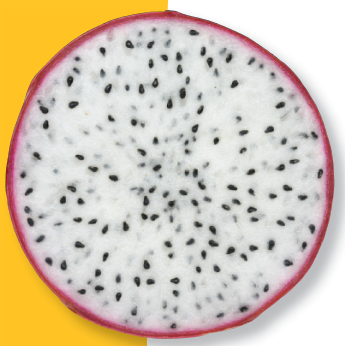
iPhone Cool Projects

DAVE MARK, SERIES EDITOR

GARY BENNETT
WOLFGANG ANTE
MIKE ASH
BENJAMIN JACKSON
NEIL MIX
STEVEN PETERSON
MATTHEW "CANIS" ROSENFELD



Apress®

iPhone Cool Projects

ISBN-13 (pbk): 978-1-4302-2357-3

ISBN-13 (electronic): 978-1-4302-2358-0

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

Lead Editor: Clay Andres
Development Editor: Douglas Pundick
Technical Reviewers: Glenn Cole, Gary Bennett
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Copy Editor: Heather Lang
Associate Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Dina Quan
Proofreader: April Eddy
Indexer: BIM Indexing & Proofreading Services
Artist: April Milne
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.

This book is dedicated to my wife, children, and friends who never stop believing in me. Also, I want to thank my friend Greg Stanley, who taught me the importance of thinking positive and being humble. This, I will always be grateful for.
—Gary Bennett

To my grandfather, Bernard Cohn, who handed down my first Apple computer.
—Benjamin Jackson

Neil Mix would like to thank his wife, Sarah, and his son, Henry, for encouraging him to take the time and energy to build Pandora Radio and contribute to this book. He would also like to thank his teammates at Pandora for being the best engineering team, period. Without them, Pandora Radio would not have been possible.
—Neil Mix

For my parents Pam & John: thanks for being my biggest fans.
—Steven Peterson

With love and thanks to Scary
—Matthew "Canis" Rosenfeld

Contents at a Glance

About the Lead Author xi
About the Technical Consultant xiii
Acknowledgments xv
Introduction xvii

WOLFGANG ANTE

CHAPTER 1 Designing a Simple, Frenzic-Style Puzzle Game 3

MIKE ASH

CHAPTER 2 Mike Ash's Deep Dive Into Peer-to-Peer Networking 29

GARY BENNETT

CHAPTER 3 Doing Several Things at Once: Performance Enhancements with Threading 57

MATTHEW "CANIS" ROSENFELD

CHAPTER 4 All Fingers and Thumbs: Multitouch Interface Design and Implementation 81

BENJAMIN JACKSON

CHAPTER 5 Physics, Sprites, and Animation with the cocos2d-iPhone Framework 107

NEIL MIX

CHAPTER 6 Serious Streaming Audio the Pandora Radio Way 133

STEVEN PETERSON

CHAPTER 7 Going the Routesy Way with Core Location, XML, and SQLite 157

INDEX 203

Contents

About the Lead Author xi
About the Technical Consultant xiii
Acknowledgments xv
Introduction xvii

WOLFGANG ANTE

CHAPTER 1 **Designing a Simple, Frenzic-Style Puzzle Game 3**

Creating Frenzic 3
Introducing Formic 5
Exploring the Formic Code 6
Setting Up the Project 8
Coding the Game Object 10
Coding the View Controller 18
Coding the Background View 21
Adding iPhone-Specific Functionality 22
Summary 25

MIKE ASH

CHAPTER 2 **Mike Ash's Deep Dive Into Peer-to-Peer Networking 29**

Planning a Simple Collaborative Game 30
Building the GUI 30
Networking the Game 35
Defining the Networking Goals 35
Designing the Network Code 36
Understanding Endianness 40
Coding the Networking 41
Integrating Networking and the GUI 50
Summary 53

GARY BENNETT

CHAPTER 3 **Doing Several Things at Once: Performance Enhancements with Threading** **57**

Beginning to Write Threading Applications 59
Knowing When to Thread 59
Understanding Threading Basics 61
Avoiding Threading Pitfalls 63
Writing the Thread the Needle Application 65
Building Our Application 65
Creating a Thread 72
Implementing a Critical Section 76
Stopping Multiple Threads at Once 77
Summary 77

MATTHEW "CANIS" ROSENFELD

CHAPTER 4 **All Fingers and Thumbs: Multitouch Interface Design and Implementation** **81**

