

THOMSON
COURSE TECHNOLOGY

Professional • Trade • Reference

GAME PROGRAMMING ALL IN ONE

2ND EDITION

JONATHAN S. HARBOUR



Premier

Press

THOMSON
COURSE TECHNOLOGY™

Professional ■ Trade ■ Reference



GAME PROGRAMMING ALL IN ONE

2ND EDITION

JONATHAN S. HARBOUR



Premier

Press

This page intentionally left blank



GAME PROGRAMMING ALL IN ONE, 2ND EDITION



JONATHAN S. HARBOUR

THOMSON
—★—
COURSE TECHNOLOGY
Professional ■ Trade ■ Reference

© 2004 by Thomson Course Technology PTR. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system without written permission from Thomson Course Technology PTR, except for the inclusion of brief quotations in a review.

The Premier Press and Thomson Course Technology PTR logo and related trade dress are trademarks of Thomson Course Technology PTR and may not be used without written permission.

Microsoft, Windows, DirectDraw, DirectMusic, DirectPlay, DirectSound, DirectX, and Xbox are either registered trademarks or trademarks of Microsoft Corporation in the U.S. and/or other countries. Apple, Mac, and Mac OS are trademarks or registered trademarks of Apple Computer, Inc. in the U.S. and other countries. All other trademarks are the property of their respective owners.

Important: Thomson Course Technology PTR cannot provide software support. Please contact the appropriate software manufacturer's technical support line or Web site for assistance.

Thomson Course Technology PTR and the author have attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

Information contained in this book has been obtained by Thomson Course Technology PTR from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Thomson Course Technology PTR, or others, the Publisher does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from use of such information. Readers should be particularly aware of the fact that the Internet is an ever-changing entity. Some facts may have changed since this book went to press.

Educational facilities, companies, and organizations interested in multiple copies or licensing of this book should contact the publisher for quantity discount information. Training manuals, CD-ROMs, and portions of this book are also available individually or can be tailored for specific needs.

ISBN: 1-59200-383-4

Library of Congress Catalog Card Number: 2004091915
Printed in the United States of America

04 05 06 07 08 BH 10 9 8 7 6 5 4 3 2 1

THOMSON
—★—™
COURSE TECHNOLOGY
Professional ■ Trade ■ Reference

Course PTR, a division of Course Technology
25 Thomson Place
Boston, MA 02210
<http://www.courseptr.com>

**SVP, Thomson Course
Technology PTR:**
Andy Shafran

Publisher:
Stacy L. Hiquet

Senior Marketing Manager:
Sarah O'Donnell

Marketing Manager:
Heather Hurley

Manager of Editorial Services:
Heather Talbot

Acquisitions Editor:
Mitzi Koontz

Senior Editor:
Mark Garvey

Associate Marketing Managers:
Kristin Eisenzopf and Sarah Dubois

Project Editor/Copy Editor:
Cathleen D. Snyder

Technical Reviewer:
Joshua Smith

**Thomson Course Technology
PTR Market Coordinator:**
Amanda Weaver

Interior Layout Tech:
Shawn Morningstar

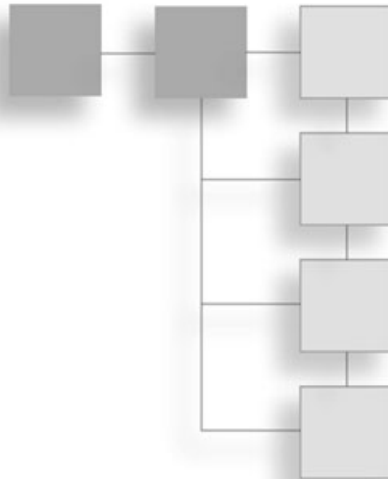
Cover Designer:
Steve Deschene

CD-ROM Producer:
Brandon Penticuff

Indexer:
Kelly Talbot

Proofreader:
Sean Medlock

For Jeremiah



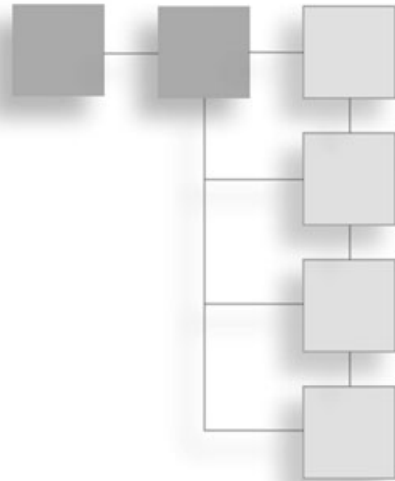
ACKNOWLEDGMENTS

A book of this size involves a lot of work even after the writing is done. It takes a while just to read through a programming book once, so you can imagine how difficult it is to read through it several times, making changes and notes along the way, refining, correcting, and preparing the book for print. I am indebted to the hard work of the editors, artists, and layout specialists at Premier Press who do such a fine job. Thank you Mitzi Koontz, Emi Smith, and Stacy Hiquet for your encouragement and support.

I owe many thanks to Cathleen Snyder, one of the most amazing editors in the business, who both managed the project and copy edited the manuscript, and to Joshua R. Smith, who offered his technical expertise and long experience in the game industry to point out my mistakes and to offer advice. I believe you will find this a true gem of a game programming book due to their efforts.

I would also like to thank Bruno Miguel Teixeira de Sousa for writing the first edition of this book. Some of his original work may still be found in this new edition, in Chapters 6, 18, 19, and 20.

ABOUT THE AUTHOR



JONATHAN S. HARBOUR has been an avid gamer and programmer for 17 years, having started with a TI-99, a Commodore PET, and a Tandy 1000. In 1994, he earned a bachelor of science degree in computer information systems. He has since earned the position of senior programmer with seven years of professional programming experience. Jonathan is a member of the *Starflight III* team, working with the original designers and other volunteers on a sequel to the first two *Starflight* games (using Allegro), originally published by Electronic Arts in 1985 and 1989, respectively. Jonathan has released two retail Pocket PC games, *Pocket Trivia* and *Perfect Match*, and has authored or coauthored five other books on the subject of game programming, including *Pocket PC Game Programming*, *Visual Basic Game Programming with DirectX*, *Visual Basic .NET Programming for the Absolute Beginner*, *Beginner's Guide to DarkBASIC Game Programming*, and *Beginning Game Boy Advance Programming*. He maintains a Web site dedicated to game programming and other topics at <http://www.jharbour.com>. Jonathan lives in Arizona with his wife, Jennifer, and children, Jeremiah and Kayleigh.



CONTENTS AT A GLANCE

Introductionxxv

PART I: INTRODUCTION TO CROSS-PLATFORM GAME PROGRAMMING **1**

CHAPTER 1 Demystifying Game Development3

CHAPTER 2 Getting Started with Dev-C++ and Allegro33

CHAPTER 3 Basic 2D Graphics Programming with Allegro71

CHAPTER 4 Writing Your First Allegro Game119

CHAPTER 5 Programming the Keyboard, Mouse, and Joystick145

PART II: 2D GAME THEORY, DESIGN, AND PROGRAMMING **185**

CHAPTER 6 Introduction to Game Design187

CHAPTER 7 Basic Bitmap Handling and Blitting215

CHAPTER 8 Basic Sprite Programming: Drawing Scaled, Flipped, Rotated, Pivoted, and Translucent Sprites237

CHAPTER 9 Advanced Sprite Programming: Animation, Compiled Sprites, and Collision Detection279

CHAPTER 10	Programming Tile-Based Backgrounds with Scrolling	339
CHAPTER 11	Timers, Interrupt Handlers, and Multi-Threading	381
CHAPTER 12	Creating a Game World: Editing Tiles and Levels	429
CHAPTER 13	Vertical Scrolling Arcade Games	455
CHAPTER 14	Horizontal Scrolling Platform Games	489

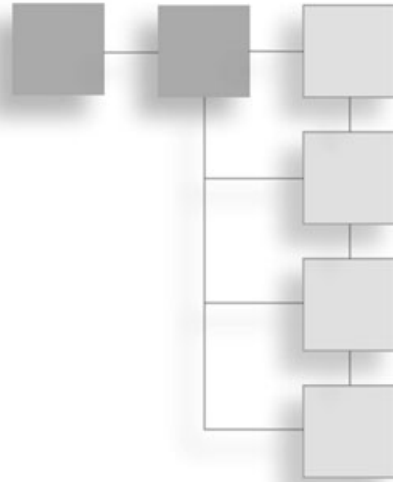
PART III: TAKING IT TO THE NEXT LEVEL **509**

