iPhone Games Projects

DAVE MARK, SERIES EDITOR

PJ CABRERA
JOACHIM BONDO
AARON FOTHERGILL
BRIAN GREENSTONE
OLIVIER HENNESSY
MIKE KASPRZAK
MIKE LEE
RICHARD ZITO
MATTHEW AITKEN
CLAYTON KANE

Apress®

iPhone Games Projects

ISBN-13 (paperback): 978-1-4302-1968-2

ISBN-13 (electronic): 978-1-4302-1969-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Clay Andres
Developmental Editor: Douglas Pundick
Lead Author and Technical Reviewer: PJ Cabrera
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager | Production Director: Grace Wong
Copy Editors: Kim Wimpsett, Marilyn Smith
Associate Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor | Interior Designer: Diana Van Winkle
Proofreader: Nancy Bell
Indexer: BIM Indexing & Proofreading Services
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`. You will need to answer questions pertaining to this book in order to successfully download the code.

To my parents, Pedro and Cecilia: LQM
—PJ Cabrera

To my brother Adam, who does 50% of the work and gets 0.01% of the mentions ;)
—Aaron Fothergill

To my teammates, past and present, who have made me the engineer I am
—Mike Lee

Contents at a Glance

Contents

AARON FOTHERGILL

BRIAN GREENSTONE

OLIVIER HENNESSY
AND CLAYTON KANE

MICHAEL KASPRZAK

MIKE LEE

Foreword

iPhone games are hot, hot, hot! As I write this, there are more than 40,000 apps on the App Store, of which nearly 9,000 are in the Games category, the largest category by far. The next largest category is Entertainment, at just over 5,000 apps. There are nearly 40% more games on the App Store than any other kind of app.

Games is not only the biggest category on the App Store, but it is also the best-selling category. During the promotion of its one billionth download, Apple provided a list of all-time most popular apps. Of the top-20 all-time paid apps, more than 14 were games. Many of these apps were on the top-10 paid apps list at one time or another during the nine months the App Store had been in business. Many are still on the top-100 list. These sold thousands of copies a day in their time, with daily revenue between a few thousand to tens of thousands of dollars.

With such great numbers, it is understandable why the interest in developing iPhone games is so high. It is probably the reason you are reading this foreword. You couldn't have picked up this book at a better time. If you want to get in on the fun and potential profit of making iPhone games, the time to get started is now, and this book is your ticket there!

This book contains lots of information from master independent iPhone game developers—information that is hard to find anywhere else. Some of the authors of this book are responsible for all-time popular games:

- Brian Greenstone developed Enigmo and Cro-Mag Rally.
- Aaron Fothergill developed Flick Fishing.
- Mike Lee developed the original Tap Tap Revolution, the most downloaded game in App Store history.

This book also features a chapter by award-winning game developer Mike Kasprzak, finalist for Best Mobile Game in the Game Developers Conference's Independent Games Festival Mobile 2009. Other authors, including Richard Zito, Joachim Bondo, and Olivier Hennessy, have received glowing reviews and accolades for their games.

The content in this book is phenomenal. The authors provide a variety of points of view and different approaches to iPhone development, showing you how iPhone games are made with various technologies. You will gain knowledge of how to optimize your game using iPhone SDK tools such as Instruments and Shark, as well as optimization tricks that the masters have

learned through the crucible of experience. You will also get invaluable insight into game design, arguably the most important aspect of game creation. If your game is not properly designed, all the technical prowess in the world isn't going to help it be popular.

I am honored to have worked with the independent game development professionals in the creation of this book. I have learned from their experience thanks to the chapters they have written. And with this book, you, too, can acquire the know-how and insight to make the next generation of all-time most popular and award-winning games.

Now, pick up this book and get started!

PJ Cabrera
FoneGears Systems LLC founder and lead developer

About the Lead Author and Technical Reviewer

PJ Cabrera is a software engineer with more than 12 years of experience developing information systems in various industries, programming in C, C++, Java, PHP, Python, and Ruby. But his real passion for many years has been hacking gadgets (i.e., turning a Sega Dreamcast into a NetBSD router, or running Android and Debian GNU/Linux on a Palm TX) and making home-brewed games for consoles such as Dreamcast, PlayStation 2, GameBoy Advance, and PSP. He is very excited that he can finally share his creative side on iPhone and XBox 360 with the general public through the App Store and XNA Community Games.