Looking at the iPhone's Capabilities 82
Designing for Multitouch 84
Exploring the Multitouch API 87
Handling Events 87
Recognizing Gestures 89
Implementing Multitouch Controls 92
Handling Touches 94
Deciding What Movement Means 97
Applying the Movement 99
Applying Weight and Inertia 100
Tying Up Loose Ends 102
Summary 103

BENJAMIN JACKSON

CHAPTER 5 **Physics, Sprites, and Animation with the cocos2d-iPhone Framework** **107**

Getting Started with Game Programming 108
Introducing OpenGL ES 109
Introducing cocos2d and Chipmunk 109

Developing Arcade Hockey 109
Tracking the User's Finger 112
Detecting Collisions 114
Simulating 3D Lighting in 2D Space 118
Creating a Simple Application 119
Setting Up the Xcode Project 119
Setting the Scene 121
Creating the Game Layer 122
Summary 129

NEIL MIX

CHAPTER 6 **Serious Streaming Audio the Pandora Radio Way 133**

Choosing to Develop for the iPhone 133
Introducing Pandora Radio's Technology 134
Grasping the Basics of Audio Development 134
Managing Complexity 136
Outlining Our Sample Application 136
Streaming Audio 137
Keeping Your Code Format Agnostic 138
Using Envelopes and Encoding 138
Designing Our Sample Application 139
Implementing the Player 141
AudioSession 142
AudioRequest 143
AudioFileStream 145
AudioQueue 147
AudioPlayer 147
Ending with a New Journey 148
Falling Behind in a Slow Network 148
Dropped Connections 150
Minimizing Gaps Between Songs 151
Resuming a Song 151
Improving Application Responsiveness 151
Finding Help Resources 152
Testing: Saving the Best for Last 152
Summary 153

STEVEN PETERSON

CHAPTER 7 **Going the Routesy Way with Core Location, XML, and SQLite** **157**

Starting from Scratch 158
Assessing the Application Requirements 158
Creating the Routesy User Interface and Classes 160
Bringing Real-Time Predictions to Routesy 179
Adding Location-Based Information to Routesy 191
Putting the Finishing Touches on Routesy BART 195
Summary 200

INDEX **203**

About the Lead Author

Gary Bennett is the lead author on this project. He served for 10 years as a nuclear power engineer on two different nuclear powered submarines. On shore duty, Gary completed his Bachelor of Science degree in computer science.

After college, he worked for GTE Data Services and Arizona Public Service converting hundreds of thousands of lines of OS/2 code to Windows NT. Gary then worked for several technology and health care companies developing Windows NT and Linux applications, including satellite communications. After that, Gary was chief information officer of a young health care company that successfully completed an IPO.

In 2007, Gary started his own technology company, xcelMe.com, focusing on Mac and iPhone development. In 2008, xcelMe.com was hired to develop leading ski and snow report iPhone applications. Since 2008, Gary has been dedicated to teaching others iPhone development. xcelMe.com has developed online iPhone development and marketing courses affordable to all. Gary has taught hundreds of students iPhone development online throughout the world. Gary continues to release helpful iPhone development YouTube videos that benefit the iPhone development community.

In 2009, he worked with EA Sports at their Tiburon studios in Orlando, Florida, where he launched his third iPhone App, Tee Shot Live. He is currently working for a financial institution developing an online banking iPhone app.

About the Technical Consultant

Glenn Cole was the technical consultant on this book. He has been a professional software developer for nearly three decades, from COBOL and IMAGE on the HP 3000 to Java, Perl, shell scripts, and Oracle on the HP 9000. He is a 2003 alumnus of the Cocoa Bootcamp at the Big Nerd Ranch. In his spare time, he enjoys road trips and furthering his technical skills.

Acknowledgments

This book is a compilation of a lot of great work by some really smart authors. They have focused and contributed their chapters based on areas of their expertise. You get to benefit from the years of their expertise; enjoy it!

I am so impressed by the fine people at Apress. I believe their books are the finest on the market. Additionally, they are great to work with. I have made many friends.

I would like to thank "Admiral" Clay Andres whose vision and ability to put together a talented team made this great book possible. He is actually not an Admiral, but should be. Our copy editor, Heather Lang, and development editor, Douglas Pundick, were so very helpful in making sure the quality of the book was what you would want. Special thanks to Laura Esterman and Dina Quan for managing the book production process when it needed it the most.