CHAPTER 15	Mastering the Audible Realm: Allegro's Sound Support	511
CHAPTER 16	Using Datafiles to Store Game Resources	539
CHAPTER 17	Playing FLIC Movies	551
CHAPTER 18	Introduction to Artificial Intelligence	563
CHAPTER 19	The Mathematical Side of Games	585
CHAPTER 20	Publishing Your Game	611

PART IV: APPENDIXES **631**

APPENDIX A	Chapter Quiz Answers	633
APPENDIX B	Useful Tables	651
APPENDIX C	Numbering Systems: Binary and Hexadecimal	657
APPENDIX D	Recommended Books and Web Sites	663
APPENDIX E	Configuring Allegro for Microsoft Visual C++ and Other Compilers	671
APPENDIX F	Compiling the Allegro Source Code	685
APPENDIX G	Using the CD-ROM	691
	Index	693

CONTENTS



Introductionxxv

PART I: INTRODUCTION TO CROSS-PLATFORM GAME PROGRAMMING **1**

Chapter 1 Demystifying Game Development3

Introduction4

 Practical Game Programming5

 Goals Revisited.....6

The High-Level View of Game Development6

Recognizing Your Personal Motivations9

 Decision Point: College versus Job10

 Every Situation Is Unique10

 A Note about Specialization.....12

 Game Industry Speculation.....13

 Emphasizing 2D.....14

 Finding Your Niche15

Getting into the Spirit of Gaming18

 Starship Battles: An Inspired Fan Game.....18

 Axis & Allies: Hobby Wargaming22

 Setting Realistic Expectations for Yourself24

An Introduction to Dev-C++ and Allegro25

 DirectX Is Just Another Game Library25

	Introducing the Allegro Game Library.....	26
	Supporting Multiple C/C++ Compilers.....	29
	Summary	30
	Chapter Quiz	31
Chapter 2	Getting Started with Dev-C++ and Allegro	33
	Introduction	34
	Installing and Configuring Dev-C++ and Allegro	35
	Installing Dev-C++	36
	Updating Dev-C++	37
	Installing Allegro	41
	Taking Dev-C++ and Allegro for a Spin	43
	Testing Dev-C++: The Greetings Program	44
	Testing Allegro: The GetInfo Program.....	53
	Gaining More Experience with Allegro	63
	The Hello World Demo	63
	Allegro Sample Programs	65
	Summary	68
	Chapter Quiz	68
Chapter 3	Basic 2D Graphics Programming with Allegro	71
	Introduction	72
	Graphics Fundamentals	74
	The InitGraphics Program	75
	The DrawBitmap Program	79
	Drawing Graphics Primitives	82
	Drawing Pixels	82
	Drawing Lines and Rectangles	84
	Drawing Circles and Ellipses	95
	Drawing Splines, Triangles, and Polygons	103
	Filling in Regions	109
	Printing Text on the Screen	112
	Constant Text Output	112
	Variable Text Output	113
	Testing Text Output.....	114
	Summary	115
	Chapter Quiz	116

Chapter 4	Writing Your First Allegro Game	119
	Tank War	119
	Creating the Tanks	120
	Firing Weapons	122
	Tank Movement	125
	Collision Detection	126
	The Complete Tank War Source Code	126
	Summary	141
	Chapter Quiz	142
Chapter 5	Programming the Keyboard, Mouse, and Joystick	145
	Handling Keyboard Input	146
	The Keyboard Handler	146
	Detecting Key Presses	147
	The Stargate Program	148
	Buffered Keyboard Input	152
	Simulating Key Presses	153
	The KeyTest Program	154
	Handling Mouse Input	155
	The Mouse Handler	156
	Reading the Mouse Position	156
	Detecting Mouse Buttons	157
	Showing and Hiding the Mouse Pointer	157
	The Strategic Defense Game	158
	Setting the Mouse Position	165
	Limiting Mouse Movement and Speed	167
	Relative Mouse Motion	167
	Using a Mouse Wheel	167
	Handling Joystick Input	170
	The Joystick Handler	170
	Detecting Controller Stick Movement	171
	Detecting Controller Buttons	174
	Testing the Joystick Routines	175
	Summary	182
	Chapter Quiz	182

PART II: 2D GAME THEORY, DESIGN, AND PROGRAMMING

185

Chapter 6	Introduction to Game Design	187
	Game Design Basics	188
	Inspiration	188
	Game Feasibility	188
	Feature Glut	189
	Back Up Your Work	189
	Game Genres	190
	Game Development Phases	195
	Initial Design	196
	Game Engine	196
	Alpha Prototype	196
	Game Development	197
	Quality Control	197
	Beta Testing	198
	Post-Production	198
	Official Release	199
	Out the Door or Out the Window?	199
	Managing the Game	199
	A Note about Quality	200
	Empowering the Engine	200
	Quality versus Trends	201
	Innovation versus Inspiration	202
	The Infamous Game Patch	202
	Expanding the Game	203
	Future-Proof Design	203
	Game Libraries	204
	Game Engines and SDKs	204
	What Is Game Design?	204
	The Dreaded Design Document	205
	The Importance of Good Game Design	206
	The Two Types of Designs	206
	Mini Design	206
	Complete Design	207
	A Sample Design Document Template	207
	General Overview	208
	Target System and Requirements	208

Story	208
Theme: Graphics and Sound	208
Menus	208
Playing a Game	208
Characters and NPCs Description	208
Artificial Intelligence Overview	208
Conclusion	209
A Sample Game Design: Space Invaders	209
General Overview	209
Target System and Requirements	209
Story	209
Theme: Graphics and Sound	210
Menus	210
Playing a Game	210
Character and NPC Description	211
Artificial Intelligence Overview	211
Conclusion	211
Game Design Mini-FAQ	212
Summary	212
Chapter Quiz	212

Chapter 7 Basic Bitmap Handling and Blitting215

Introduction	215
Dealing with Bitmaps	217
Creating Bitmaps	219
Cleaning House	221
Bitmap Information	221
Acquiring and Releasing Bitmaps	223
Bitmap Clipping	224
Loading Bitmaps from Disk	224
Blitting Functions	227
Standard Blitting	227
Scaled Blitting	228
Masked Blitting	229
Masked Scaled Blitting	229
Enhancing Tank War—From Graphics Primitives to Bitmaps	229
Summary	234
Chapter Quiz	234

Chapter 8	Basic Sprite Programming: Drawing Scaled, Flipped, Rotated, Pivoted, and Translucent Sprites . . .	237
	Basic Sprite Handling	238
	Drawing Regular Sprites	238
	Drawing Scaled Sprites	242
	Drawing Flipped Sprites	244
	Drawing Rotated Sprites	245
	Drawing Pivoted Sprites	252
	Drawing Translucent Sprites	256
	Enhancing Tank War	259
	What's New?	260
	Modifying the Source Code	262
	Summary	276
	Chapter Quiz	276
Chapter 9	Advanced Sprite Programming: Animation, Compiled Sprites, and Collision Detection	279
	Animated Sprites	280
	Drawing an Animated Sprite	280
	Creating a Sprite Handler	283
	The SpriteHandler Program	286
	Grabbing Sprite Frames from an Image	291
	The SpriteGrabber Program	293
	The Next Step: Multiple Animated Sprites	298
	The MultipleSprites Program	300
	Run-Length Encoded Sprites	306
	Creating and Destroying RLE Sprites	307
	Drawing RLE Sprites	307
	The RLESprites Program	307
	Compiled Sprites	313
	Using Compiled Sprites	314
	Testing Compiled Sprites	315
	Collision Detection	317
	The CollisionTest Program	319
	Enhancing Tank War	324
	Summary	336
	Chapter Quiz	337