Acknowledgments

Special thanks to Glenn Cole who reviewed every chapter, provided quality control, and contributed the introduction.

I would like to acknowledge the support of my family as I worked on this book. Thanks for putting up with my absence every evening and weekend I had to work on my chapters, and for your unreserved love.

And thanks to my friends Ronada Wanner, Lenny Ramirez, Cesar Otero, and Luis Pulido, for cheering me on and believing in me.

This book would not have been possible without the tireless professionalism of many staff at Apress. Many thanks go out to Clay Andres, for believing in me; our project manager Grace Wong, for her gentle reminders of my looming immutable deadlines; Caroline Rose, Kim Wimpsett, and Marilyn Smith for editing and copy editing duties (my chapter is much improved because of their efforts); Laura Esterman, Douglas Pundik, and Glenn Cole in production editing (the chapters look awesome!). To everyone else at Apress, thanks for everything you did to make this book a reality.

PJ Cabrera

Introduction

First, the obvious: the iPhone rocks. Less obviously, so does the iPod touch—no phone, no cellular network (and the associated monthly fee), just a wonderful game-playing, application-downloading, insanely great Cocoa touch machine. Consumers love them both for their stability, ever-present network access, ineffable coolness, and (as we developers know firsthand) all of those great third-party apps.

Apple's App Store rocks, too, and as the owners of these devices have made it clear, games rock the most! The *Wall Street Journal* reported data from Mobclix showing that over a quarter of the applications in Apple's App Store are games (see `http://blogs.wsj.com/digits/2009/03/23/no-recession-for-iphone-game-apps/`). Venture capitalists are getting in on the action as well, financing game developer Ngmoco to the tune of $10 million. There has even been a conference devoted to gaming on the iPhone (see `http://www.igsummit.com/`).

Think of this book as a guide and companion through your personal journey of game development for the iPhone and iPod touch. And like any good travel guide, it's full of tips, highlights, and invaluable suggestions for avoiding the costly and time-consuming mistakes of the early explorers who have gone before you.

But the life of the game developer is not all fun and, well, games, so there's more. After all, games development requires knowledge of the same basic coding and language skills as any form of app development. So we've loaded the book with code you can reuse as building blocks for your own apps. Even those developers who don't have game development as their highest priority will find a lot of solid information here for use in their applications.

Who This Book Is For

This book is for everyone writing game applications for iPhone and iPod touch. It's assumed that you already have some development experience, both in general and specifically with Objective-C and Cocoa on the iPhone. We have made no attempt to cover every aspect of every API for every sort of game imaginable. We have left a lot of room for your imagination, and have instead filled the book with key concepts and compelling stories we hope you will find both useful and inspirational.

We're able to do this because this book is part of Apress's comprehensive road map of Mac and iPhone developer titles. So, if you have experience with Objective-C and Cocoa, but not specifically for the iPhone, *Beginning iPhone Development* by Dave Mark and Jeff LaMarche is a great way to fill in the gaps. If you have experience with an object-oriented language like C++, C#, or Java, a glance through the first few chapters of Apple's free online reference "The Objective-C 2.0 Programming Language" (available from `http://developer.apple.com`) may be sufficient before moving to *Beginning iPhone Development*. Otherwise, you'll want to begin either with *Learn Objective-C on the Mac* by Mark Dalrymple and Scott Knaster, if you have experience with C but not with Objective-C, or *Learn C on the Mac* by Dave Mark, if you're starting from scratch or have not used a C-like language like PHP, ASP, or JavaScript.

Just as some portions of the iPhone SDK are based on standard C rather than on the Cocoa frameworks, so, too, do some portions of this book use standard C pointers and structs. If you are not comfortable with these, *Expert C Programming* by Peter van der Linden (published by Prentice Hall) will help to bring you up to speed.

What's in the Book

The journey begins by giving serious consideration to the user experience. We've all heard that an application should be well polished before being released, but what does that really mean? Joachim Bondo, author of the elegant chess application Deep Green, shares his thoughts on what makes a great user interface, and how he decides which features to add and which to leave out. Seeing all of the thought that went into Deep Green will show you what polish really means.