Lastly I would like to thank Michelle Lowman for connecting Clay Andres and myself and giving me the privilege to be part of this great project.

Gary Bennett

I would like to thank Ivan Neto, Benjamin Maslen, and Rafael Cruz for their hard work on Arcade Hockey, and my parents Lillian Cohn and Larry Jackson for their nonstop love and support.

Benjamin Jackson

Introduction

You are going to love this book! I know I do, and I had to read every word of it and check every line of code, twice!!

If you're like me, you've registered as an iPhone developer with Apple, read some documentation, and sought help in taking the next bold step. Perhaps you've picked up "Beginning iPhone Development," dutifully working through all of the projects, and you understood most of it. If not, I heartily recommend it. The book is great because it gently guides you through many of the technologies that make up an iPhone application. Make no mistake; the book covers a lot of ground. But the projects are kept relatively simple to keep the lessons focused.

First step taken, now boldly onward into the fray!

This book picks up where "Beginning iPhone Development" leaves off. The projects herein were developed specifically for this book, but these are no lightweight applications. Some projects are based on shipping products, showing how various technologies are integrated into a cohesive application. Other projects cover difficult topics and thus are more focused.

The projects illustrate advanced topics such as game timers, XML parsing, streaming audio, multithreading, recognizing advanced gestures, and even designing your own network protocol using UDP (and why you would want to do this). You'll be discussing mutexes, race conditions, sockets, packets, and endianness in no time!

Those who want to develop immersive games have long heard that using a game engine is important, but getting started has been a challenge. Here at last is a game that is built around the open source cocos2d game engine, explained in great detail.

All the chapters represent the personal experience of successful developers; they are written by the developers whose skills we admire and respect.

In short, your next steps are clearly laid out for you.

Who this book is for

This book is for all iPhone and iPod touch developers who want to know more so that they can tackle more difficult programming tasks on their way to creating the next great app. Perhaps you have completed an introductory book such as "Beginning iPhone Development"

by Dave Mark and Jeff LaMarche, or possibly you have already completed a simple app and you're ready for the next step on your journey.

It also helps to be comfortable with Cocoa Touch, basic Xcode tools, and Objective-C. You can pick up extra help from "Learn C on the Mac" by Dave Mark, "Expert C Programming" by Peter van der Linden, and "Learn Objective-C for the Mac" by Mark Dalrymple and Scott Knaster.

Mostly, this book is for anyone who wants to write better apps for iPhone and iPod touch and is willing to put in a little time to learn from some of those who have already succeeded at it.

What's in the book

We open with Wolfgang Ante, the developer behind the Frenzic puzzle game, showing how the game was developed and guiding us through the process of creating a similar game called Formic. Timers, animation, and intelligence are used to make the play engaging. If you have been wanting to write a game but have had difficulty getting started, this chapter will provide the guidance and inspiration you need!

Chapter 2 finds Rogue Amoeba's Mike Ash explaining how to design a network protocol using UDP, and demonstrating its use in a peer-to-peer application. This topic is not for the faint of heart, but Mike explains it in a way that makes sense to us mere mortals. I had never seen this topic covered before, so I'm thrilled to see it here.

Next up with Chapter 3 is Gary Bennett covering the daunting but important task of multithreading. The CPUs in the iPhone and iPod touch won't be mistaken for those of the Mac Pro, but they pack enough power that frequently they are waiting for something to do. Multithreading can be used to keep the user interface responsive while working on other tasks in the background. Gary demonstrates how to do this, and highlights traps to avoid along the way.

In Chapter 4, Canis Lupus (a.k.a. Matthew Rosenfeld) describes the development of the Keynote-controlling application Stage Hand, how the user interface evolved, and the lessons learned from that experience. This knowledge is then demonstrated in a project showing how to recognize many complex gestures at once, including flicking (with inertia!) and rotating an object. Remote controls should all be this handy.

Benjamin Jackson introduces us to two open source libraries in Chapter 5: cocos2d for 2D gaming, and Chipmunk for rigid body physics (think "collisions"). He describes the development of Arcade Hockey, an air hockey game, and explains some of the code used for this. Benjamin then guides us through the creation of a miniature golf game. It's definitely helpful to have such clear guidance through these very murky waters.

