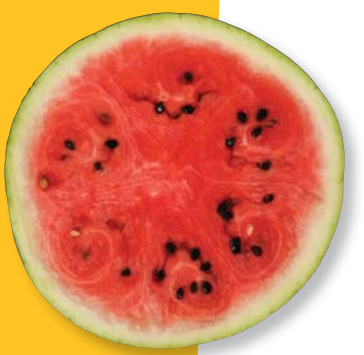
iPhone Games Projects

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.

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.

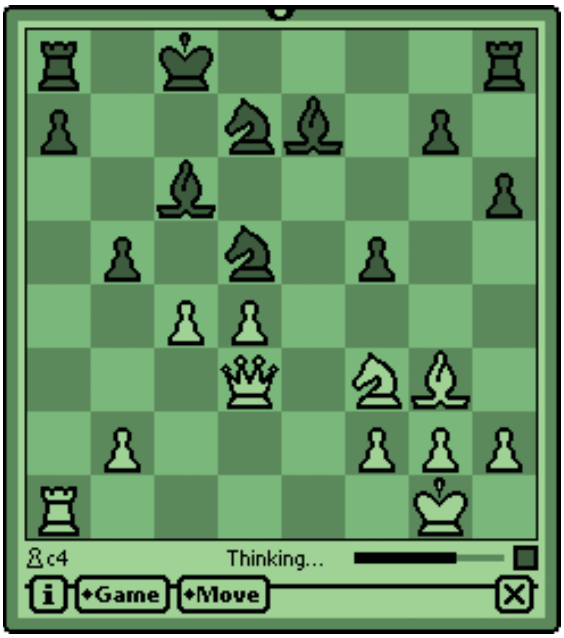


Figure 1-1. *Deep Green for the Newton*

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).

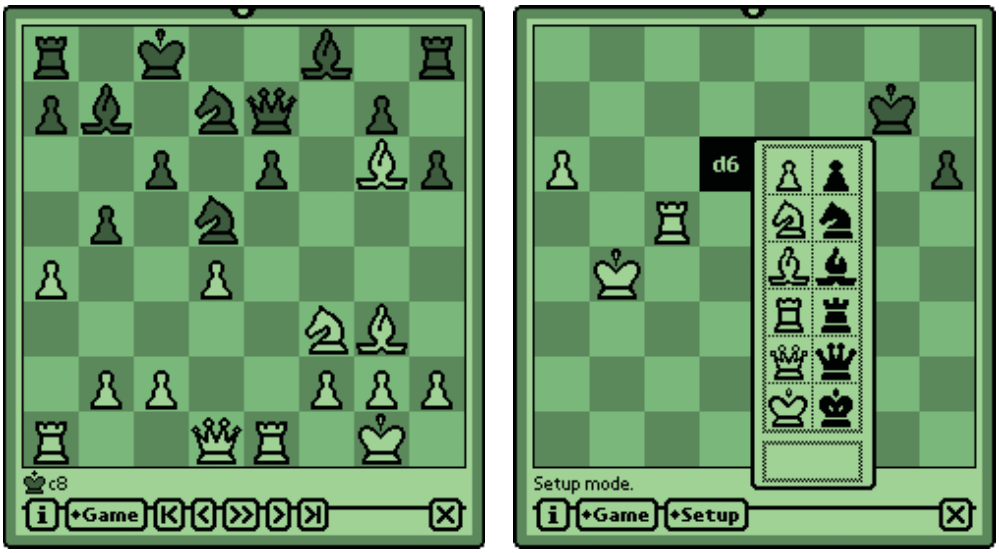


Figure 1-2. *Deep Green in Playback and Setup modes on the Newton*

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).

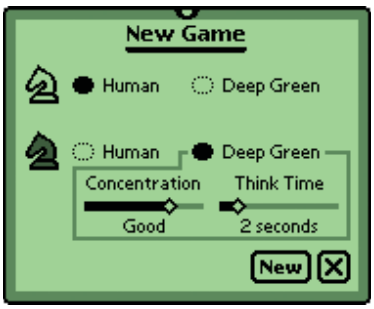


Figure 1-3. *Deep Green's New Game "slip" on the Newton*

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.