In Chapter 2, PJ Cabrera shares his ideas on bringing back the social aspect of games. He backs this up with code to show how to call a web service, how to parse the resulting XML, how to post data back to the web service, and for truly RESTful web services, how to simplify the task with a drop-in library. As a bonus, you will also see how to prototype a RESTful web service using Ruby on Rails, so you can get started quickly.

Long-time developers will appreciate how the old text game Star Trek has evolved into using OpenGL ES in the game Space Hike. Aaron Fothergill explains the process in Chapter 3, and also shares a nifty technique for saving struct-based data.

Chapter 4 finds "rock star" Brian Greenstone of Pangea Software—makers of Bugdom, CroMag Rally, and more—sharing his "must-do" list for iPhone action games. Brian also guides you through performance analysis using both Instruments and Shark. The first time I read this chapter, I was exhausted by the end.

In Chapter 5, Olivier Hennessy and Clayton Kane share some of their tricks of the trade in the form of the game design document (think of it as a plan for success). They also provide an overview of the available game engines and why you should consider using one.

Are you thinking about other platforms as well as the iPhone? In Chapter 6, Mike Kasprzak shares his techniques for writing an OpenGL ES–based game for the iPhone that can also run on Windows and on Linux with minimal changes. Also included are techniques for "frame skipping" and physics simulation.

Mike Lee—previously of Delicious Monster, cofounder of Tapulous, and now at Apple—shares his insights on the code optimization process in Chapter 7. Mike goes into the thought process behind optimization, including key observations that help drive toward a solution. He also makes a strong argument for embracing the vendor's frameworks, even when you're tempted to roll your own.

The book concludes with Richard Zito and Matthew Aitken writing about multiplayer gaming. They explore the various techniques for networking, demonstrating the pros and cons of each. Low-level sockets are covered, both with and without Bonjour. You'll learn how to "publish" the game's availability on the network, and how to allow users to join in.

We've also added a special online bonus chapter by Jamie Gotch on how to implement a clone of the game Puyo, a falling blocks game, using the A* (A-star) path-finding algorithm. A* is typically used to implement artificial intelligence for unit movement in genres such as real-time strategy, adventure, and role-playing games. This chapter shows how to use A* to find adjacent blocks of the same color in the game. Jamie's game Fieldrunners was the winner of the Game Developers Conference's Independent Games Festival Mobile 2009 competition.

There is something here for everyone, from experienced OpenGL developers, to enterprise software developers, to those who have completed only an introductory text. I know, because I've read every chapter and checked every line of code. It has been an illuminating journey through applications so diverse and rich with experience that I kept finding new surprises and deeper insights. I hope you get as much out of this book as I have.

Glenn Cole

Joachim Bondo

Company: *Cocoa Stuff (one-man shop)*

Location: *Copenhagen, Denmark*

Former Life As a Developer: *I have 27 years of experience in starting up and running smaller software development companies and developing software using a wide range of programming languages, such as BASIC, COMAL 80, Pascal, C, C++, Objective-C, SQL, NewtonScript, PHP, JavaScript, and Bash. I've worked in many environments, such as THINK C and Think Class Library (TCL), Macintosh Programmer's Workshop (MPW), Metrowerks CodeWarrior and PowerPlant, 4th Dimension, Newton Toolkit (NTK), Sybase, MySQL, TextMate, Xcode, Cocoa, and Cocoa Touch. Platforms include Mac OS 3–8, Newton OS, Palm OS, Unix (FreeBSD and Mac OS X), Mac OS X Panther and Leopard, and iPhone OS.*

Life As an iPhone Developer: *I created Deep Green, a chess game, using the official iPhone SDK from Apple (since the day it was released).*

What's in This Chapter: *With the focus on creating a beautiful, elegant, and powerful user interface, and in a noncode language, the chapter covers the key areas that made Deep Green a successful application in the App Store, featured by Apple in several sections such as What's Hot and Staff Favorites.*

Key Technologies:

- *User interface design*
- *Simplicity*
- *Product statement*

Chapter 1

Simplify the User Interface for Complex Games: Chess, the Deep Green Way

On the day Deep Green was released, John Gruber of Daring Fireball quoted me for saying this:

> *When I compare the various iPhone chess apps (I bought them all), Deep Green offers pretty much the same functionality as the rest, and sometimes more, but with a fraction of the UI. Achieving this is why I'm four months later than the rest.*

Creating a successful application requires intelligent design from the very beginning, or you may walk down the wrong path. In this chapter, I'll share some of my design decisions that ended up making Deep Green a success—ten years ago as well as today.