Processing streaming audio seems like yet another black art. Luckily for us, Neil Mix of Pandora Radio reveals the science behind the magic in Chapter 6. How do you debug what you can't see? Neil guides us through the toughest challenges, sharing his experience of what works and what to watch for. Audio is hard; I'm thankful to have such a difficult topic explained so clearly. Some of the techniques shown can be used for non-audio applications as well.

The book concludes with Steven Peterson demonstrating a more prosaic integration of iPhone technologies. He weaves Core Location, networking, XML, XPath, and SQLite into a solid and very useful application. Games are great fun, but this is the type of application that makes the device so compelling for the non-gamer. You've seen some of the pieces before; now you'll see how to put them all together.

Software development can be hard. Introductory books lay the foundation, but it can be challenging to understand where to go next. This book shows how to integrate the pieces into a complete application. In addition, many of the topics covered here are notoriously difficult. You'll want to read the chapters more than once, then keep them handy for reference.

Working through the chapters was great fun, and I learned a tremendous amount. I'm sure you will as well!

Glenn Cole

Wolfgang Ante

***Company:** ARTIS Software*

***Location:** Vienna, Austria*

***Former life as a developer:** Macintosh Software Developer since 1994. Received the* **Macworld** *Editor's Choice Award (1999) and MacUser Award 2004, both for Best Graphics Utility.*

***Life as an iPhone Developer:** Built the Frenzic puzzle game with Xcode and Interface Builder*

***What's in this chapter:** After providing some insight into the development of Frenzic, this chapter discusses a similar game called Formic that shows the basic techniques behind the game logic and animations of a puzzle game.*

Key technologies

- *Using* `UIView` *animations for visual feedback*
- *Using* `NSTimers` *to keep the game running*
- *Using* `NSUserDefaults` *to save and restore games*

Chapter 1

Designing a Simple, Frenzic-Style Puzzle Game

This chapter is about Frenzic, a popular puzzle game created by ARTIS Software and the Iconfactory. We'll begin by telling you the story behind Frenzic and discussing the design process and some things learned while we developed the game. Finally, we'll guide you through creating a game called Formic, which will demonstrate some of the concepts used in Frenzic.

NOTE

If you do not know Frenzic, head over to `http://frenzic.com` to download it and see about it for yourself. The version for the iPhone will cost you $2.99, but a version for the Mac that you can download and try for free is also available.

Creating Frenzic

First, let's talk a bit about its history. Frenzic is quite old, I have to confess. I had the basic idea for Frenzic about 18 years ago, while watching a cheesy game show similar to *Wheel of Fortune*. The show involved spinning a big wheel with a ball inside that landed on money values for contestants to win prizes—something clicked, and the basic idea for Leblon (the original name of Frenzic) was born. Initially, there were no power-ups and purely random pies. The game evolved, was ported over to several computer platforms, and got a bit better on every step of the way. You can see its current incarnation in Figure 1-1.

There were two major milestones in advancing the game play: ideal games and power-ups.

In early versions of the game, players felt that, late in the game, they would get unfair pies that they could not set. The pies were chosen randomly, so even if they played a perfect game, players could get pies that made them lose lives. Wolfgang Sykora had the idea to let the application itself play an ideal game in the background, with the same pies you get. An 'ideal game' means clearing circles as soon as possible. Based on this ideal game, players would never be given a pie that could not be set. This made a huge difference! If players try to clear pies as soon as possible and don't make mistakes, they can now possibly play forever if they are fast enough (though, at times, players may decide instead to take risks by filling circles with pie pieces of a single color to potentially win a life).

Figure 1-1. *The game screen of Frenzic*

The second big improvement to game play came when I showed the game to Gedeon Maheux of Iconfactory. He invented the three power-ups that further improved the strategy of the game. Now, players can take even more risks by filling pies with pieces of a single color in one of the three dedicated power-up circles. Activating the power-ups later allows players to keep the game going even longer and play even faster.

Apart from the game play innovations, several other things have been crucial to the success of Frenzic, the biggest one is my partnership with Iconfactory. Most of the time ARTIS Software is just me, though my wife Arta helps me a lot with testing (she will break, in record time, any code that is not ready). Apart from this, I am a single developer working from my home office, which I love, but being truly successful would require me to be good at all the things that make up great software. Most people, and that includes me, will not be able to do everything well alone. So finding someone who would complement my skills was very important. I have been very lucky to find these partners in Iconfactory. They are some of the best designers in the field of icon and user interface design, and it's an honor to work with them. While I did all the programming on Frenzic, Gedeon Maheux designed the user interface, and David Lanham created the beautiful artwork. The extensive web site was crafted by Anthony Piraino, while Craig Hockenberry and I wrote the code behind it so it works even under heavy load. Last, but not least, Dave Brasgalla did the wonderful music and sound effects.

