you have the opportunity to fix any issues before the public points them out to you.

## Dystopia

| | |
|---|---|
| Mod name | *Dystopia* |
| Version | Demo update 4, released 09/10/2005 |
| Author | Team Dystopia |
| E-mail | `dystopia@dystopia-mod.com` |
| Web site | `www.dystopia-game.com` |
| Filename | `dystopia_demo_client_full_u4.exe` |

*Dystopia* is actually a complete modification of *Half-Life 2*. The game comes with its own models, loading screens, menus, and more. However, what I want to focus on is that this modification is focused around a compilation of maps.

Creating a group of levels that center around a central theme can create a specific fan base. *Dystopia,* for instance, is based around high technology and what Team Dystopia calls *cyberpunk.* It's this theme that everything is built around, including levels, skins, models, and more.

### Possible pitfalls

Vast creations often require vast resources, and modding is no different. An undertaking such as *Dystopia* requires a team effort to complete. So much custom content goes into a themed set of levels and accompanying assets that one person can't likely complete this alone.

Mod teams aren't easy to establish or maintain. This is a common reason for their failing and why you don't see too many completed mods available. If you plan on creating a mod and managing a team, also plan on that management taking a lot of your time. Despite a lot of work, the payoff in the end is be greater when you get to hand off your creation to the rest of the gamers out there who end up loving it as much as you.

## High-Resolution Skins

| | |
|---|---|
| Mod name | *FakeFactory & Ogg's Cinematic Mod* |
| Version | 2.32, released 08/05/2005 |
| Author | Team Dystopia |
| E-mail | `info@madservice.de` |
| Web site | Unknown |
| Filename | `fakefactory_cinematic_2.32_full.zip` |

A quick way to dress up a game is to create new skins for the models. *FakeFactory & Ogg's Cinematic Mod* is a package of skins and other additions that replace many of the character faces, textures, sounds, and more in the game. These faces in particular were reproduced in a much higher quality than the originals, which makes them look much more realistic.

Adding realism to the game can breathe new life into it. Installing these high-resolution face skins might have you reloading the game and playing it again just to experience the higher quality provided with this mod.

To create such changes, different methods could have been applied. One such method could simply be using Photoshop. The creator could have layered other faces or images over the originals to add to the complexity of the skins. However, when working with faces — as opposed to weapons or outfits — this method is often more difficult.

Another method is to re-create the characters of the game within a 3D modeling program, such as Maya or 3D Studio Max. Within these programs, you can re-create the faces and add the missing details. After you're done, you can export these new face skins to an image file that can be used within the game. This might be an easier method, but it requires the know-how of working with modeling programs.

### Possible pitfalls

Higher quality usually means that your computer requires more resources. In this case, you need enough memory in your computer to handle these large image files. Each image needs to be preloaded by the game be displayed. This is perhaps why Valve didn't provide them upon release of the game.

Such additions to the game can exclude potential fans of your work because if those who download and install your mod don't have the necessary equipment to properly load and display these larger images, they can't partake in your creation.

## Razor's Weapon Sound Package

| | |
|---|---|
| Mod name | *Half-Life 2 Weapon Sound Pack* |
| Version | 2, released 11/03/2005 |
| Author | Razor |
| E-mail | Unknown |
| Web site | Unknown |
| Filename | `razor_hl2_weaponsoundpackv2.zip` |

If you've ever watched a scary movie with the sound turned off, you've likely noticed that the movie doesn't seem as scary or seem as interesting. The same holds true for games. A game is much more interesting with sound, such as background effects, theme music — or, in the case of this mod, weapons.

In this modification, Razor altered the sounds of the weapons. He added more reverb in some cases; for others, he changed them completely. The result is a new feel for how the game plays.

### Possible pitfalls

When you change the sounds used by a well-known game, the result can be a dramatically changed feel for the game. Many gamers might not like this change and might also take offense.

Consider if you changed the music in the movie *Halloween.* This movie's theme song and background music are unique. If you change them, the movie would seem different, and diehard fans might dislike what was done.

REMEMBER

What you change or add to a game might not go over well. Consider testing your change on some friends and getting their reaction before releasing it to the public.