More specifically, I'll cover simplicity from a user interface (UI) perspective more than dealing with the underlying code. In fact, you won't see a single line of code in this chapter.

Code is the foundation of your application, but even more important, your application will be judged by its user interface. If it fails to deliver a phenomenal user experience, your application won't likely be a success.

It's my belief that simplicity and success go hand in hand in general—and even more so in the context of iPhone application development.

Once Upon a Time . . .

When I bought my first Newton in 1996, I soon realized that a portable device like that was the perfect companion for playing chess. It was always at hand, always ready, and ever patient.

This was back when Apple didn't enjoy the mass adoption of its products and when developers were counted by the dozens, not the thousands. At the time, three years after the Newton was launched, there existed only one chess application. It was expensive, feature-poor, and slow. From a UI perspective, it was even unattractive.

In the summer of 1997, I made the first sketches for my own chess application. The year before, IBM's Deep Blue chess computer became the first machine to win a chess game against a reigning world champion (Garry Kasparov). With Newton's green enclosure and screen, the name for Deep Green was obvious.

In early 1998, I released the first public beta, as shown in Figure 1-1.

Figure 1-1. *Deep Green for the Newton*

Users were enthusiastic, praising Deep Green for its simple, yet powerful, implementation.

Here was a type of application, often implemented in a very complicated and clumsy way, made appealing and fun—maybe even like Apple would have done it. Moves were being animated, Newton OS–native gestures were deeply integrated, and the application looked great by the standards of the time. Games could be played back and forth and set up as desired (see Figure 1-2).

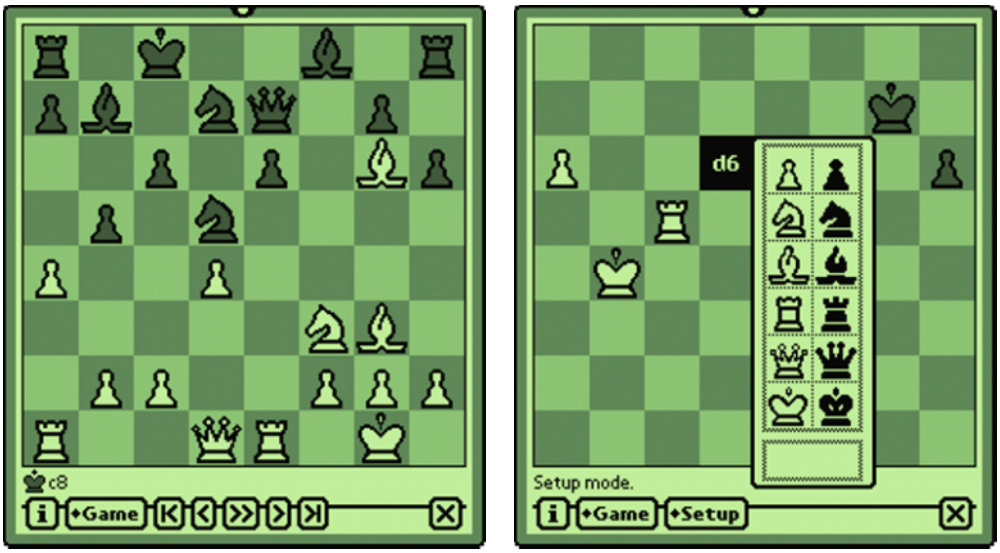

Figure 1-2. *Deep Green in Playback and Setup modes on the Newton*

In a matter of days thereafter, Steve Jobs terminated the Newton project altogether.

It didn't seem to have much impact on the interest in Deep Green. However, I knew it wouldn't last, and I probably wouldn't get to implement the many ideas I had for future versions.