The web site is a very important part of Frenzic—it may be the most comprehensive high-score list ever created. It also includes player cards that can be customized, player statistics, comments, and different ways to compete: against time (called devotion), against friends, or locally (using the GPS location from the phone). At the time of this writing, more than one million scores are recorded on the Frenzic server. Access to the global high-score tables is possible from the web site as well as from inside the application (see Figure 1-2), so we had to implement web services to communicate with the application and secure the submission of scores to the server to prevent script kiddies from cheating. The whole high-score system amounted to about half of the work that went into Frenzic.

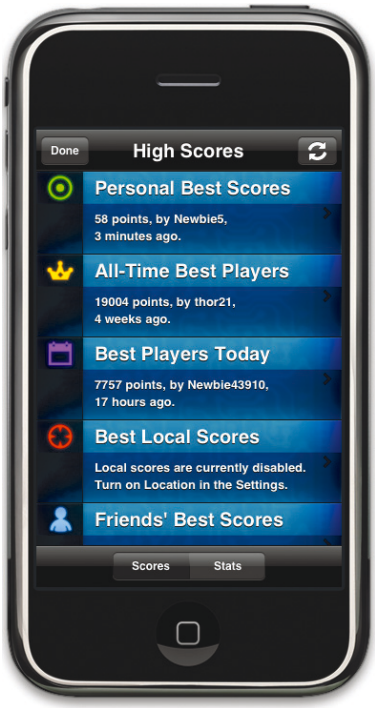


Figure 1-2. *The high-score screen of Frenzic*

Introducing Formic

In this chapter, I want to show you a few of the things Frenzic does. For that, I have created a slimmed-down game called Formic, shown in Figure 1-3. Instead of just showing you some snippets from Frenzic's code, I want to show you a complete game that you can compile, run, and even modify. In the following sections, I will explain the game logic and game graphics in more detail, but I assume some basic knowledge of Xcode and Cocoa.

Like Frenzic, Formic has a middle circle where you will get pieces that you can move to the surrounding circles by tapping on them. The pieces have distinctive shapes. If the center circle's shape matches a surrounding circle's shape, you can move the center piece to the outer circle, and both pieces will be moved out and replaced by new ones. Pieces also have a color, and when you bring together pieces of same shape and color, you win a point. The time to decide where to move a piece is limited and gets shorter the longer you play. If you cannot place a piece in the given time, you lose one of your five lives. The game is over when you have lost all your lives.

Formic is a great project to demonstrate a few things. This very simple and complete game is somewhat similar to Frenzic, but not as much fun. It lacks sound, but it is fully animated and persistent (when a phone call comes in, or you simply quit it by pressing the home button, the game will remember its state and offer to continue the game where you left it on the next launch).

Figure 1-3. *The game screen of Formic*

NOTE

The complete source code of Formic is included on this book's Source Code page of the Apress web site. I have tried to keep it extremely compact and still contain a complete game. There are a few things missing, like sound, but overall, it is a complete game.

Exploring the Formic Code

Formic uses pure and simple Cocoa Touch. It uses `NSTimer` for scheduling and `UIView` animations for its graphic effects, just like Frenzic. If you want to write a graphic-intense game, you should probably take a look at OpenGL ES, but for simple puzzle games that just move around a few pieces, this approach is the way to go in my opinion. Nonetheless, keep in mind that Core Animation was built for simple, single animations: it is optimized for ease of use, not for performance. If you decide to use `UIView` animations or Core Animation, be sure to write some test code that simulates the most demanding animation your game will probably face, and don't forget to also play sound. Don't wait and add sound at the end, as

playing music and sound effects on the iPhone does consume noticeable amounts of processing power. Playing sounds has to be part of the simulation.

It also uses the classic Model View Controller (MVC) pattern in a loose way, where the model would be the game object. Figure 1-4 shows a basic MVC pattern.

