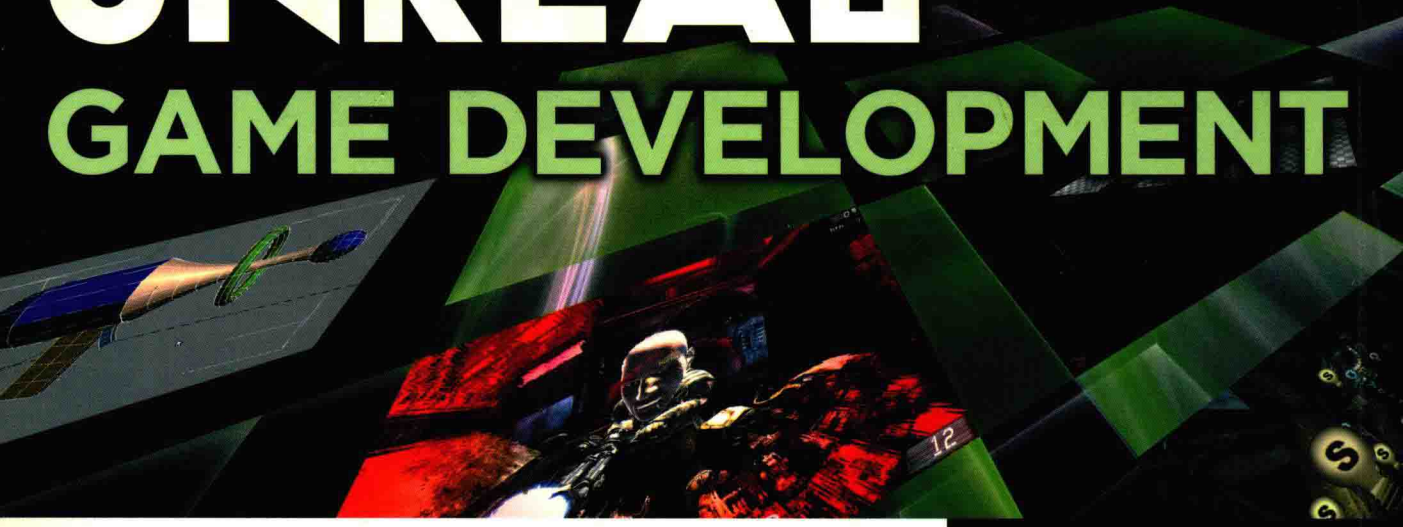

UNREAL
GAME DEVELOPMENT
Ashish Amresh
Alex Okita

UNREAL GAME DEVELOPMENT



Using Unreal Engine 3, this book teaches aspiring game makers the fundamentals of designing a computer game. The only prerequisite is a basic working knowledge of computers and a desire to build an original game. The authors provide a step-by-step tutorial for game creation, explaining how to design levels, demonstrating how to use the UnrealScript to populate and refine levels, and offer some art tools for polishing your level. By the end of the book, you will have completed a fully functional game with your own design, code, and art.

The process taught in *Unreal Game Development* is the same one used by professional game designers. So to get the most out of this book, gather up some friends and work through the book together as a team and with time limits, mimicking the key elements of real-world commercial game development.

Programming and design tools used in developing your game include:

- **a 3D authoring tool (such as Autodesk 3ds Max or Blender)**
- **a 2D image-editing tool (such as GIMP or Adobe Photoshop)**
- **a programming tool (such as WOTgreal or UltraEdit)**
- **a design tool (either the Unreal Editor of Unreal Tournament III or the Unreal Development Kit)**

Check out the book's website at **www.akpeters.com/unrealgamedev** for helpful tutorials and resources, as well as tools for instructors, such as lesson plans and class presentations.

A K Peters, Ltd.

ISBN 978-1-56881-459-9

Amresh
Okita

UNREAL GAME DEVELOP

Unreal Game Development

Ashish Amresh
Alex Okita

A K Peters, Ltd.
Natick, Massachusetts

Editorial, Sales, and Customer Service Office

A K Peters, Ltd.
5 Commonwealth Road
Natick, MA 01760
www.akpeters.com

Library of Congress Cataloging-in-Publication Data
Amresh, Ashish.
Unreal game development / Ashish Amresh, Alex Okita.
p. cm.
ISBN 978-1-56881-459-9 (alk. paper)
1. Computer games--Programming. 2. UnrealScript (Computer program language)
I. Okita, Alex. II. Title.
QA76.76.C672.A52 2010
794.8'1526--dc22
2010010527

Printed in India
14 13 12 11 10 10 9 8 7 6 5 4 3 2 1

Unreal Game Development

I dedicate this book to my parents, Latha and Amresh, who for years have patiently withstood all my whims and fancies and to my wife, Kiran, without whose unquestioning support this book would not have been possible.