The only real complaint I got from users was that Deep Green was simply too hard to beat, even at the lowest level. The Newton MessagePad 2000/2100 with its screaming 160 MHz StrongARM processor gave users an especially hard time. The engine was written in C++, and the user interface was written in NewtonScript. I had to implement what I ended up calling Concentration in order to have the engine choose among lesser moves at lower concentrations (see Figure 1-3).

Figure 1-3. *Deep Green's New Game "slip" on the Newton*

When Steve Jobs unveiled the iPhone ten years later at the Macworld conference of 2007, I knew right away I had to make Deep Green for my soon-to-be favorite handheld device. And this time, I reckoned, he wouldn't pull the carpet underneath my feet.

It took me a year of all my spare time, and a good bit of goodwill from my family, to finish Deep Green for the iPhone and iPod touch. The result, I think, is a fine testament to its predecessor, based on the same underlying design goal: *simplicity* (see Figure 1-4).

Figure 1-4. *Deep Green on the Newton (left) and on the iPhone (right)—a decade apart*

In the following sections, I'll talk exactly about that—simplicity. For reasons I'll explain herein, I believe simplicity is the most important feature you can add to your application. Read on.

Why Should We Care About Simplicity?

I'll be talking a lot about simplicity because simplicity was, and still is, the overall design goal of Deep Green. In the context of software development, I define simplicity as follows:

simplicity = beauty (how things look) + elegance (how things work)

That is, a high degree of simplicity requires a high degree of beauty and elegance. You can develop a beautiful application without it being elegant, and vice versa, but in order to create a truly *simplistic* application, you need to maximize and marry both beauty and elegance.

So, why is simplicity important, why should we developers care, and why do our users care? Well, we should care exactly for *that* reason: because our users care.

They may not know they do, however. In fact, I'll argue that most users think they want features. Features can be quantified, measured, and compared. And that's what users can express their need for. But underneath the desire for features is something bigger. *Control*. I believe what we all want as software users is the feeling of being in control. If we're in control, we'll be able to focus on what's important. We'll have more time with pure enjoyment. Simplicity is the means for creating the feeling of control.

A good example of the opposite, in my opinion, is Microsoft Word. It offers an amazing amount of features, out of which I'm guessing the average user may use 10 percent. You find yourself frustrated over and over, because you can't find the feature you want—it's buried in that 90 percent you don't use. Which few buttons in the countless numbers of toolbars are relevant to you at any given moment? It's very hard to tell. You often find yourself browsing around for the relevant functionality. You're out of control. You're having a bad user experience.

How Simplicity Was Achieved

In order to being able to develop a simplistic application, you need to find out what's important in your application. And to do this, you need to have a *product statement*. The product statement describes in one sentence what your product is. In Deep Green's case, the product statement is as follows:

A simplistic chess application for the casual player

The product statement is my guiding principle during the entire process from idea through design and development to release and maintenance. It's the beacon I'm relentlessly steering toward for every decision I make through the entire process. If I'm uncertain whether to include a feature or where to place it in the application, I go back to this statement and see whether and where the feature fits in.

If I lose track of the product statement, I lose control of the entire process, and Deep Green with it, and it's up to forces beyond my control what happens to my application.

With this in mind, I'll take you through some of the main areas of Deep Green and explain my motivations for doing what I did, which ultimately led to the application I'm so proud of (see Figure 1-5).

Figure 1-5. ***Deep Green's application icon***

Distilling the Essence

The keywords of the product statement are *chess*, *casual*, and *simplistic*. This leads to defining the primary and secondary, and possibly tertiary, functionality. The primary functionality should be supported by the most prominent user interface elements and be available to the user at all times, where the secondary functionality should be one gesture away. The tertiary functionality should probably not even make it to an iPhone application, but if so desired, it should be hidden away from the rest, and more important part, of the user interface. In Deep Green, I defined the primary functionality as follows:

- Displaying board and pieces
- Moving pieces
- Making a new game
- Taking back moves
- Showing the last move
- Getting a hint from the engine
- Seeing whose turn it is

This functionality is what chess boils down to when you play it casually. I wanted to create the sense of playing over the board with a friend, where Deep Green is the friend.