Chapter 10	Programming Tile-Based Backgrounds with Scrolling . . .	339
	Introduction to Scrolling	340
	A Limited View of the World	341
	Introduction to Tile-Based Backgrounds	345
	Backgrounds and Scenery	346
	Creating Backgrounds from Tiles	347
	Tile-Based Scrolling	347
	Creating a Tile Map	351
	Enhancing Tank War	355
	Exploring the All-New Tank War	356
	The New Tank War Source Code	359
	Summary	378
	Chapter Quiz	378
Chapter 11	Timers, Interrupt Handlers, and Multi-Threading	381
	Timers	381
	Installing and Removing the Timer	381
	Slowing Down the Program	382
	The TimerTest Program	383
	Interrupt Handlers	392
	Creating an Interrupt Handler	392
	Removing an Interrupt Handler	393
	The InterruptTest Program	393
	Using Timed Game Loops	395
	Slowing Down the Gameplay...Not the Game	395
	The TimedLoop Program	396
	Multi-Threading	397
	Abstracting the Parallel Processing Problem	398
	The Pthreads-Win32 Library	399
	Programming with Posix Threads	400
	The MultiThread Program	403
	Enhancing Tank War	413
	Description of New Improvements	414
	Modifying the Tank War Project	415
	Future Changes to Tank War	426
	Summary	426
	Chapter Quiz	426

Chapter 12	Creating a Game World: Editing Tiles and Levels	429
	Creating the Game World	429
	Installing Mappy	430
	Creating a New Map	430
	Importing the Source Tiles	432
	Saving the Map File as FMP	433
	Saving the Map File as Text	435
	Loading and Drawing Mappy Level Files	436
	Using a Text Array Map	437
	Using a Mappy Level File	442
	Enhancing Tank War	445
	Proposed Changes to Tank War	446
	Modifying Tank War	447
	Summary	453
	Chapter Quiz	453
Chapter 13	Vertical Scrolling Arcade Games	455
	Building a Vertical Scroller Engine	455
	Creating Levels Using Mappy	457
	Filling in the Tiles	459
	Let's Scroll It	460
	Writing a Vertical Scrolling Shooter	464
	Describing the Game	464
	The Game's Artwork	466
	Writing the Source Code	468
	Summary	487
	Chapter Quiz	487
Chapter 14	Horizontal Scrolling Platform Games	489
	Understanding Horizontal Scrolling Games	490
	Developing a Platform Scroller	490
	Creating Horizontal Platform Levels with Mappy	491
	Separating the Foreground Tiles	495
	Performing a Range Block Edit	497
	Developing a Scrolling Platform Game	498
	Describing the Game	498
	The Game Artwork	499
	Using the Platform Scroller	500
	Writing the Source Code	501

Summary506
 Chapter Quiz507

PART III: TAKING IT TO THE NEXT LEVEL 509

Chapter 15 Mastering the Audible Realm: Allegro’s Sound Support .511

- The PlayWave Program512
- Sound Initialization Routines514
 - Detecting the Digital Sound Driver 515
 - Reserving Voices 515
 - Setting an Individual Voice Volume..... 515
 - Initializing the Sound Driver..... 516
 - Removing the Sound Driver 516
 - Changing the Volume 516
- Standard Sample Playback Routines517
 - Loading a Sample File..... 517
 - Loading a WAV File..... 517
 - Loading a VOC File 517
 - Playing a Sample 517
 - Altering a Sample’s Properties 518
 - Stopping a Sample..... 518
 - Creating a New Sample..... 518
 - Destroying a Sample 518
- Low-Level Sample Playback Routines518
 - Allocating a Voice 519
 - Removing a Voice 519
 - Reallocating a Voice 519
 - Releasing a Voice..... 519
 - Activating a Voice 519
 - Stopping a Voice 519
 - Setting Voice Priority..... 520
 - Checking the Status of a Voice..... 520
 - Returning the Position of a Voice 520
 - Setting the Position of a Voice..... 520
 - Altering the Playback Mode of a Voice..... 520
 - Returning the Volume of a Voice..... 521
 - Setting the Volume of a Voice 521
 - Ramping the Volume of a Voice..... 521
 - Stopping a Volume Ramp..... 521

	Returning the Pitch of a Voice	521
	Setting the Pitch of a Voice	521
	Performing a Frequency Sweep of a Voice	521
	Stopping a Frequency Sweep	522
	Returning the Pan Value of a Voice	522
	Setting the Pan Value of a Voice	522
	Performing a Sweeping Pan on a Voice	522
	Stopping a Sweeping Pan	522
	The SampleMixer Program	522
	Enhancing Tank War	525
	Modifying the Game	525
	Final Comments about Tank War	536
	Summary	537
	Chapter Quiz	537
Chapter 16	Using Datafiles to Store Game Resources	539
	Understanding Allegro Datafiles	540
	Creating Allegro Datafiles	541
	Using Allegro Datafiles	544
	Loading a Datafile	544
	Unloading a Datafile	545
	Loading a Datafile Object	545
	Unloading a Datafile Object	545
	Finding a Datafile Object	545
	Testing Allegro Datafiles	545
	Summary	547
	Chapter Quiz	548
Chapter 17	Playing FLIC Movies	551
	Playing FLI Animation Files	551
	The FLI Callback Function	552
	The PlayFlick Program	552
	Playing an FLI from a Memory Block	554
	Loading FLIs into Memory	554
	Opening and Closing FLI Files	555
	Processing Each Frame of the Animation	555
	The LoadFlick Program	556
	The ResizeFlick Program	558
	Summary	561
	Chapter Quiz	561

Chapter 18	Introduction to Artificial Intelligence	563
	The Fields of Artificial Intelligence	564
	Expert Systems	564
	Fuzzy Logic	565
	Genetic Algorithms	567
	Neural Networks	569
	Deterministic Algorithms	570
	Random Motion	571
	Tracking	572
	Patterns	573
	Finite State Machines	575
	Fuzzy Logic	577
	Fuzzy Logic Basics	577
	Fuzzy Matrices	579
	A Simple Method for Memory	580
	Artificial Intelligence and Games	581
	Summary	581
	Chapter Quiz	582
Chapter 19	The Mathematical Side of Games	585
	Trigonometry	586
	Visual Representation and Laws	586
	Angle Relations	589
	Vectors	590
	Addition and Subtraction	591
	Scalar Multiplication and Division	593
	Length	594
	Normalization	594
	Perpendicular Operation	595
	Dot Product	596
	Perp-Dot Product	597
	Matrices	598
	Addition and Subtraction	598
	Scalars with Multiplication and Division	598
	Special Matrices	599
	Transposed Matrices	600
	Matrix Concatenation	601
	Vector Transformation	602

Probability	603
Sets	603
Union	603
Intersection	604
Functions	605
Integration	606
Differentiation	607
Summary	608
Chapter Quiz	608
Chapter 20 Publishing Your Game	611
Is Your Game Worth Publishing?	611
Whose Door to Knock On	612
Learn to Knock Correctly	613
No Publisher, So Now What?	613
Contracts	614
Non-Disclosure Agreement	614
The Actual Publishing Contract	614
Milestones	615
Bug Report	615
Release Day	615
Interviews	616
Paul Urbanus: Urbonix, Inc.	616
Niels Bauer: Niels Bauer Software Design	622
André LaMothe: Xtreme Games LLC	624
Summary	625
References	626
Chapter Quiz	626
Epilogue	629

PART IV: APPENDICES 631

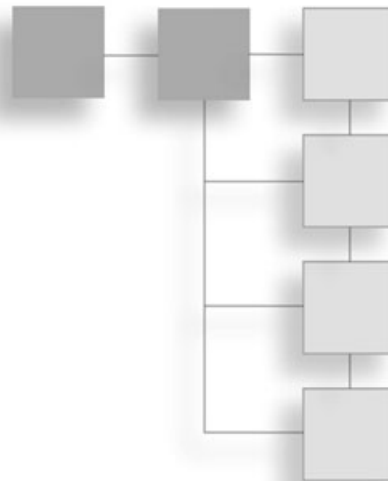
Appendix A Chapter Quiz Answers	633
Chapter 1	633
Chapter 2	634
Chapter 3	635
Chapter 4	635

Chapter 5	.636
Chapter 6	.637
Chapter 7	.638
Chapter 8	.639
Chapter 9	.639
Chapter 10	.640
Chapter 11	.641
Chapter 12	.642
Chapter 13	.643
Chapter 14	.643
Chapter 15	.644
Chapter 16	.645
Chapter 17	.646
Chapter 18	.647
Chapter 19	.648
Chapter 20	.648
Appendix B Useful Tables	.651
Integral Equations Table	.651
Derivative Equations Table	.652
Inertia Equations Table	.652
ASCII Table	.653
Appendix C Numbering Systems: Binary and Hexadecimal	.657
Binary	.657
Decimal	.659
Hexadecimal	.659
Appendix D Recommended Books and Web Sites	.663
All in One Support on the Web	.663
Game Development Web Sites	.663
Publishing, Game Reviews, and Download Sites	.664
Engines	.664
Independent Game Developers	.664
Industry	.665
Computer Humor	.665
Recommended Books	.665