—A. A.

I thank my mother, Sandi Okita, for always supporting me.

—A. O.

Acknowledgments

When Camp Game was started seven years ago at New York University (NYU), it was by and large an experiment to see what teenagers would come up with if taught state of the art video game development skills. The magic created by the students each year eventually convinced us that that the camp curriculum needed to be shared with a larger audience. This book is an effort to help educators create similar magic by introducing the process of video game development into their schools.

First and foremost we would like to thank the students and their parents for participating in the camp game programs at NYU and Arizona State University (ASU). Secondly, we would like to thank the instructors who have lectured over the years at Camp Game, namely Dov Jacobson and Tim Fielder at NYU; Ara Shirinian, Geoff Wall, Robert Srinivasiah, Clark Morrisaint, Joseph Grossmann and Arnaud Ehgner at ASU. We would like to specifically thank Ara and Geoff for helping us with proofreading and content editing. We would like to thank the sponsors of the program over the years, especially Mark Rein, Mike Capps, and Tim Sweeney at Epic Games, Steve Singer at Nintendo, Paul Skiera at Adaptive Curriculum, Mark Buchignani at Rainbow Studios, and Maureen Higgins at Autodesk. We'd also like to thank Stan Miskiewicz, Alejandro Gil and the rest of the folks who I worked with at Black Point Studios for years of great work with the Unreal Engine, and also to wish the best for Justin Miur and Martin Murphy formerly of Midway Games. Finally we would like to thank Alice Peters and her team at A K Peters for believing in us and helping us pave the way for this fantastic curriculum to be accessible to students all over the world.

Required Tools

This book uses the Unreal Engine 3 to teach game design. We cover both the Unreal Development Kit, better known as UDK, and Unreal Tournament 3, or UT3. UT3 can be downloaded via Steam (http://store.steampowered.com/). UDK can be downloaded at http://www.udk.com/. UDK is updated monthly and many changes and advancements have been added to it. UT3 hasn't been drastically updated since its release but most of the tools found in UT3 are still in use in UDK. Both UDK and UT3 are based on Unreal Engine 3 or UE3 (licensing information available at http://www.unrealtechnology.com/).

Some of the tutorials found in this book were created using UT3: at the time of the authoring of this book, UDK was not yet available. But this book can be used with either UDK or UT3; while UT3 may have features that visually differ from UDK, much of the functionality remains unchanged. As a beginning student learning to use the Unreal Engine, you will not need to know the details of the latest releases for UDK. Many of the additions to UDK simply add new tools on top of the engine's core functionality.

When you install and play most Unreal Engine–based games, the Editor is already present: the game is also the Editor! Simply adding "editor" to the command line of the shortcut to the game will launch the game in editor mode. These sorts of topics will be covered in greater detail when we begin.

In addition to Unreal Engine 3, you'll also be required to use some 3D authoring software. Unreal Engine has very primitive modeling tools but these are not meant for building characters, vehicles, or props. For these sorts of complex models we'll be using 3D Studio Max, a leading 3D-modeling software produced by Autodesk. A 30-day free trial is available at http://autodesk.com/. Alternatively, software like Blender 3D, an open source 3D modeling application available at http://www.blender.org/, can be used.

To complete the tutorials, you'll also need 2D image-editing software. For the tutorials in this book we'll introduce you to GIMP, a popular open source 2D image-editing software available for free at http://www.gimp.org/. You may already be familiar with Adobe Photoshop (see https://www.photoshop.com/), which also works well.

To complete the programming section of this book you can use any text editor. Unreal Engine 3 uses an interpreted language called Unreal Script, or UScript for short. Unreal Script is similar to Java but has a lot of custom programming conventions specifically designed for multiplayer games. Though we don't recommend using Notepad, you don't have to have anything more than that. But we recommend text editing software capable of syntax highlighting. For this we'll be using WOTgreal, a text editor with syntax highlighting specifically designed for Unreal Script and available as freeware at http://www.wotgreal.com/. UltraEdit, available for purchase at http://www.ultraedit.com/, is another available text editor.