I defined the secondary functionality as follows:

- Setting up a position
- Playing back the game one move at a time
- Seeing captured pieces
- Swapping sides

In Deep Green all of this is one gesture away with a tap, a swipe or flick, or a multifinger gesture.

If you're successful in defining these areas and you keep them in mind when designing and developing your application, chances for a truly simplistic application are much higher than if you uncritically squeeze all functionality into the UI at once.

Pushing the Pixels

First, I have to dwell a bit on the graphics because that's what immediately meets the eye when launching Deep Green. Unfortunately, I'm not capable of producing graphics that satisfy my own preposterously high standards for looks and quality. Fortunately, others are, and I was lucky enough to get the chance to work with Japan's Mikio Inose on Deep Green's graphics. In him I met a person who would follow me to the very last pixel—and often even further.

Deep Green, as what I would call an application of high quality, will always be priced at the higher end. In order to give users value for money, this had to be reflected at every level of the application—from the quality of the underlying code all the way up to the "materials" used to depict the objects in the user interface, such as pieces and board.

Figure 1-6. *Deep Green's ebony and ivory pieces on noble wood*

In Deep Green, the pieces are supposed to be ebony and ivory, and the board is noble wood. If you look closely at the icon in Figure 1-5, you'll see the grains of the ebony. You can even see it in the smaller pieces on the board. The ivory pieces even feature the perhaps not-so-well-known Schreger lines that are used by connoisseurs to distinguish real ivory from fake (see Figure 1-6).

Do you notice the resemblance to the Newton version? The pieces are based on the same font, Adobe's Cheq—or, rather, its freeware derivative, Chess Regular by Alastair Scott. I wanted to continue on the same path of branding that I started ten years earlier.

The board is worn and has dents from years of use. Yes, even just playing casually can cause its wear and tear. The board can be flipped on its back, just like a real board, and on the backside you'll see the engine and other elements depending on the context (see Figure 1-7).

Notice the engravings on the main gear. It says "Cocoa Stuff, DENMARK." Above the visible gears there's an engraving in the wood saying "Manufactured by Cocoa Stuff." I'll go into even more detail with the engine in the "Making the User Smile" section.

Figure 1-7. *The backside of the board showing the engine and splash ornaments*

Letting the User Focus

Going back to the product statement, Deep Green is about playing chess. Nothing more, nothing less. It doesn't have the bells and whistles of more capable desktop applications. When you launch Deep Green, you are taken directly to the main view showing the board with pieces and the toolbar. It's very clean and pleasing and ready to play (see Figure 1-8).

I even contemplated sliding the toolbar out of the way, to make the experience even more secluded, but I figured the user would be more puzzled about not finding anything anywhere than relieved by the simplicity. Remember, I want my primary functionality exposed.

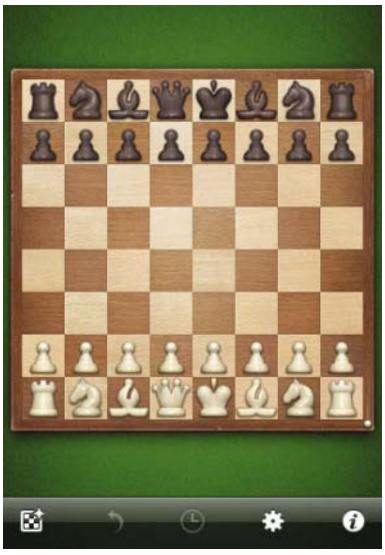

Figure 1-8. *Main view of Deep Green in Play mode*

Contrary to any of the other chess applications I have downloaded from the App Store, Deep Green lets you move pieces via both tap-tap and drag-and-drop gestures. Tapping a piece makes it wiggle like when moving the icons in the home screen. If you've started a tap-tap move, you can still move the piece or any other piece simply by starting to drag or by tapping it. You don't have to set a preference as I've seen in some apps. You just go ahead and do what's most natural. I tend to drag and drop for shorter distances and tap-tap for longer. It's not entirely trivial to implement (it's certainly not rocket science either), but this is something the user will be doing repeatedly. As a developer, you *have* to spend that extra time polishing that experience.