My point in all this is to say that this way of hierarchically structuring views and information is another way of achieving simplicity. Instead of presenting the user with everything at once, I'm simplifying or summarizing the information in such a way that the user controls what areas he or she wants to explore, and only then do I present that particular subset of information.

Figure 1-11. *The highest-level Game Info view (left) showing secondary information and Game State (right) showing tertiary information*

Less Is More

My goal with the initial version of Deep Green for the iPhone was to at least match the feature set of the Newton version. In that, I had three game modes: *Play mode*, *Setup mode*, and *Playback mode*. These modes would offer the functionality to play a game, set up a given position (taken from the newspaper or a chess book, for example), and step back and forth in the current game.

The solution for setting up a position that many chess applications for the iPhone seem to implement is to squeeze in another view, in an already very tight view, that displays all the pieces and little arrows or widgets to specify the game state such as castling availability, en passant square, color to move, and other information needed to fully specify a legal game position (see Figure 1-12).

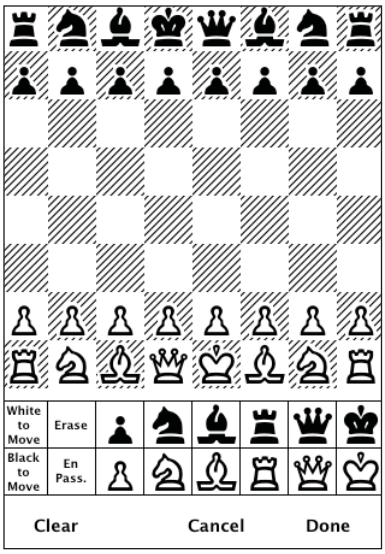


Figure 1-12. *A constructed example of the easy solution*

Figure 1-13. *Deep Green with the simplistic solution*

Although this is a very straightforward, and possibly even intuitive, solution, it's not very user friendly. Even though everything is there, it's up to the user to figure out what it all means. There are pieces everywhere, and it's all very confusing. It's not clear what to do and how to do it.

The solution I ended up implementing in Deep Green was to introduce the *drawer*. The drawer resides under the board and slides out when needed. When switching to Setup mode, it slides out with the different pieces, as shown in Figure 1-13.

What this solution perhaps suffers from in straightforwardness, it more than compensates for in simplicity and power. When seeing the drawer for the first time, the user will invariably ask herself, where are the white pieces? But how long does it take her to guess it's under the white color switch button? My guess is, not very long.

By stimulating the user's curiosity, I'm not only engaging her, but I'm also allowing myself to get rid of a bunch of clutter. The user doesn't need to drag both white and black pieces from the drawer onto the board simultaneously.

And here's the interesting part; by hiding the UI, I'm making it even more powerful. In this context I've done that by allowing the user to double-tap a piece to toggle its color. So if you have a black knight on the board and you want it to be white, simply double-tap it. So, you could easily place all the pieces in just one color and then double-tap the pieces that need to be the opposite color. If you double-tap a piece in the drawer, all drawer pieces swap color (just like tapping the color switch button), as shown in Figure 1-14.

The power is in what's *not* there, rather than in what *is* there. That's my simplistic solution.

My solution for setting the game state is to reuse the UI that's already there in the form of the Game Info view under the *i*-button (shown earlier in Figure 1-11). Once again, I've eliminated unneeded UI by implementing a design that can be used for both displaying and setting the game state and information while both playing the game and setting up a given position.

Figure 1-14. *Drawer pieces in the process of toggling from white to black in a chain reaction type of animation*

NOTE

The ultimately simplistic way of setting up a position is of course the FEN string, which, by the way, is one of the options Deep Green offers in its internal API. The FEN string lets you specify the position by listing the pieces on each rank and appending the game state. The position shown in Figure 1-14 would be `r4rk1/pppn3p/3b2p1/3b1Bq1/3P4/2P4P/P1Q2Pp1/RR4K1 w - - 0 20`. Using this method in the user interface on the iPhone would perhaps be challenging the user too much, I think.

Empowering the User

The last of the three game modes, besides Play mode and Setup mode, is Playback mode. This is where you can step back and forth in the game and resume playing from any position.

Here, also, I've taken a somewhat different approach than the other chess applications currently available for the iPhone and iPod touch. Where these applications don't seem to offer an explicit mode, they let you rather implicitly undo and redo moves. On one side, this is very straightforward, but on the other, it's rather constraining.