All content, tutorials, and resources used and created in this book are available for download at http://www.akpeters.com/unrealgamedev. The site also includes tools for instructors, including lesson plans and class presentations.

	Preferred Tool	Alternate Tool
3d Art Assets	**Autodesk 3ds Max*, Maya** www.autodesk.com	**Blender** (freeware) www.blender.org
2d Art Assets	**Adobe Photoshop** www.photoshop.com	**GNU Image Manipulation Program** (freeware) www.gimp.org
Programming	**WOTgreal** (freeware) www.wotgreal.com	**UltraEdit** www.ultraedit.com
Design	**Unreal Editor of Unreal Tournament III** (included)	**Unreal Development Kit** www.udk.com

* A free 30-day trial of 3ds Max is available from Autodesk's website.

Introduction: So You Want To Make Games

Who This Book Is For

I remember one of the first times I ever laid eyes on a video game. It was 1986, and I was ten years old. My mother had foolishly allowed me to loiter around a small video arcade at a Montgomery Ward department store as she went about her shopping business. It was foolish for her, of course, because it would take some considerable effort on her part to finally pry me away.

Quarter-less yet curious, I was perfectly content to gaze into the screen across some teenager's shoulder. As I stood there, everything else in the department store just faded away. The room, the arcade cabinets, the people—none of them existed any more as my complete attention was on Mario, in *Vs. Super Mario Bros.*, traversing this strange world of pipes and creatures.

I didn't fully understand what I was seeing at the time, certainly not in terms that I understand now, but the memory of that game was burned into my head so completely that I can remember the exact screens and sequence of this strange kind of emotional drama unfolding in front of me: not drama in the sense of a soap opera or of someone losing his temper—but the emotional weight of what was actually going on in the game, exaggerated by the novelty of it all. Mario stands up on a pipe, as the bad guy below wanders back and forth. How is the player going to get past this? Mesmerized, I watched him skillfully jump down from the pipe and take care of the bad guy by a few deft dodges and jumps. The action was rudimentary, but to see a video game like this for the first time was exciting! Before I could take another breath, Mario jumped up onto another pipe, and this time there were *two* bad guys below—now *this* was a serious challenge! But the pair of Goombas, as I later learned the bad guys were called, were too much for the brave player, and my introduction to Mario had come to an end.

And so began what developed into a teenage obsession with playing games, which eventually turned into a lifetime obsession with designing games. In those early years, I had no idea how I was going to do it, or even if I was smart enough to do it, but I knew for sure that I wanted to make games. While seeking my own path toward becoming a video game designer, without anyone to guide me or show direction, on that day in

Montgomery Ward began 16 years of muddling, dabbling, dreaming, and a myriad of tiny failures and successes.

If you can relate to this story—if you love playing video games, if you crave to learn how to make them but you don't know where to start, what to do, or even what is involved in making a game—then this book is for you.

I Have No Skills (Yet), and I Must Make Games!

To get everything out of this book that we have put in, we don't expect you to have any programming ability or knowledge. You don't have to be an artist or know how to draw, and you most certainly don't have to be a game designer. This book was written for aspiring game makers who know that they want somehow to be involved in making games but aren't sure what role they want to specialize in. Even if you're uncertain whether making games is something you seriously want to pursue in the first place, this book will help you solidify your goals one way or the other.

For the most part, all you need to start out with is an interest in creating video games. However, having said that, we have made a few assumptions about your ability when it comes to using computers.

Our lessons and tutorials go over every important operation step by step, and we will spend a great amount of detail explaining how the various tools we use to create games work; but you should already be able to perform everyday tasks on any recent Windows computer without difficulty. If you check email on a regular basis and know where your files are on your computer and how to use them, you shouldn't have any issues whatsoever.

If using computers is new to you, then we recommend getting up to speed with a basic Windows user book either before or in conjunction with working on our material.

This book mirrors the curriculum we use at CampGame, a six-week summer program organized for high school students at The New York University and Arizona State University that has been running successfully for over five years. Students enter with no prior knowledge of game making whatsoever, and through the course of six intensive weeks, they finish as teams of budding game developers who have already completed fully functional games with their own designs, code, and art.

What Can You Accomplish in Six Weeks?