Appendix E	Configuring Allegro for Microsoft Visual C++ and Other Compilers671
	Microsoft Visual C++672
	Dev-C++673
	KDevelop for Linux679
	Final Comments683
Appendix F	Compiling the Allegro Source Code685
	Microsoft Visual C++685
	Borland C++/C++Builder687
	Dev-C++688
	KDevelop for Linux689
Appendix G	Using the CD-ROM691
	Index693

This page intentionally left blank

INTRODUCTION



Greetings! This book is the second edition to the best-selling *Game Programming All in One* by Bruno Miguel Teixeira de Sousa, to whom I am indebted for the original work. This new second edition is a complete rewrite of *Game Programming All in One*, with a completely new direction, new goals, new assumptions, and new development tools. *All in One 2E*, as I have come to call it, has done away with the C++ tutorials, Windows programming tutorials, and DirectX tutorials. In fact, this book does not cover Windows or DirectX at all. Instead, this book focuses on the subject of game programming using a cross-platform game library called Allegro. This library is extremely powerful and versatile. Allegro opens up a world of possibilities that are ignored when you focus specifically on Windows and DirectX. A full quarter of the first edition was devoted to a C++ language primer, while another fourth of the book focused on Windows and DirectX basics. I decided that for this second edition, we did not need to cover those subjects again; thus, this book uses the standard C language, and the sample programs will compile on multiple platforms.

The Windows version of Allegro uses DirectX, as a matter of fact, but it is completely abstracted and transparent, hidden inside the internals of the Allegro game library. Instead, you are provided with a basic C program that includes the Allegro library and is capable of running in full-screen DirectDraw mode using any supported resolution and color depth. Additionally, Allegro provides a uniform interface for sound effects, music, and device input, which are implemented on the Windows platform with DirectSound, DirectMusic, and DirectInput. Specifically, Allegro supports DirectX 8. Imagine writing a high-speed arcade game using DirectX, and then being able to recompile that program (without changing a single line of code) under Linux, Mac OS X, Solaris, FreeBSD, IRIX, and other operating systems! Allegro is a cross-platform game library that will double or triple the user base for the games you develop with the help of this book, and at no loss in performance.

Cross-Platform Game Programming

This book will teach you to write complete games that will run on almost any operating system. Specifically, I focus on three compilers—Visual C++, Dev-C++, and KDevelop—and the sample programs will be written using both Windows and Linux, with screenshots taken from both operating systems. In all likelihood, you will have the opportunity to use your favorite development tool because Allegro supports several C compilers, including Borland C++, Borland C++Builder, Apple Development Tools 2002, and several other compilers on various platforms, including the ubiquitous GNU C++ (GCC).

The target audience for this book is beginning to intermediate programmers who already have some experience with C or C++. Also, those who want to learn to write games using C or C++ can use this book as an entry-level guide. The material is not for someone new to programming—just someone new to *game programming*. I must assume you have already learned C or C++ because there is too much to cover in the game libraries, interfaces, and so on to focus on the basic syntax of the actual language. It was difficult enough to support three different compilers and integrated development environments without also explaining every line of code. Intermediate-level programming experience is assumed, while extreme beginners (newbies) will definitely struggle.

In Appendix D, “Recommended Books and Web Sites,” I recommend introductory books for those readers. I encourage you to keep a C primer handy while reading through *All in One 2E* because this book moves along at a rapid pace. My goal was not to cover a *lot* of information, but to quickly get into the *important* information you’ll need to write good games. This book is not extremely advanced—the source code is straightforward, with no difficult libraries to learn per se, but I do not explain every detail. I do cover most of the function library in Allegro, since that is the focus of this book, but I do not explain any standard C functions. The goal is to get up and running as quickly as possible with some game code! In fact, you will be writing your first graphics programs in Chapter 3 and your first game in Chapter 4. You will, however, quickly ramp up to advanced topics, such as creating game levels and scrolling the game world on the screen, with sample code, such as the *PlatformScroller* program (see Chapter 14).

Yes, it is true, this book focuses entirely and exclusively on 2D games. This is a huge genre that includes many real-time and turn-based strategy games, such as *Civilization III*, the *Age of Empires* series, *Diablo*, *Starcraft*, and so on. If you scoff at 2D games, then I encourage you to pick up *3D Game Programming All in One* (Premier Press, 2004) instead of (or in addition to) this volume. I make no apologies for ignoring 3D because these two books were designed to complement each other in the *Game Development* series.

Someone who has done some programming in Visual C++, CodeWarrior, Watcom C, Borland C++, GNU C++, or even Java or C# will understand the programs in this book.

Those with little or no coding experience will benefit from a C primer before delving into these chapters. I recommend many good C primers and C programming books in Appendix D. The emphasis of this book is on a cross-platform, open-source compiler, integrated development environment, and game library. You will not need to learn Windows or DirectX programming, and these subjects are not covered. The primary IDE is an open-source (freeware) program called Dev-C++, released by Bloodshed Software (<http://www.bloodshed.net>), and it is included on the CD-ROM. The game library is called Allegro; it is also freeware, open-source, and included on CD-ROM that accompanies this book. You have all the free tools you need to run the programs in the book, and then some! Using these tools, you can write standard Windows and DirectX programs with or without Allegro, and without the cost of an expensive compiler, such as Visual C++. This book is highly accessible to all C programmers, regardless of their platform of choice.

Use Your Favorite Compiler

Dev-C++ is a capable compiler package that includes an editor with source code highlighting. It uses the infamous GNU C++ compiler (GCC) to convert your chicken-scratch code into real programs with targets for Win32 or console programs and full support for DirectX 9. In other words, you might find Dev-C++ a useful companion for writing games with or without Allegro, and many of the sample programs in other Premier Press game development books will compile with Dev-C++ as well. It is a worthy, free, and easy-to-use alternative to a commercial compiler.

This book's source code and sample programs will run without modification on all of the following systems: Windows 9x/2000/Me/XP/2003, Mac OS X, Linux (any version), BeOS, QNX, and many other UNIX systems (IRIX, Solaris, Darwin, FreeBSD, to name a few) with X Windows. Believe it or not, these programs will also run under MS-DOS (DJGPP, Watcom C). That is almost every computer system out there. It's a sure bet if someone wants to use an old but mainstream C compiler, it will probably run the code in this book (with perhaps some limitations on compiling the Allegro library itself, which uses a modern makefile). I tell you this, not believing that you will need to write a game for MS-DOS, but just to demonstrate the versatility of Allegro.

Yet, at the same time, the Windows version of Allegro supports DirectX. The programs in this book will run in full-screen or windowed mode with support for just about any video card out there. Allegro is not an advanced, next-generation 3D engine; it is a cross-platform game library with a long history that dates back to the original Atari ST version. You might not care about cross-platform programming at this point, but imagine the possibilities if you were able to double the number of people who would play your game, just by compiling your game for other operating systems—and all without modifying any of your

source code. When is the last time you saw an online multiplayer game with Mac, Linux, and Windows players? Although I do not cover online multiplayer games in this book, they are a very real possibility using Allegro and standard TCP/IP socket libraries. As an example, I cover multi-threading in this book using a Windows port of the Posix Thread library, and the sample program I wrote to demonstrate multi-threading compiles under Windows and Linux without modification! The same is true for other libraries that conform to a standard, such as Berkeley Sockets for TCP/IP network programming.

This book is not *entirely* about cross-platform programming, though. I do discuss the subject in the first two chapters, but from that point forward, I simply focus on Allegro and specific game concepts, such as scrolling and animation. The overall theme and focus of this book are on writing games. To that end, you will develop a complete game and add to it in each chapter of the book, starting in Chapter 4.

Is This Book for You?

If you have any experience with the C language, then you will be able to make your way through this book. If you are new to the C language, I recommend against reading this book as your first experience with C because it will be confusing due to the extensive use of Allegro. (Very few standard C functions are used.) The example programs use a simple C syntax with no complicated interfaces or lists of include files. In fact, most of the programs will have a simple format like this:

```
#include "allegro.h"
int main(void)
{
    allegro_init();
    allegro_message("Welcome To Allegro!");
    return 0;
}
END_OF_MAIN();
```

This is a very simple program that is used as a test program for Appendix E, “Configuring Allegro for Microsoft Visual C++ and Other Compilers,” just as an example. This program simply verifies that the Allegro library has been linked with the main program and is working as expected. This particular program outputs to the console and does not run in graphical mode. Allegro provides comprehensive support for all of the video modes supported on your PC, including full-screen and windowed DirectX modes used by most commercial games. On the UNIX side, Allegro supports the X Window system, SVGAlib, and other libraries (as appropriate to the platform), providing a similar output no matter which system it is running on. For instance, the `allegro_message` function is displayed in a pop-up message box in Windows, but prints a message to a terminal window in Linux.