## Source Racer

| | |
|---|---|
| Mod name | *Source Racer* |
| Version | Beta 2, released 01/30/2005 |
| Author | Stefan Lednicky and Team |
| E-mail | `lednicky@gmail.com` |
| Web site | `sourceracer.co.uk` |
| Filename | `srclient20.rar` |

Sometimes designing new maps can lead to the need for code changes. You find that you created a great map, but certain weapons, settings, or some other change needs to be made to make your map that much better. This is how *Source Racer* might have been created.

*Source Racer* is based on racing vehicles. The maps are built like an obstacle course that you race around multiple times. The code changes provide the game with a clear starting and finishing point as well as keep track of laps around the tracks. To top it off, additional pickups were created to give the vehicles another strategy element, such as a burst of turbo or a drop trap for the other players.

Other additions to the game include

- **Levels:** Five custom levels come with the installation. Also included is one uncompiled map, which I assume is here to encourage outside development of maps (meaning that you, too, could create a custom level for this modification).
- **Textures:** Quite a few custom textures were created for this mod, such as an arrow indicator that helps players know where they are to drive next.
- **Models:** The most notable additions to the game are the special pickups that available around the track. These pickups are models that can be dropped into the game, like entities.
- **Resource files:** Resource files cover a broad range of changes made to the game. They include custom fonts for text used by the Heads Up Display (HUD) or other areas of the game as well as menu and HUD changes. Reviewing the resource files within a plain text editor can help you to see what changes were made and perhaps how to make them yourself in your own mod.
- **Scripts:** The scripts in this modification primarily affect two components of the game: the custom sounds and the vehicle pickups. Take a look through these scripts with your text editor for information on the changes that were made.
- **Sounds:** Several custom sound effects were added to the game. If you can create your own sounds, adding them to the game isn't overly difficult. Refer to the source files for this modification as well as some of the information in Chapter 16 to see how to make and add your own custom sounds to a game.

### Possible pitfalls

A lot of work is involved in the upkeep of a mod and its team as the original team leader, Stefan, realized. He eventually had to give up his position and pass it onto a new team leader, DeadlyDan, and the mod has since been slow to release updates. You can't blame DeadlyDan, however. Mods are time consuming and require a lot of commitment.

## BlockStorm

| | |
|---|---|
| Mod name | *BlockStorm Modification* |
| Version | Beta 1, released 08/24/2006 |
| Author | Baxayaun |
| E-mail | `Baxayaun@gmail.com` |
| Web site | `www.sdk-project.com` |
| Filename | blockstorm20beta201.rar |

The author of *BlockStorm*, Baxayaun, is a very clever modder. He is involved in many different *Half-Life 2* modifications, as his Web site points out.

What makes *BlockStorm* so unique is that it doesn't look at *Half-Life 2* in the traditional sense. Baxayaun doesn't use the Source engine as a means to create a special first-person shooter game. Rather, he uses it to re-create a 2D game within a 3D world.

*BlockStorm* is a remake of the classic game *Arkanoid.* The idea of the game is to move bounce the moving ball around the map to break all the blocks in the playing field. Doing so moves you onto the next level, presenting a different arrangement of blocks to break.

It's very interesting to see a new idea of how to play a game presented with something as established as *Half-Life 2.* Considering making games like this can open your eyes to the vast number of possibilities that you can do with your own creation.

### Possible pitfalls

Working on a new idea such as *BlockStorm* requires a lot of trial and error. This beta release shows that this game can end up being quite good when it's finished, but it also shows that a lot of improvement needs to be made before it's considered finished. For instance, I found the speed of the bumper pad used to bounce the ball to be too slow. The only way to discover trouble spots like these is to play the game, make adjustments, play again, make more adjustments, and repeat as needed.

I am sure that a lot of play-testing goes on behind the closed development door of this modification. However, the result should be a great game built on a unique game engine.