Video games today are a big and serious business. Commercial projects frequently cost tens of millions of dollars and employ several dozen of development staff, including designers, programmers, artists, producers, and testers at a minimum. What's more, most game projects take at least a year of full-time effort among the staff to complete, and it's not unheard of to see this development cycle consume two or even three years of time. We've come a long way since Atari 2600 of the early 1980s. Projects in those

days usually employed a staff of just one programmer. Incidentally, this programmer also happened to be the game designer, the artist, the sound designer and engineer all rolled into one.

All of this may sound quite intimidating. After all, if it took a year for a skilled jack-of-all-trades to complete a crude Atari video game, what can you hope to possibly accomplish in just six weeks, starting from scratch? For our program, the key is in the various widely-available tools we employ (more on this in the next section) and in a tight focus on the essentials of the game development process. Let's face it: No matter how smart and talented you are, it's simply not possible to make the next *Legend of Zelda* game in this span of time. However, you will have the tools and ability to create something of that scale afterward if you so desire.

Another way we make efficient use of our program, at least in the CampGame curriculum, is that we divide the course into three major tracks. In the first few weeks, we teach every student the basics of each track: Design, Programming, and Art. From that point forward, we ask each student to choose one of these specializations and spend the majority of the remainder of the program focusing their work on that track. Another essential part of the CampGame program is that we eventually divide the class into teams of four to six students, each having at least one designer, programmer, and artist. The teams spend the second half of CampGame planning and implementing their own game projects. Of course, in such a compressed space of time, our students also learn quite a bit about planning and prioritizing. The stark reality of game development is that you never have enough time to make everything that you want. The commercial nature of game making means that you have only a limited time in which to finish your projects, and so decisions about what *not to* work on are usually just as important as those about what *to* work on.

You can approach this book in one of two ways: as an individual study, or as part of a team of at least three people—a designer, a programmer, and an artist. If possible, we recommend that you gather up some of your friends who are just as interested in game making as you are and work through this book in the CampGame way: impose time limits on yourself as indicated in each chapter, pick your specializations, and use the lessons as tools for completing your own project in six weeks. Although going solo at your own leisurely pace is certainly a viable approach and will teach you a tremendous amount, working through this book with a team and time limits will also give you experience in team dynamics and prioritizing—elements that are absolutely crucial in the real world of commercial game development.

Technologies and Tools We Use

Tools are the key to efficient game development. Over the years, game developers realized that even if they made several rather different games, they frequently had to do the same work over and over with the creation of each game. Back in the old days,

there was a lot of this kind of repetition. Today, however, there are all kinds of tools (which is just another word for a computer application) available that automate the boring, repetitive stuff in game development.

If making a game is like building a house, the tools we use allow us to design the shape, looks, and composition of each room just by saying what we want and where we want each piece to go. The electrical wiring behind the walls, the plumbing, the insulation, the foundation of the house, the air conditioning system—all the required bits that nobody notices when they walk into your home—all that stuff is automatically taken care of by our tools. In this way, we can spend all our time actually making an interesting, beautiful house.

	Preferred Tool	Alternate Tool
3d Art Assets	**Autodesk 3ds Max*, Maya** www.autodesk.com	**Blender** (freeware) www.blender.org
2d Art Assets	**Adobe Photoshop** www.photoshop.com	**GNU Image Manipulation Program** (freeware) www.gimp.org
Programming	**WOTgreal** (freeware) www.wotgreal.com	**UltraEdit** www.ultraedit.com
Design	**Unreal Editor of Unreal Tournament III** (included)	**Unreal Development Kit** www.udk.com

* A free 30-day trial of 3ds Max is available from Autodesk's website.

Each specialty has its own tools, which are detailed in the table above. While we will focus on one particular, preferred tool for each category of work that needs to be done in our development cycle, we'll also list alternate free tools that provide the same or very similar functionality. You don't have to purchase any additional software to complete all the work in this book, but having access to the preferred tools will give you an advantage, as they are also industry standards.

Some tools are used to create (or *export*) original assets, or files, while others take those files, or *import* them, and allow you to make compositions that make up our game. The way all our tools work together is called our *work flow* or *tool chain* or *pipeline*.