If you are a Windows user and you don't care about Linux, that won't be a problem. The screenshots presented in this book look exactly the same no matter what operating system you are using, and my choice of Windows or Linux in each particular case is simply for variety. Likewise, if you are a Linux user and you care not for Windows, you will not be limited in any way because every program in this book is tested on both Windows and Linux. The CD-ROM that accompanies this book includes the complete source code for the sample programs in this book, with project files for Visual C++ (Windows), Dev-C++ (Windows), and KDevelop (Linux). The tools on the CD-ROM include both Windows and Linux versions in most cases. If you are using an operating system other than these two, you should have no problem adapting the source code to your compiler of choice.

Do you like games, and would you like to learn how to create your own professional-quality games using some of the same tools used by professional game developers? This book will help you get started in the right direction toward that goal, and you'll have a lot of fun learning along the way! This is a very practical programming book, not rife with theory, so you will find many, many sample programs herein to reinforce each new subject.

System Requirements

The programs in this book will run on many different operating systems, including Windows, Linux, Mac OS X, and almost any UNIX variant that supports the X Window system. All that is really required is a decent PC with a video card and sound card.

Here are the recommended minimum hardware requirements:

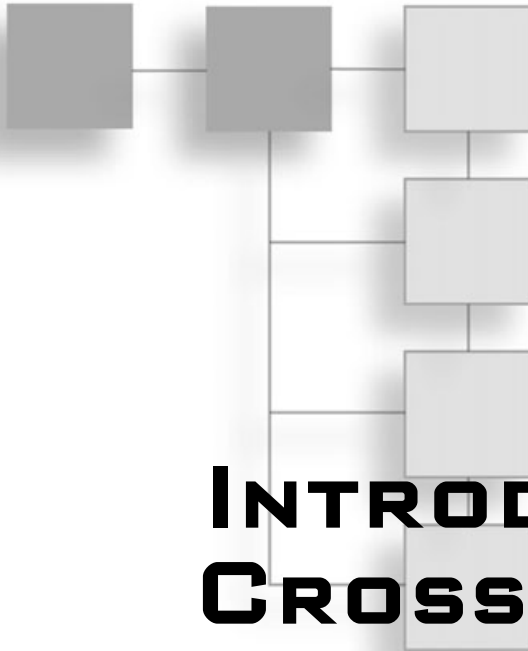
- Pentium II 300 MHz
- 128 MB memory
- 200 MB free hard disk space
- 8 MB video card
- Sound card

Book Summary

This book is divided into four parts:

- **Part I: Introduction to Cross-Platform Game Programming.** This first section provides all the introduction you will need to get started writing cross-platform games with Allegro and Dev-C++, with screenshots from both Windows and Linux. By the time you have completed this first set of chapters, you will have a solid grasp of compiling Allegro programs. This section concludes with a sample game called *Tank War* that you will enhance throughout the book.

- **Part II: 2D Game Theory, Design, and Programming.** This section is the meat and potatoes of the book, providing solid tutorials on the most important functions in the Allegro game library, including functions for loading images, manipulating sprites, scrolling the background, double-buffering, and other core features of any game. This section also provides the groundwork for the primary game developed in this book.
- **Part III: Taking It to the Next Level.** This section is comprised of more theoretical chapters covering basic artificial intelligence, a chapter on basic game physics, and a chapter about publishing your game.
- **Part IV: Appendixes.** This final section of the book provides answers to the chapter quizzes, a tutorial on numbering systems, a set of useful mathematical tables, tutorials on installing and using Allegro, a list of recommended resources, and an overview of the CD-ROM.



PART I

INTRODUCTION TO CROSS-PLATFORM GAME PROGRAMMING

CHAPTER 1
Demystifying Game Development3

CHAPTER 2
Getting Started with Dev-C++ and Allegro33

CHAPTER 3
Basic 2D Graphics Programming with Allegro71

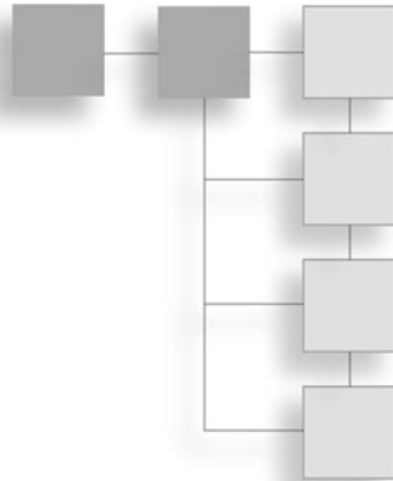
CHAPTER 4
Writing Your First Allegro Game119

CHAPTER 5
Programming the Keyboard, Mouse, and Joystick145

Welcome to Part I of *Game Programming All in One, 2nd Edition*. Part I includes five chapters that introduce you to the basic concepts of game development with Allegro. Starting with an overview of game development roots and covering the subject of motivation, this part goes into detail about how to use the free Dev-C++ compiler/IDE and the Allegro game programming library. Also, this part shows how to write, compile, and run several Allegro programs.

CHAPTER 1

DEMYSTIFYING GAME DEVELOPMENT



This chapter provides an overview of the game industry, the complexities of game development, and the personal motivations that drive members of this field to produce the games we love to play. Herein you will find discussions of game design and how your world view and upbringing, as well as individual quirks and talents, have a huge impact on not only whether you have what it takes to make it big, but also whether it is a good idea to work on games at all. There is more to writing games than motivation. While some programmers see game development purely as a monthly salary, some perceive games at a higher level and are able to tap into that mysterious realm of the unknown to create a stunning masterpiece. In this chapter, I discuss that vague and intangible (but all too important) difference.

I also give you a general overview of what it is like to work as a programmer. If you are interested in game programming purely for fun or as a hobby, I encourage you to absorb this chapter because it will help you relate to those on the inside and judge your own creations. When you consider that it takes a team to develop a retail game—and you are an individual—it is not unreasonable to believe that your own games are high in quality and worthy of note. What you must consider are total invested project hours and the size of the team. How does your solo project compare to a team game development project? You see, your solo (or rather, “indie”) game may be comparable to a retail game, all things being equal. One goal of this chapter is to help you realize this fact, to encourage you to continue learning, and to create games from your imagination. Whether you are planning a career in the game industry or simply partaking in the joy of writing games to entertain others, this chapter has something beneficial for you. After all, there are employed game programmers who only make their mark after going solo, and some solo game programmers who only make their mark after joining a team. Taking games seriously from the start is one way to attract attention and encourage others to take your work seriously.

Here is a breakdown of the major topics in this chapter:

- Gaining a high-level view of game development
- Recognizing your personal motivations
- Getting into the spirit of gaming
- Getting an introduction to Dev-C++ and Allegro

Introduction

Before I delve into the complexities of learning to write a game, I want to take a few moments to discuss the big picture that surrounds this subject. I'd like to think that some of you reading this book very likely will enter the game industry as junior or entry-level programmers and make a career of it. I am thrilled by that possibility—that I may have contributed in some small way to fulfilling a dream. I will speak frequently to both the aspiring career game programmer and the casual hobbyist because both have the same goals—first, to learn the tricks and techniques used by professional game programmers, and then to learn enough so it is no longer difficult and it becomes fun. Programming is difficult already; *game* programming is exponentially more difficult. But by breaking down the daunting task of writing a modern game, you can learn to divide and conquer, and finish a great game! Thus, my goal in this chapter is to provide some commentary along those lines while introducing you to the technologies used in this book—namely, the C language and the Allegro game library.

First, a disclaimer—something that I will repeat several times to nail the point home: DirectX is *not* game programming. DirectX is one library that is indisputably the most popular for Windows PCs. However, consoles such as the Sony PlayStation 2, Nintendo GameCube, Nintendo Game Boy Advance, and the many other handheld devices do not use DirectX or anything like it (although Xbox does use DirectX). There are dozens of DirectX reference books disguised as game programming books, but they often do little other than expose the interfaces—DirectDraw, Direct3D, DirectInput, DirectSound, DirectMusic, and DirectPlay. Talk about getting bogged down in the details! In my opinion, DirectX is the means to an end, not the goal itself. Learning DirectX is optional if your dream is to write console games (although I recommend learning as much as possible about every subject).