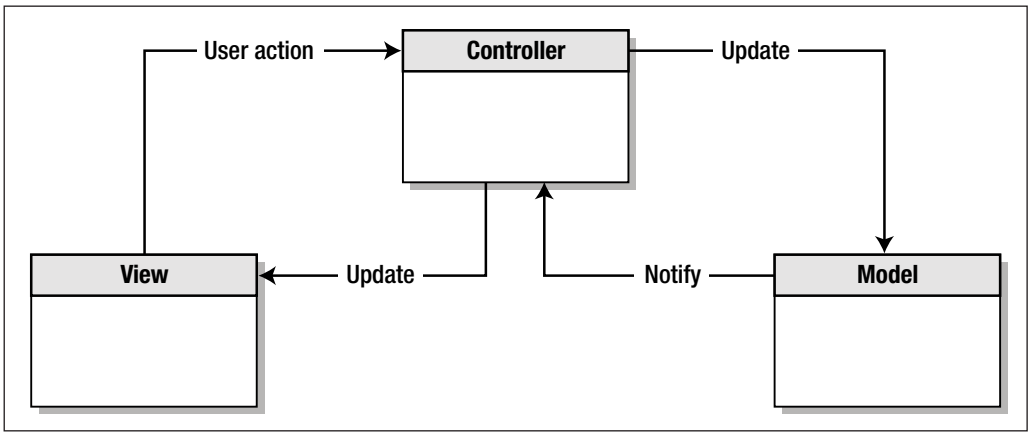


Figure 1-4. *The classic Cocoa MVC flowchart*

The views themselves are quite dumb: they just know how to display themselves. Most of them are simple `UIImageViews`, with one exception—the background view draws the circles and knows about their positions. Therefore, it also accepts the taps and translates the coordinates back into the tapped circles. This input is then sent directly to the game object, bypassing the controller. The main view controller is responsible for keeping all the views together and animating them. The game logic is isolated in a model object; it keeps the game running and talks to the view controller to make the state of the game visible. This layout leads to the updated flowchart for Formic's objects shown in Figure 1-5.

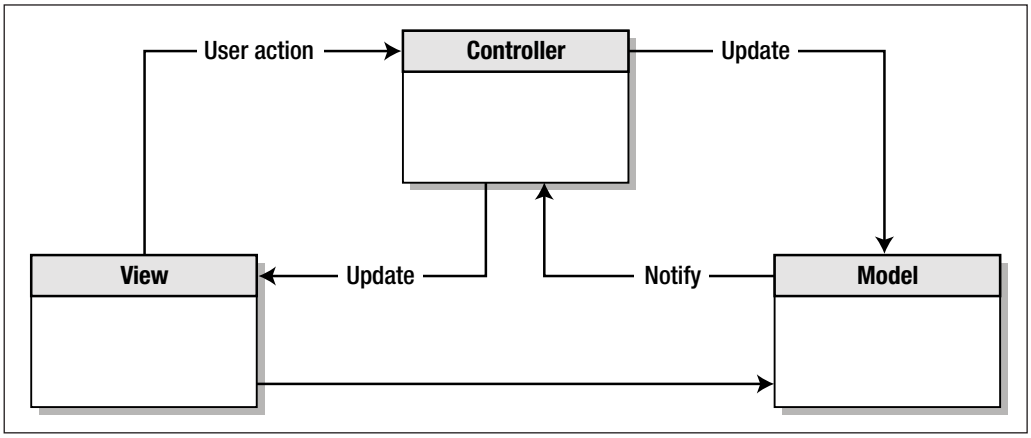


Figure 1-5. *The MVC flowchart for Formic*

You should always try to keep the game logic and graphics separate, though it is sometimes difficult to keep them 100 percent apart from each other. But keeping these functionalities in different objects will make it easier to adapt and fine-tune the game, which is something that will take a lot of the total development time of your game. Good games are not created on the drawing board; you have to play them to see what's great and what's not, and alter accordingly.

In the following sections, you will learn to create Formic. We'll starting from an empty project and create the game object that contains all the game logic, the view controller that keeps all the views together and animates them. Finally, we'll create the custom view that sits in the background of all the views, accepts the player's taps, and converts them into logical taps for the circles that are directly fed into the game object.

Setting Up the Project

Before starting to write code, you need to set up a project. From Xcode's **File** menu, choose **New Project**, and chose **View-Based Application**, as shown in Figure 1-6.