In our pipeline, we use 3ds Max to create our three-dimensional (3D) models and characters and GIMP to create *textures,* or two-dimensional (2D) images that will be applied to the surfaces of those 3D models, as well as any art in your game that is 2D. We'll use WOTgreal to help us write our program code. Although you could use any text editor to accomplish this, WOTgreal has some nice features to make the whole process easier. Finally, we'll use Unreal Editor, or UnrealEd, in several important ways. With UnrealEd, you can import all the 3D and 2D assets and code

we created with the other programs and place them into your game's levels. UnrealEd's main role is that of a level editor, but it's really much more than that. It's where all of the pieces come together to form your complete product, and it's even used for creating certain art elements that are not (strictly speaking) game design-related, such as particle systems, materials, and animated matinee sequences. But we're getting ahead of ourselves here.

The Game Development Cycle

Your exposure to the game development cycle in this book will be just a hint of what usually happens in game development studios the world over. Nevertheless, it's good to have even a basic understanding of what happens when a studio tries to make a game. We'll present to you a simplified look here, as anything more is beyond the scope of this book, but if you are interested in the details of project planning and production there are all kinds of volumes out there to satisfy your curiosities.

Game development is a fascinating process. Most kinds of commercial projects or products are a bit like building a bridge—it's been done thousands of times before, so there is usually a practiced and standard way of doing things. Your bridge might have a fancy design or some fancy paint; but for the most part, all of the variables are known before you make the bridge, and so you can fairly easily say that such a bridge will take so much time to build, and you'll rarely be too far off from the reality.

The reality when it comes to making games, however, is much different. Even for experienced developers, making a game, in many cases, is a bit like trying to figure out how to make a bridge when you've never had to build a bridge before. What's more, you'll have to build your bridge while trying to figure out how to do it at the same time. But that's not all. As you're building, the pieces you're using might change shape from one day to the next. You might even have to go back and break apart bridge sections you thought were already finished. At this rate it won't be very long before you stand back and ask yourself, "What kind of crazy bridge project is this, anyway?"

There are simply too many unknowns in game making to treat it like making a bridge—at least at the start of a project. Traditionally, game projects are divided into three main phases: preproduction, production, and testing.

Essentially, preproduction is when you decide how you're going to make your game. Production is when you actually build it, and testing is when you make sure that it works the way you intended. You finish one step, and then you move onto the next. Simple, right?

The model is simple, but unfortunately it's not good for making games, because it assumes that once you're done with one phase, that part's all done, and you move on and never look back, kind of like building a bridge. If you make games in this way, you might work on your game for a long time before you realize that your original techniques are now causing you a lot of problems, and you should really have done it

a different way. So now you are stuck with the dilemma of tearing down part of your game and re-doing it, or sticking with these inefficient, unwieldy techniques. It's difficult to appreciate exactly what this feels like before you've actually experienced it firsthand, but it's something that every game developer goes through.

So what now? Well, the ugly truth in game making is that pretty much all phases of the project are happing all the time during the project. You might do most of your design (as part of preproduction) in the beginning of the project, but really you will be designing all the way up until the end. You might do most of your testing at the end of a project, but really you will be testing your game all the time, even just after the first few lines of code you write.

This approach, where you are doing all parts of the process at every step over and over, is called an *iterative* process; and it very accurately captures good game-making habits. In an iterative process, you test each bit of progress you make to make sure it works. Then you stand back, look at it, and re-evaluate your plan. Maybe you decided in the beginning that your character in a game will have five lives. Now, after you have finished a couple of levels, you find that five lives won't work so well—you decide that your character needs to have a life bar instead. In this way, you stay flexible, and your current design or plan is nothing more than that: it's what you plan to do currently. You can change the plan as your product develops.

The key for you, the aspiring game maker, is not to get too stuck on one way of doing things. Be open to changing your plan, and you may find your project much better for it. However, this isn't an excuse to make sweeping changes or to do something completely different every week—you'll never finish a game that way. You'll have to decide whether your game character has lives or health bars or some other system as quickly as is practical. But with an iterative approach, you make and settle on this decision once you have something working, instead of just thinking in your head what might be good. In the end, each decision about what you want to change has a cost associated with it. If you think the change is worth the cost, go for it!

Project Roles

Whether you are following this book as part of a team or just by yourself, eventually you will want to pick an area of specialization. More than ever, game development studios primarily look for specialists to join their teams. Although it's absolutely valuable to have knowledge in each area of expertise, complete generalists will usually have a difficult time getting hired. In this book, we distinguish between three specialties: designer, programmer, and artist.

Each of these roles can be broken down into even more precise specialties. If you want to pursue a career after learning the material in this book, it would be wise to select a specific area of expertise that best fits your talents and focus on developing that particular skill. For example, just in the category of artist, there are concept