A big part of learning chess, besides simply playing it, of course, is studying others' games. As a user, you'd want to step through a game in order to learn from it—without taking back moves. There's a big difference between *taking* back and forth moves and *stepping* back and forth. Where taking back and forth modifies the game, stepping doesn't.

I took this even further in Deep Green by letting the user single-step back and forth, jump to the beginning and to the end of the game, and *auto-replay* a game by animating the moves in sequence—just like playing a movie for the user to sit back and enjoy. All these operations animate the pieces smoothly (see Figure 1-15).

The slider in the familiar drawer even lets you *scrub* back and forth with instant feedback. So when you drag the slider, the position on the board updates itself immediately to reflect the given position. This way you can quickly jump to a certain stage in the game and fine-tune the position on the back and previous buttons.

I find myself doing this all the time, and the feedback is extremely rewarding. You feel in total control, and you get a sense of a very powerful device and application because Deep Green reacts immediately.

Figure 1-15. *Playback mode with an intuitive user interface*

To summarize, playing back a game belongs to the group of secondary functions and shouldn't be confused with the primary function of undoing a move, for which there is a button in the toolbar in Play mode. And because it's a secondary function, it's one gesture away.

The easiest way of entering Playback mode is to flick right with your finger below the board where the drawer will come out. In the built-in Photos application, flicking right on a photo animates to the previous photo—it goes back in time. Going to Playback mode in Deep Green is also like going back in time because it lets you explore the historic moves.

Left-flicking in Photos goes forward in time by showing the next photo. And so does doing it in Deep Green, because a left-flick takes you to Setup mode, where you can set up a position to play next.

Making the User Smile

Like a real-life chessboard, and so many other things in real life, the chessboard in Deep Green has a backside. When making a horizontal flick from one of the board edges toward the center, the board flips around on its back in the direction of the flick (see Figure 1-16).

When in Play mode, the backside shows the captured pieces and a representation of the *chess engine*. The board backside with the engine is really just a gimmick. It doesn't have much of a function. At best, it duplicates existing functionality from the Game Info view in that it shows the captured pieces.

UI that doesn't have a function can be categorized as bloat or clutter, and that's what I've been advocating against in the whole chapter. So, why on Earth did I choose to spend half the graphics designer budget on bloat? I don't know either. Or, that's not entirely true. I do know, because I spent a lot of time and effort thinking about it before asking the designer to make the engine for me.

Figure 1-16. ***Board backside with captured pieces. The graphics file for the engine consisted of about 600 layers, and the gears can actually turn.***

One of the many things I like about the Macintosh platform is the odd little things the creative developers are adding to their software in order to surprise us and make us smile—just some day, perhaps after months of use. If you haven't already, you'd be surprised what can happen when clicking around in your installed applications while holding the Alt key down, for example.

I wanted the same for Deep Green, because I like these hidden features myself, especially if they're useful in some way or another. And although you can see the captured pieces in the Game Info view, I personally prefer to quickly flip the board around and then flip it back to continue the game. Again, this is a secondary function, one gesture away, that I don't want to spoil the aesthetics with in the main layout.

So, besides being semi-useful, it also makes sense that you can actually touch and affect the board. I mean, you can with the pieces.

The design of the mechanical engine, which obviously doesn't have anything to do with the artificial intelligence of Deep Green's chess engine, was constructed in a 3D application and rendered into bitmaps, resulting in a Photoshop file with almost 600 layers of gears, shadows, and other effects. From there it was pixel-perfected by hand. As a result of its 3D model origins, the gears can actually turn—and they do. Try flipping the board while Deep Green is "thinking." The gears will animate. While having a Deep Green vs. Deep Green game, you can flip the board around and watch the gears spin, and the captured pieces appear as the game progresses.

I hope it makes you smile when you encounter this. It sure makes me.

Summary

These were my words on how I developed a successful application, ten years ago as well as today. I'm sure this also explains at least part of the reason why Deep Green has been featured on the App Store in several sections such as What's Hot and Staff Favorites. To me, simplicity is the single most important feature you can add to your application. It's the prerequisite for delivering a superior user experience.

I hope I've inspired you to dare leave out features that are secondary and tertiary to your product statement and that spending a large amount of time, and even money, on the user interface is a clever investment.