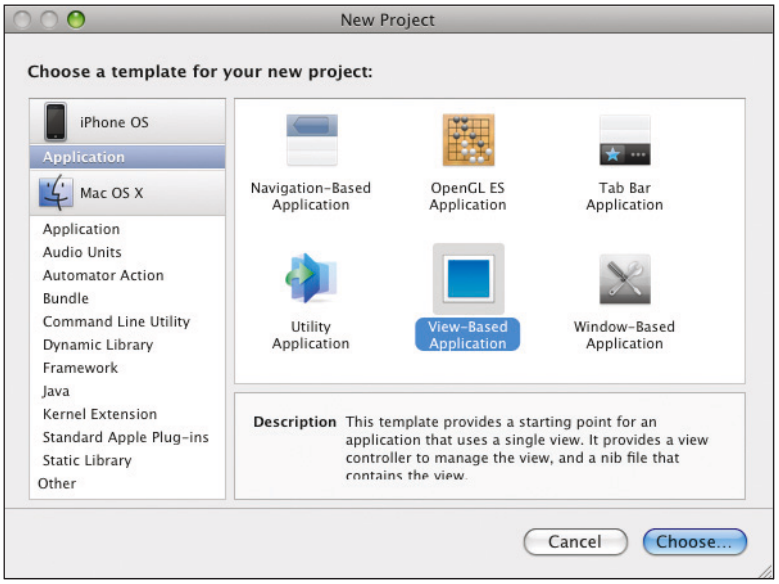


Figure 1-6. *The New Project dialog in Xcode*

This will create a basic project that has a lot of things already set up for you. This simple command created the complete structure of the application, so the only thing left is to create the game object. In this example, I called the project Formic, so it set up the source files for the application delegate and called them *FormicAppDelegate.h* and *FormicAppDelegate.m*. It did the same for the view controller, which it created inside an Interface Builder file that it called *FormicViewController.xib*. It set up the source files for this too, named *FormicViewController.h*

and *FormicViewController.m*. Finally, it set up all the necessary connections in Interface Builder so that you already have a convenient `FormicViewController` variable inside your application delegate.

Although these files are created automatically for you, it's a good idea to take a step back and look at what has been created and where to find it.

The FormicApplicationDelegate is the starting point. When the application has started, it will call the `applicationDidFinishLaunching:` method. This is where the code can get things going like creating the game object.

The `FormicViewController` itself lives inside the XIB file. It will be instantiated by the application at startup. You will find a pointer to your view controller in the application delegate, and you will find empty shells for your view controller source files in your project. Just add your controller logic there.

Finally, the view that has been set up for you already lives inside the XIB file. This simple `UIView` will not display anything. To get something displayed, you will have to create a subclass of `UIView`. To do this, select the Classes group in the project tree and choose **New File** from Xcode's **File** menu, as shown in Figure 1-7.

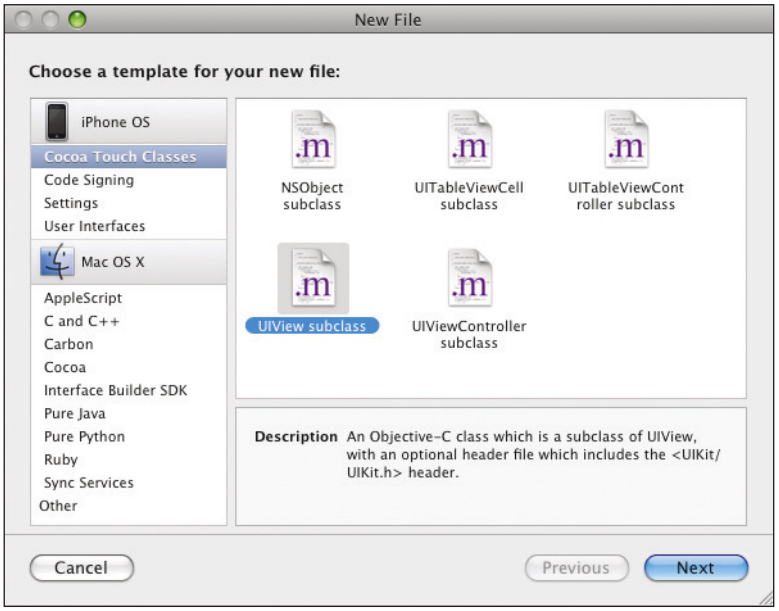


Figure 1-7. *Xcode's New File dialog to create the UIView subclass*

Call them *FormicView.m* to keep to the naming scheme used so far. The files will be created prefilled with all the code necessary to subclass from `UIView` and added to your project. Add your view code in these files.