For the newcomer to game development, this misconception about the nature of some so-called game programming books can be a source of consternation. Beginners can be impatient (as I have been myself, and will discuss later in this chapter). Let me summarize the situation: You want to get something going quickly and easily, and *then* you want to go back and learn all the deep and complicated details, right? I mean, who wants to read an 800-page programming book before they actually get to write a game?

Practical Game Programming

This book focuses on the oft-misused phrase “game programming” and has no prerequisites. I don’t discuss Windows or DirectX programming at all in this book. For some excellent reference books on those subjects (which I like to call *logistical* subjects), please refer to Appendix D, “Recommended Books and Web Sites.” If I may nail the point home, allow me to present a simple analogy—one that I will use as a common theme in this and other chapters. Writing a game is very similar to writing a book. There are basic tools required to write a game (such as a compiler, a text editor, and a graphic editor), just as there are tools required to write a book (such as a word processor, a dictionary, and a thesaurus). When you are planning a new project, such as a game, do you worry about electricity? As such, when you are planning a new book, would you worry about the alphabet? These things are base assumptions that we take for granted.

I take the operating system completely for granted now, and I try to abstract my computing experience as much as possible. It is a liberating experience when I am able to get the same work done regardless of the electronics or operating system on my computer. Therefore, I take those things for granted, whether I am using Windows Explorer or GNOME, Internet Explorer or Mozilla, Visual C++ or Dev-C++. This is an important concept that I encourage you to consider because the game industry is in a constant state of flux that conducts the vibrations of the entire computer industry.

The concept of a “new computer” is important to the general public, but to a computer industry professional, “new” is a very relative term that only lasts a few weeks or months at most. Everyone has his or her own way of dealing with constant change, and it is part of the experience of working with computers. (Those who can’t handle it never last long in this industry.) Rather than seeing change as a tidal wave and trying to keep ahead of it, I often let the wave crash over my head, so to speak, and wait for the next wave. It’s an intriguing experience, allowing high technology to pass you up and zoom ahead. But do you know why there is some wisdom in skipping a trend now and then? Because technology is not only in a constant state of change, but it is also in a constant state of experimentation. Not every new “improvement” is good or accepted. Remember videodiscs? Probably not! The movie industry had to rethink videodisc technology in part because the discs resembled vinyl records, which the public perceived as old technology.

For example, the computer hardware industry markets heavily for the need to constantly upgrade computers. It is logical that these companies would do so because the general public really believes that everything is obsolete year by year. In fact, it is the gross inefficiency of the software that makes this so. Rather than grasping at the latest everything with a must-have belief system, why not continue to use known, stable systems and stand up to the frequent tidal waves of technology? What one calls progress, another calls marketing. Games have single-handedly pushed the personal computer industry to extraordinary new

heights in the past decade due for the most part to advances in graphics technology. But that cutting edge leaves a lot of well-meaning and talented folks out in the cold when they might otherwise be developing well-loved games.

So we come back to the point again: What is the cutting edge of game development, and what must I do to write great games? For the first part, the cutting edge is gameplay, not the latest 3D buzzword. Second, to write a great game, you must be passionate and talented. Studying the subject at hand (game programming) is another factor—although it is the focus of this book! For my own inspiration, I look at games such as *Sid Meier's Civilization III* and *Age of Wonders: Shadow Magic*, among other recent 2D titles. You can find your inspiration in whatever subject interests you, and it need not always be a video game.

Goals Revisited

One of the aspects of this book that I want to emphasize early on is that my goal is to reach a majority of hobbyists and programmers who are either aspiring to enter the game industry as career programmers or who are simply writing games for the fun of it. As I explained in the Introduction, this book won't hold your hand because there is so much information to cover. At the same time, it's my job to make a difficult subject easy to comprehend; if you have some fun along the way, that's even better. I don't want to simply present and discuss how to write 2D graphics code; my goal is for you to master it.

By the time you're finished with this book, you'll have the skills to duplicate any game released up to the late 1990s (before 3D hardware acceleration came along for PCs). That includes a huge number of games most often not regarded by the “twitch generation”—that is, those gamers who would describe “strategy” as which direction to circle strafe an enemy in a first-person shooter, the best kind of car to “jack” to make the most money, or how to escape via a side alley where the cops never follow you. We can poke fun at the twitch generation because they wouldn't know what to do with a keyboard, let alone how to write game code; therefore, they are not likely to read this book. But if there are any twitch gamers now reading, I congratulate you!

The High-Level View of Game Development

Game development is far more important to society than most people realize. Strictly from an economic point of view, the design, funding, development, packaging, delivery, and sale of video games (both hardware and software) employs millions of workers around the world. There are electronics engineers building the circuit boards and microprocessors. Programmers write the operating systems, software development kits, and games. Factory workers mass-produce the packaging, instructions, discs, controllers, and other peripherals. Technical support workers help customers over the phone. There are a large number of investors, business owners, managers, lawyers, accountants, human resource workers, network

and PC technical support personnel, and other ancillary job positions that support the game industry in one way or another. What it all amounts to is an extraordinarily complex system of interrelated industries and jobs, and millions of people who are employed solely to fill the shelves of your local video game store. The whole point of this is simply to entertain you. Because we're talking about high-quality interactive entertainment, we have a tendency to spend a lot of money for it, which increases demand, which drives everyone involved to work very hard to produce the next bestseller.

Although this narrative might remind you of the book publishing industry, where there are many people working very hard to get high-quality books onto store shelves, I submit that games might be more similar to motion pictures than to books. All three of these subjects are closely related forms of entertainment, with music included. Books are turned into movies, movies into video games, and both movies and video games into books. All the while, music soundtracks are available for movies and video games alike. Much of this has to do with marketing—getting the most income from a particular brand name. One excellent side effect of this is that many young people grow up surrounded by themes of popular culture that spawn their imaginations, thus producing a new generation of creative people every few years to work in these industries.

Consider the effect that science fiction novels and movies have had on visionaries of popular culture, such as Gene Roddenberry and George Lucas, who each pushed the envelope of entertainment after being inspired by fantastic stories of their time, such as *The Day the Earth Stood Still* and *The Twilight Zone*, to name just two. Before these types of programs were produced, Hollywood was enamored with westerns—stories about the old West. What was the next great frontier, at least for an American audience? Having spread across the continent of North America, and after fighting in two great and terrible world wars, popular culture turned outward—not to Earth's oceans, but to the great interstellar seas of space. What these early stories did was spurn the imaginations of the young up-and-coming visionaries who created *Star Trek*, *Lost in Space*, *Star Wars*, and action/adventure themes such as *Indiana Jones*, set in a past era (where *time* is often associated with *space*). These are identifiable cultural icons.

The game industry is really the next generation of entertainment, following in the footsteps of the great creative powerhouses of the past few decades. Games have been growing in depth and complexity for many years, and they have come to be so entertaining that they have eclipsed the motion picture industry as the leading form of entertainment. But just as movies did not replace books, neither will games replace movies as a dominant player. Although one might eclipse the others in revenue and profit, all of these industries are interrelated and interdependent.

Thinking hypothetically, what do you suppose will be the next stage of cutting-edge entertainment, the likes of which will supercede games as the dominant player? In my opinion, we have not seen it yet and we might never see it. I believe that books, music, movies, and

video games will continue unheeded to inspire, challenge, and entertain for decades to come. But I do hold an opinion that is contrary to my last statement. I believe that western society will embrace entertainment less and adopt more productive uses for games in the decades to come. Why do I feel this way, you might ask? Momentum and progress. Games are already being used for more than just entertainment. They are being used by governments to train soldiers in the strategy and tactics of a modern battlefield, one in which military commanders no longer have the luxury of experiencing for real. Without real long-term engagements like those during World War II (battles since that time have been skirmishes in comparison), modern militaries must rely on alternative means of training to give troops a feel for real battle. What better solution than to play games that are visceral, utterly realistic, shocking in unpredictability, and awe-inspiring to behold? Who needs a real battlefield when a game looks and feels almost like the real thing?