## Strider Mod

| | |
|---|---|
| Mod name | *Pilotable Strider Mod* |
| Version | 0.3.5.1, released 07/12/2006 |
| Author | TheQuartz |
| E-mail | `thequartz@stridermod.com` |
| Web site | `www.stridermod.com` |
| Filename | `StriderMod0351full_installer.zip` |

Here is a mod that spawned from the question, "What if I could ride in one of those striders?" Well, TheQuartz was able to make it happen. He started by creating a vehicle modification that allows the player to ride inside the strider creature. In another release of the mod, he added the ability to spawn a helicopter and fly that within the existing single player maps.

Now the *Strider Mod* has been released with two maps and a lot of added assets, it is a complete mod by itself rather than just an addition to the game. The levels start out with striders ready for mounting so that you and an opposing player can battle one-on-one. The levels are also designed for multiple teammates to join and help the cause with rockets and other weapons.

### Possible pitfalls

When dealing with models that weren't exactly meant for player interaction, many things can — and will — go wrong. A lot of testing is required to find the possible problems that come along with making such a modification.

In this case, the primary problem that I found with riding a strider is that maps generally aren't built with the strider in mind. Only large maps accommodate the striders because these creatures are large. And when making large maps, you must also make large barricades to prevent the new vehicle from breaking out of bounds. I was able to find my way into areas of the maps that I wasn't meant to find. Errors such as these need to be discovered in testing before release to the public.

## Garry's Mod

| | |
|---|---|
| Mod name | *Garry's Mod* |
| Version | 9.0.4, released 11/27/2006 |
| Author | Garry Newman |
| E-mail | `garrynewman@gmail.com` |
| Web site | `www.garrysmod.com` |
| Filename | `gmod_9_0_4.exe` |

One of the most popular modifications for *Half-Life 2* is *Garry's Mod.* This is a modification of the physics within the game giving you, the player, the ability to manipulate the world around you. It's more of a playground — or *sandbox*, as Garry likes to call it — than it is a level of game play. It's a place where the player can create and move things around that would normally be impossible.

Garry's mod is a modification of code. It started with a few changes to the weapons within the game and kept growing from there. Now, not only can you play the mod, but you can create your own mods for it. Yup, you can mod for the mod.

Garry's mod offers you the ability to create custom maps, models, and anything else you can think of to include into his mod. Then, when you load the standard modification, you are presented with options to select and load additional modifications.

Although version 10 of this modification is coming soon, version 9 is the last to offer free downloads. Valve liked the modification well enough to offer Garry the option to place it on the Steam downloader for a small fee of $10. This is testament to the fact that if you work hard and people like what you create, you will be noticed and can turn your modification hobby into a full career.

### Possible pitfalls

Making a modification and keeping it up to date means that your hobby will turn into a second job. Most, if not all, of your free time will be spent writing and rewriting code for your modification. Garry has rewritten his modification at least three times already, and I wonder whether he has time for anything else.

However, for many, the potential of turning your hobby into a full-time career is a dream come true. It's not impossible, but it's also not easy. It takes long hours and dedication. Work hard, though, and you will reap the rewards.

## Portal Challenge

| | |
|---|---|
| Mod name | *gm_portal_challenge* |
| Version | Released 08/22/2006 |
| Author | DaMaN |
| E-mail | `Da_HL_MaN@yahoo.ca` |
| Web site | `unknown` |
| Filename | `1031_gm_portal_challenge_package.zip` |

*Portal Challenge* is a modification for *Garry's Mod.* As I mention earlier, *Garry's Mod* offers you the ability to create modifications for his modification. This is just one of those mods.

With the anticipation of the *Portal,* which is to be released with *Half-Life 2: Episode 2,* some modders are deciding not to wait. Instead, they are creating their own versions of what *Portal* might be like — giving us, the gamers, a chance to experience it.

I chose to show you this version of the *Portal* game mod because it is an extension of *Garry's Mod* and because it's a great example of how you can make a mod for a mod — or perhaps mod another mod altogether.

### Possible pitfalls

Creating a mod of a game that is coming to market often means that your mod won't stay popular for long. For now, you can bask in the glory of being one of the first few who offer a taste of a game that is highly anticipated. However, that anticipation and glory will dissipate after the actual game is released. If you can offer more than what is expected from the game, you have a chance of keeping your mod alive for longer. However, this is often not the case.