Since the user's finger will cover and obscure the destination square when moving the finger around on the board, I wanted to help identify both the piece that's currently being dragged and where it would go should the finger be lifted. I did that by enlarging the piece, offsetting it slightly upward, and making it semitransparent so that the board can be seen underneath. At the same time, I'm leaving a semitransparent copy of the dragged piece on its original square so that it's obvious what piece is currently being dragged and where it would go back to if released on an illegal square.

Finally, I'm displaying horizontal and vertical guides to give feedback on what square the finger is currently over. If the current square represents a legal move, the guides will highlight in green. They will highlight in red on an illegal move (see Figure 1-9).

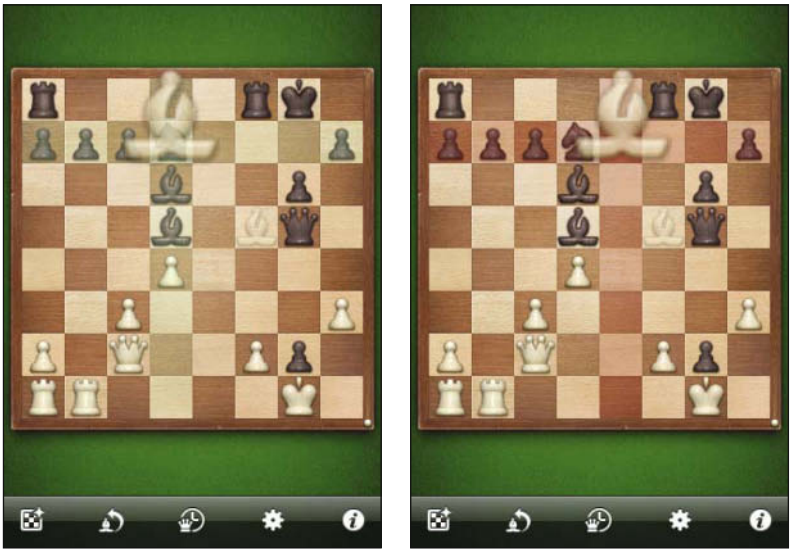

Figure 1-9. *Dragging a piece over a legal (left) and illegal (right) square*

Notice the second and third buttons from the left in the toolbar. They are for taking back your last move and for showing the last move, respectively. The buttons feature a miniature piece that denotes which type of piece the given move involves. In Figure 1-9, it shows that if you tap the Take Back button, you'll take back your last move, which was a bishop move.

Tapping the Show Last button—Deep Green's last move in this case—will show a queen's move. Just by looking at the buttons, you may be able to figure out what will happen if you take back a move or what Deep Green's last move was. By incorporating a primary feature this way, I can save an ugly move list somewhere. Keep in mind, I'm targeting the casual player, not the type of player who wants to look at a list of algebraic move notations.

The fourth button in the toolbar is the Engine button. When it's Deep Green's turn and the engine is working, the gear spins. When it's your turn, tapping it will make it suggest a move (while also spinning because the engine is hard at work).

Drilling Down

There are *application settings*, and there are *game settings*. Application settings are rarely changed, probably only once, and should be maintained via the general Settings application. In Deep Green these are settings such as how to notify the player about events such as check, checkmate, and stalemate (they could be via callouts, sounds, and vibration), whether to display coordinates on the board, whether to prevent the iPhone from going to sleep, and so on, as shown in Figure 1-10.

Figure 1-10. *Deep Green application settings displayed in the Settings application*

Game settings, on the other hand, will be changed much more often, perhaps as often as each game. As a secondary feature, these settings are only a tap away—and within the application.

The screen real estate on the iPhone is very limited compared to desktop and laptop computers, and Apple designed the iPhone user interface with this in mind. It allows you to drill down to the information you seek very easily by initially being presented with the high-level information, allowing you to tap to the lower-level information. This is how the Settings application works, and I've implemented the Game Info view the very same way. Initially, it shows the high-level information such as the players and captured pieces (see Figure 1-11), and from there you can drill down to tertiary information such as game state, current castling availability, full-move number, half-move clock, and so on.

I chose to adopt the same implementation as that of the Settings application so that the user would feel at home with the user interface right away, even to the extent that it looks exactly like the table views in Settings. Unfortunately, these elements are not available in Apple's Cocoa Touch framework, so I had to reverse-engineer them to pixel perfection.