I have now explored several areas of our society that benefit from the game industry. What about gamers themselves—you, me, and other video game fanatics? We love to play games because it is exhilarating to conquer, pillage, destroy, and defeat an opponent (especially if he or she is a close friend or relative). But there is the converse to this point of view, regarding those games that allow you to create, imagine, build, enchant, and express yourself. Some games are so artistic that it feels as if you are interacting with an oil painting or a symphonic orchestra. To conclude this game brings forth the same set of emotions you feel upon finishing a good book, an exceptional movie, or an orchestral performance—exhilaration, joy, pride, fascination, appreciation, and yet a tinge of disappointment. However, it is that last emotion that draws you back to that book, movie, game, painting, or symphony again, where it brings you some happiness in life. This experience transcends mere entertainment; it is a joy felt by your soul, not simply a sensual experience in your mind and through your eyes and ears.

Interactivity has much to do with some of the new lingo used to describe the game industry. Although insiders won't mince words, those who are concerned with public consumption and opinion prefer to call the game industry a form of interactive or electronic entertainment. Game programming has become game development. Outlining the plot of a game has become game design. Very lengthy scripts are now written for games, and some designers will even storyboard a game. Do you begin to see similarities to the movie industry?

Storyboarding is a process in which concept artists are hired to illustrate the entire game scene by scene. This is a very expensive and time-intensive process, but it is necessary for complicated productions. Some films (or games, for that matter) are rather simple in plot: Aliens have invaded Earth, so someone must stop them! Although a storyboard might help a hack-and-slash type of game, it is often not necessary, particularly when the designer and developers are intimately familiar with the subject matter. For instance, think about

a game adaptation of a novel, such as Michael Crichton's *Jurassic Park*. The developers of a game based on a novel do not always have the benefit of a feature film, as was the case with *Jurassic Park* and other movies based on Michael Crichton novels. Simply reading the book and watching the movie is probably enough to come up with a basic idea for what should happen in the game; you probably don't need to storyboard.

Why do I feel that this discussion is important? It is absolutely relevant to game development! In fact, "game programming" has become such a common phrase in video game magazines, on Web sites, and in books that it is often taken for granted. What I'm focusing on is the importance of perspective. There is a lot more to consider than just what to name a program variable or what video resolution to use for your next game. You need to understand the big picture, to step away from the tree to see the entire forest.

Recognizing Your Personal Motivations

Why do you want to learn game programming? I want you to think hard about that question for a moment, because the time investment is great and the rewards are not always up to par in terms of compensation. *You must love it*. If you don't love absolutely everything about video games—if you don't love to argue about them, review them online, and play them obsessively—then I have some good but somewhat hard advice. Just treat video games as an enjoyable hobby, and don't worry too much about "breaking in" to the game industry or getting your game published. Really. Because that is a serious source of stress, and your goal is supposed to be to have fun with games, not get frustrated with them.

note

For a fascinating insider narration of the video game industry's early years, I highly recommend the book *Hackers* by Steven Levy, which puts the early years of the game industry into perspective. For a historic ride down memory lane, be sure to read *High Score! The Illustrated History of Electronic Games* by Rusel DeMaria and Johnny Wilson (former editor of *Computer Gaming World*), a full-color book with hundreds of fascinating photos. Browsing the local bulletin board systems in the late 1980s and early 1990s to download shareware games was also a fun pastime. For an intriguing look into this era, I recommend *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture* by David Kushner.

I was inspired by games such as *King's Quest IV: The Perils of Rosella*, *Space Quest III: The Pirates of Pestulon*, *Police Quest*, *Hero's Quest: So You Want To Be A Hero?*, and other extraordinarily cool adventure games produced by Sierra. There were other companies, too, such as Atari, Electronic Arts, Activision, and Origin Systems. I spent many hours playing *Starflight*, one of the first games that Electronic Arts published in 1985 (and one of the greatest games made at the time) and the sequel, *Starflight II: Trade Routes of the Cloud Nebula*, which came out in 1989.

Decision Point: College versus Job

In the modern era of gaming, a college education is invaluable. What if you grow tired of the game industry after a few years? Don't cringe; this is a very real possibility. A lot of hardcore gamers have moved on to casual gaming or given it up entirely while pursuing other careers.

Focus every effort on writing complete and polished games, however big or small, and consider every game as a potential entry on your résumé. If you want to work on games for a living, go for it full tilt and don't halfheartedly fool around about it. Be serious! Go get a job with *any* game studio and work your way up. On the other hand, if you want to get involved in high-caliber games, then go to college and focus heavily on your studies. Let the game industry pass you by for a short time, and when you graduate, you will be ready and equipped to get a great job. There are some really great high-tech colleges that are offering game programming degree programs. University of Advancing Technology in Tempe, Arizona, for instance, has associate's, bachelor's, and master's degree programs in game development! Take a look at <http://www.uat.edu>.

Once you have made the decision to go for it, it's time to build your level of experience with real games that you will create on your own. Don't assume that one of your hobby games isn't good enough for an employer to see. Most game development managers will appreciate brimming enthusiasm if you have the technical skills to do the job. Showing off your previous work and recalling the joy of working on those early games is always enjoyable for you and the interviewer. They want to see your *personality*, your *love* of games, and how you spent *hundreds* of hours working on a particular game, fueled by an uncontrollable drive to see it completed. Your emphasis should be on completed games. Most important, always be genuine.

I would go so far as to say that having a dozen shareware games (of good quality) on your résumé is better than having worked on a small part of a commercial game. Yes, suppose you did work on a retail game. That doesn't guarantee choice employment with another company. What sort of work did you do on that game—level editors, unit editors, level design, play testing? These are common tasks for entry-level programmers on a professional team where the “cool” positions (such as 3D engine and network programming) are occupied by the highly skilled programmers with proven track records who always get the job done quickly.

The best hobbies will often pay for themselves and might even earn a profit. If you have a full-time job that is otherwise fine, then you might turn the hobby of game programming into a money-making adventure. Who knows—you might release the next great indie game.

Every Situation Is Unique

There are many factors to consider in your own determination, and there is no best direction to take in life. We all just try to do the best that we can do, day by day and year by

year. I recommend that you pursue a career that will bring you the most enjoyment while still earning the highest possible salary. You might not care about salary at this point in your life; indeed, you might feel as if you would pay someone to hire you as a game programmer. I know that feeling all too well! I thought it was a strange feeling, getting paid to work on a retail game. When that game came out in stores and I saw it on the shelf, then it was an exhilarating feeling.

However, most of the world does not feel the same way that you do about video games. Very few people bother to read the credits. The feeling of exhilaration is really an internal one, not widely shared. You might already feel that this is true, given your own experience with relatives and friends who don't understand why you love games so much and why you wig out over the strangest things.

I remember the first time I discovered Will Wright's *Sim City*; it was in the late 1980s. It was quite an educational game, but extremely fun, too. Traveling with my parents, I would point out along the road, "Residential zone. Commercial zone. Ah ha! There's an industrial zone. Sure to be a source of pollution." I would note traffic jams and point out where a light rail alongside the road would ease the traffic problems. The fact is, the way you feel about video games has a strong bearing on whether you will succeed when the going gets rough, when the hours are piled on and you find yourself with no free time to actually play games anymore. All you have time to do is write code, and not even the most interesting code at that. But that spark in your eye remains, knowing that you are helping to complete this game, and it will go on your résumé as an accomplishment in life, maybe as a stepping stone in your career as a programmer.

Another argument that you might consider is the very real possibility that you could always go to college later and focus on your career now, especially if you have a lead for a job at a game company. That trend seems to be dwindling because games are now exceedingly complex projects that require highly trained and educated teams to complete them. Any self-taught programmer might have found corporate employment in the 1970s and 1980s, but the same is no longer the case with game companies. Now it has become an exceedingly competitive market. As you already know, competition causes quality to rise and costs to go down. A programmer with no college degree and little or no experience will have a very difficult time finding employment with a recognizable game company. Perhaps he can find work with one of the few hundred independent studios, but even private developers are in need of highly skilled programmers.

You might find more success by taking the indirect route to a career in game development. Many developers have gone professional after working on games in their spare time, by selling games as shareware or publishing them online. And there are as many success stories for high school graduates as there are for college graduates. As I said, every situation is unique. During this period of time, you can hone your skills, build your résumé of games (which is absolutely critical when you are applying for a job in the game industry),

develop your own game engines, and so on. Even if you are interested in game programming (which is a safe bet if you are reading this book!) just as a hobby, there is always the possibility that you will come up with something innovative, and you might be surprised to receive an unexpected job offer.