REMEMBER

You modify a game that is about to come to market. However, your mod might be short lived. If you can deal with that, go for it.

# Appendix
# CD Installation Instructions

*In This Appendix:*

- System requirements
- Using the CD with Windows
- What you'll find on the CD
- Troubleshooting

## System Requirements

Make sure that your computer meets the minimum system requirements shown in the following list. If your computer doesn't match up to most of these requirements, you may have problems using the software and files on the CD. For the latest and greatest information, please refer to the ReadMe file located at the root of the CD-ROM.

- A PC with a Pentium 4 2.0GHz or Athlon XP 2000+ processor or higher
- Microsoft Windows 2000 with Service Pack 4, or Windows XP with Service Pack 1 or 2 or later
- At least 512MB of total RAM installed on your computer
- 650MB of available hard-disk space
- A CD-ROM drive
- A sound card for PCs
- 1,024x768 monitor resolution with 16-bit video card
- A modem with a speed of at least 14,400 bps

If you need more information on the basics, check out these books published by Wiley Publishing, Inc.: *PCs For Dummies,* by Dan Gookin; *Windows 2000 Professional For Dummies, Windows XP For Dummies,* all by Andy Rathbone.

## Using the CD with Microsoft Windows

To install the items from the CD to your hard drive, follow these steps.

1. **Insert the CD into your computer's CD-ROM drive. The license agreement appears.**

   **Note:** The interface won't launch if you have autorun disabled. In that case, click Start➪Run. In the dialog box that appears, type **D:\start.exe**. (Replace D with the proper letter if your CD-ROM drive uses a different letter. If you don't know the letter, see how your CD-ROM drive is listed under My Computer.) Click OK.

2. **Read through the license agreement, and then click the Accept button if you want to use the CD.**

   The CD interface appears. The interface allows you to install the programs and run the demos with just a click of a button (or two).

## What You'll Find on the CD

The following sections are arranged by category and provide a summary of the software and other goodies you'll find on the CD. If you need help with installing the items provided on the CD, refer back to the installation instructions in the preceding section.

*Shareware programs* are fully functional, free, trial versions of copyrighted programs. If you like particular programs, register with their authors for a nominal fee and receive licenses, enhanced versions, and technical support.

*Freeware programs* are free, copyrighted games, applications, and utilities. You can copy them to as many PCs as you like — for free — but they offer no technical support.

*GNU software* is governed by its own license, which is included inside the folder of the GNU software. There are no restrictions on distribution of GNU software. See the GNU license at the root of the CD for more details.

*Trial, demo,* or *evaluation* versions of software are usually limited either by time or functionality (such as not letting you save a project after you create it).

### Author-created material

For Windows.

All the examples provided in this book are located in the Author directory on the CD and work with Windows 95/98/NT/XP/2000 and later computers. These files contain much of the sample code from the book.

## Adobe Photoshop CS2

*Trial Version*

*For Windows.* A trial version of Adobe's powerful image manipulation software. For more information, visit `www.adobe.com`.

## Game mods

A portion of the mods listed in Chapter 17 can also be found on the CD-ROM. Please see the CD for further details.

## Normal Map Generator

*For Windows.* A tool for converting height maps to normal maps. For more information, visit `www.ati.com`.

## NVIDIA Photoshop plug-ins installer

*For Windows:* This filter takes one of your images and transforms it for use with your custom texture into a normal map.

## Pakrat

Pakrat is a compression viewer and utility with a GUI that makes creating your map packages easy.

## Room 101 — the video

A video presentation created by the author of this book that shows step-by-step how to do many of the things discussed in the chapters of this book.

### WinZip

*Trial Version*

*For Windows.* This handy utility allows you to view, extract, and manipulate ZIP archives on your computer. For more information, visit `www.winzip.com`.

## Troubleshooting

I tried my best to compile programs that work on most computers with the minimum system requirements. Alas, your computer may differ, and some programs may not work properly for some reason.