A Note about Specialization

As far as specialization goes, there is very little difference between programming a game for console or PC—all are based on the C or C++ language. These are two distinct languages, by the way. It is out of ignorance that many refer to C and C++ interchangeably, when in fact they are very different. C is a structured language invented in the 1970s, while C++ is an object-oriented language invented in the 1980s. It is a given that you must know both of these languages (not just one or the other) because that is the assumption in this industry—you simply must know them both, without exception, and you should not need a programmer's reference for most of the standard C or C++ libraries (although there are some weird functions that are seldom used). If you are a capable programmer (from a Windows, Linux, Mac, or other background), you know C and C++, and you have some experience with a game engine or library (such as Allegro), then you should be able to make your way when working on a console, such as the PlayStation 2, Xbox, or GameCube.

The software development kits for consoles typically include libraries that you must link into your program when the program is compiled and linked to an executable file. Many game companies now produce games for all of the console systems and the PC, as well as some handheld systems (such as the Game Boy Advance). Once all of the artwork, sound effects, textures, levels, and so on, have been created for a game, it is economically prudent to reuse all of those game resources for as many platforms as possible. That is why many games are released simultaneously for multiple consoles. The cost of porting a game is just a fraction of the original development cost because all of the hard coding work has been done. The game's design is already completed. Everything has been done for one platform already, so the porting team must simply adapt the existing game for a different computer system (which is essentially what a video game system is). Since all of the code is already in C or C++ (or both), the porting team must simply replace platform-specific function calls with those for the new platform.

For instance, suppose a game for the PC is being ported to Xbox—something that is done all the time. The Xbox is very similar to a Windows PC, with a Windows 2000 core and a custom version of DirectX. There is no keyboard or mouse, just a controller. Porting a PC game requires some forethought because there is a lot of input code that must be converted so the game is operated from a controller. As an example, one of the most popular online PC games of all time, *Counter-Strike*, was ported to Xbox and features online play via the Xbox Live! network.

The usual setup for a PC game includes the use of keyboard in tandem with mouse—usually the ASDW configuration (A = left, D = right, W = forward, D = backward) while using the mouse to aim and shoot a weapon. Also, you use the CTRL key to crouch and the spacebar to jump. If your mouse has a mouse-wheel, you can use that to scroll through your available weapons (although the usual way is with the < and > keys).

I have always found this to be a terribly geeky way to play a game. Yes, it is faster than a controller. But it's like we have been forced to use a data entry device for so long just to play games that we not only accept it, but we defend it. I've heard many elite *Counter-Strike* players proclaim, "I'll never switch to a controller!" The fact of the matter is, when you get used to controlling your character using dual analog sticks and dual triggers on a modern console controller (such as the Xbox Controller S, shown in Figure 1.1), it is easy to give up the old keyboard/mouse combo.



Figure 1.1 Xbox Controller S

Counter-Strike was originally a *Half-Life* *mod* (or rather, expansion). To play the original *Counter-Strike*, you had to already own *Half-Life*, after which *Counter-Strike* was a free (but very large) download. Porting the game to Xbox must have been a major undertaking if it was truly rewritten just for the Xbox. Based on the similarity to the now-aged PC game, I would suspect that it is the same source code, but very highly modified. There are no Xbox enhancements that I can see after having played the game for several years on the PC. It is interesting to see how the developers dealt with the loss of the keyboard/mouse input system and adapted the game to work with a controller. The in-game menus use a convenient, intelligent menu system in which you use the eight-way directional pad to purchase gear at the start of each game round.

Regardless of the differences in input control and hardware, the source code for a console or a PC game is very similar, and all of it is written in C or C++. (The biggest difference are the development environment and game libraries, or SDKs.) One common practice at a game studio is to fabricate a development system in which the SDK of each console is abstracted behind *wrapper code*, which is a term used to describe the process of wrapping an existing library of functions with your own function calls. This not only saves time, but it also makes it easier to add features and fix bugs.

Game Industry Speculation

According to Jupiter Research (<http://www.jupiterresearch.com>), the game industry will continue to grow, having reached an estimated \$12 billion revenue during 2003. Although console sales amount to more than PC game sales, there are many more PC gamers than console gamers, and the gap will continue to widen.

I have a theory about this apparent trend. I have seen the growth of consoles over the last five years, and I am convinced that console games will be more popular than PC games in a few years. It is just a simple matter of economics. A \$200 console is as capable and as powerful as a \$1,500 PC. Not too long ago I was a frenetic upgrader; I always found an excuse to spend another \$500 on my PC every few months.

When I stopped to look at this situation objectively, I was shocked to learn that I had been spending thousands annually—on games, essentially. Not just retail games, but the hardware needed to run those games. It seemed to be a conspiracy! The hardware manufacturers and software game companies were in league to make money. Every six months or so, new games would be released that required PC upgrades just to run. One benefit that the consoles have brought to this industry is some platform stability, which makes it far easier to develop games. Not only can you (as a game programmer) count on a stable platform, but you can push the boundaries of that platform without worrying about leaving anyone with an aging computer behind. No newly released PC game will run on a computer that is five years old (in general), but that is a common practice for the average five-year lifespan of a game console.

Given this speculation and the trends and sales figures that seem to back it up, it is very likely that the PC and console game industries—which were once mostly independent of each other—will continue to grow closer every year. That is why it is important to develop a cross-platform mindset and not limit yourself to a single platform, such as Windows. Mastery of C and C++ are the most important things, while your specific platform of choice comes second. Regardless of your proficiency with Windows and DirectX, I encourage you to learn another system. The easiest way to gain experience with console development is to learn how to program the Nintendo Game Boy Advance (GBA) because open-source tools are available for it.

Emphasizing 2D

There is a misunderstanding among many game players as well as programmers (all of whom I will simply refer to as “gamers” from this point forward) that 2D games are dead, gone, obsolete, forever replaced by 3D. I disagree with that opinion. There is still a good case for working entirely in 2D, and many popular *just-released* games run entirely under a 2D game engine that does not require a 3D accelerator at all. Also, numerous games that can only be described as cult classics have been released in recent years and will continue to be played for years to come. Want some examples?

- Sid Meier’s *Civilization III* with *Play the World* and *Conquests* expansions
- *StarCraft* and the *Brood War* expansion
- *Diablo II* and the *Lord of Destruction* expansion
- *Command & Conquer: Tiberian Sun* and the *Firestorm* expansion

- *Command & Conquer: Red Alert 2* and *Yuri's Revenge*
- *Age of Empires* and the *Rise of Rome* expansion
- *Age of Empires II* and *The Conquerors*
- *Age of Mythology* and *The Titans* expansion
- *The Sims* and a dozen or so expansions and sequels
- *Real War* and the *Rogue States* expansion

What do all of these games have in common? First of all, they are all bestsellers. As you might have noticed, they all have one or more expansion packs available (which is a good sign that the game is doing well). Second, these are all 2D games. This implies that these games feature a scrolling game world with a fixed point of view and various fixed and moving objects on the screen. Fixed objects might be rocks, trees, and mountains (in an outdoor setting) or doors, walls, and furniture (in an indoor setting). With a few exceptions, these are all PC games. There are several-hundred console and handheld games that all feature 2D graphics to great effect that I could have listed. For instance, here are just a handful of exceptional games available for the Game Boy Advance:

- *Advance Wars*
- *Advance Wars 2: Black Hole Rising*
- *Super Mario World: Super Mario Advance 2*
- *Yoshi's Island: Super Mario Advance 3*
- *The Legend of Zelda: A Link to the Past*
- *Sword of Mana*
- *Final Fantasy Tactics Advance*
- *Golden Sun: The Lost Age*

What makes these games so compelling, so hot on the sales charts, and so popular among the fans? It is certainly not due to fancy 3D graphics with multi-layer textures and dynamic lighting, representative of the latest first-person shooters. What sets these 2D games apart are the fantastic gameplay and realistic graphics for the characters and objects in the game.

Finding Your Niche

What are your hobbies, interests, and sources of entertainment (aside from your PC)? Have you considered that what interests you is also of interest to thousands or millions of other people? Why not capitalize on the fan base for a particular subject and turn that into a game? Nothing beats experience. When it comes to designing a game, there is no better source on a particular subject than a diehard fan! If you are a fan of a particular sci-fi show or movie, perhaps, then turn it into your vision of a game. Not only will you have a lot of