The two likeliest problems are that you don't have enough memory (RAM) for the programs you want to use, or you have other programs running that are affecting installation or running of a program. If you get an error message such as `Not enough memory` or `Setup cannot continue`, try one or more of the following suggestions and then try using the software again:

- **Turn off any antivirus software running on your computer.** Installation programs sometimes mimic virus activity and may make your computer incorrectly believe that it's being infected by a virus.
- **Close all running programs.** The more programs you have running, the less memory is available to other programs. Installation programs typically update files and programs; so if you keep other programs running, installation may not work properly.
- **Have your local computer store add more RAM to your computer.** This is, admittedly, a drastic and somewhat expensive step. However, adding more memory can really help the speed of your computer and allow more programs to run at the same time.

**Customer Care**

If you have trouble with the CD-ROM, please call the Wiley Product Technical Support phone number at (800) 762-2974. Outside the United States, call 1(317) 572-3994. You can also contact Wiley Product Technical Support at **http://support.wiley.com**. John Wiley & Sons will provide technical support only for installation and other general quality control items. For technical support on the applications themselves, consult the program's vendor or author.

To place additional orders or to request information about other Wiley products, please call (877) 762-2974.

# Index

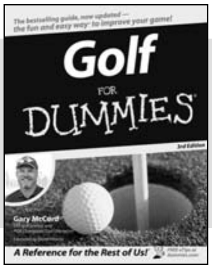
# Notes

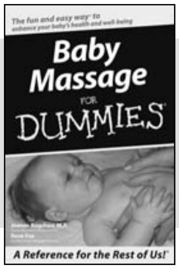
Notes

Notes

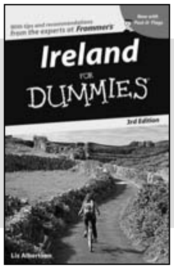
## Notes

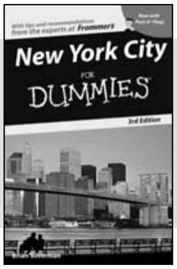
## SPORTS, FITNESS, PARENTING, RELIGION & SPIRITUALITY



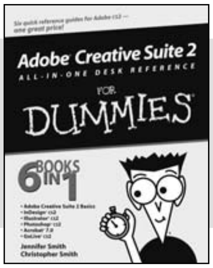


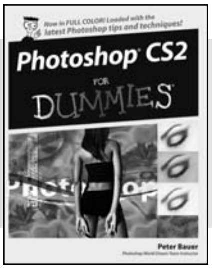
## TRAVEL



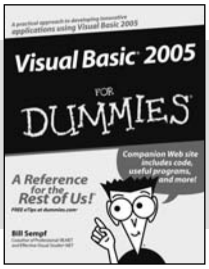


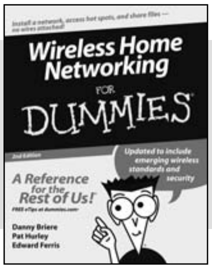
## GRAPHICS, DESIGN & WEB DEVELOPMENT

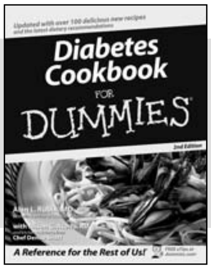

Adobe Creative Suite 2 All-in-One Desk Reference For Dummies

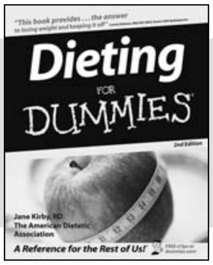


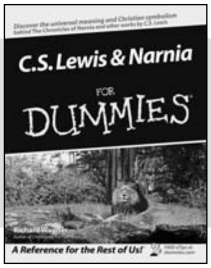
## NETWORKING, SECURITY, PROGRAMMING & DATABASES

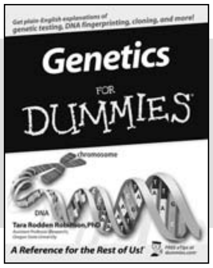





Diabetes Cookbook For Dummies
A Reference for the Rest of Us!


Dieting For Dummies
A Reference for the Rest of Us!


C.S. Lewis & Narnia For Dummies
A Reference for the Rest of Us!


Genetics For Dummies
A Reference for the Rest of Us!