

C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Learning C++ by Creating Games with UE4

Learn C++ programming with a fun, real-world application that allows you to create your own games!

William Sherif

[PACKT]
PUBLISHING

Learning C++ by Creating Games with UE4

Learn C++ programming with a fun, real-world application that allows you to create your own games!

William Sherif



BIRMINGHAM - MUMBAI

Learning C++ by Creating Games with UE4

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2015

Production reference: 1180215

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78439-657-2

www.packtpub.com

Credits

Author

William Sherif

Project Coordinator

Rashi Khivansara

Reviewers

Brandon Mann

Matt Sutherlin

Alan Wolfe

Proofreaders

Martin Diver

Lawrence A. Herman

Paul Hindle

Commissioning Editor

Edward Bowkett

Indexer

Hemangini Bari

Acquisition Editor

Subho Gupta

Graphics

Sheetal Aute

Content Development Editor

Anand Singh

Production Coordinator

Melwyn D'sa

Technical Editor

Saurabh Malhotra

Cover Work

Melwyn D'sa

Copy Editors

Dipti Kapadia

Deepa Nambiar

About the Author

William Sherif is a C++ programmer with more than 8 years' programming experience. He has a wide range of experience in the programming world, from game programming to web programming. He has also worked as a university course instructor (sessional) for 7 years.

He has released several apps on to the iTunes store, including strum and MARSHALL OF THE ELITE SQUADRON.

In the past, he has won acclaim for delivering course material in an easy-to-understand manner.

I'd like to thank my family, for their support when I was writing this book; Mostafa and Fatima, for their hospitality; as well as Ryan and Reda, for letting me write.

About the Reviewers

Brandon Mann is a well-rounded game developer with over 7 years of professional game-development experience. He has worked on a wide range of titles, from Indie Games to AAA titles, and at companies such as Warner Bros., Midway, and 343 Industries. He has worked on several titles, including *Blacklight*, *Tango Down*, *Gotham City Impostors*, and *Battle Nations*.

Matt Sutherlin has been working in the games industry for over a decade now, where he's held job titles ranging from QA and scripter to engine programmer, and technical artist. Most recently, he has been heavily focusing on graphics technology, working on engine renderers, art pipelines, and shaders for AAA titles, such as *Heroes of the Storm* and *Halo 5: Guardians*.

I would like to thank my beautiful wife, Megan, for years of support and understanding and Matthew Phillips for giving me my debut in the industry.

Alan Wolfe is a self-taught game and engine programmer who has worked at companies such as inXile Entertainment, Midway, Warner Bros., and Blizzard Entertainment. He has worked on titles including *Line Rider 2*, *Unbound*, *Gotham City Impostors*, *Battle Nations*, *Insanely Twisted Shadow Planet*, and *StarCraft II: Heart of the Swarm*. Alan is currently a senior engine programmer at Blizzard Entertainment, where he works on *StarCraft II* and *Heroes of the Storm*.

I'd like to thank Packt Publishing and the author for allowing me to contribute to this book and to help budding game programmers learn the same way I did. If you want to succeed as a game programmer, practice implementing everything you learn, hang out with like-minded individuals, who want to achieve the same things you do, and never stop learning new things.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: Coding with C++	9
Setting up our project	9
Using Microsoft Visual C++ on Windows	10
Using XCode on a Mac	14
Creating your first C++ program	17
Semicolons	21
Handling errors	21
Warnings	22
What is building and compiling?	23
Scripting	23
Exercise – ASCII art	23
Summary	24
Chapter 2: Variables and Memory	25
Variables	26
Declaring variables – touching the silicon	26
Reading and writing to your reserved spot in memory	27
Numbers are everything	28
More on variables	29
Math in C++	31
Exercises	32
Generalized variable syntax	33
Primitive types	33
Object types	34
Exercise – Player	36
Pointers	37
What can pointers do?	38
Address of operator &	39
The Null pointers	40

cin	41
printf()	41
Exercise	42
Solution	42
Summary	43
Chapter 3: If, Else, and Switch	45
Branching	46
Controlling the flow of your program	46
The == operator	47
Coding if statements	47
Coding else statements	49
Testing for inequalities using other comparison operators (>, >=, <, <=, and !=)	50
Using logical operators	51
The Not (!) operator	51
Exercises	52
Solution	52
The And (&&) operator	52
The Or () operator	53
Our first example with Unreal Engine	53
Exercise	60
Solution	60
Branching code in more than two ways	61
The else if statement	61
Exercise	62
Solution	63
The switch statement	64
Switch versus if	66
Exercise	67
Solution	68
Summary	69
Chapter 4: Looping	71
The while loop	71
Infinite loops	73
Exercises	74
Solutions	74
The do/while loop	75
The for loop	76
Exercises	77
Solutions	78
Looping with Unreal Engine	78
Summary	80

Chapter 5: Functions and Macros	81
Functions	81
An example of a <cmath> library function – sqrt()	83
Writing our own functions	84
A sample program trace	85
Exercise	88
Solution	88
Functions with arguments	88
Functions that return values	89
Exercises	91
Solutions	91
Variables, revisited	92
Global variables	92
Local variables	93
The scope of a variable	94
Static local variables	96
Const variables	97
Function prototypes	97
.h and .cpp files	97
prototypes.h contains	98
funcs.cpp contains	99
main.cpp contains	99
Extern variables	100
Macros	100
Advice – try to use const variables where possible	101
Macros with arguments	101
Advice – use inline functions instead of macros with arguments	102
Summary	103
Chapter 6: Objects, Classes, and Inheritance	105
struct objects	106
Member functions	107
The this keyword	107
Strings are objects?	108
Invoking a member function	109
Exercises	110
Solutions	110
Privates and encapsulation	111
Some people like it public	112
class versus struct	113

Getters and setters	113
Getters	114
Setters	115
But what's the point of get/set operations?	116
Constructors and destructors	117
Class inheritance	118
Derived classes	118
Syntax of inheritance	122
What does inheritance do?	123
is-a relationship	123
protected variables	124
Virtual functions	124
Purely virtual functions (and abstract classes)	125
Multiple inheritance	125
private inheritance	126
Putting your classes into headers	127
.h and .cpp	129
Exercise	131
Summary	131
Chapter 7: Dynamic Memory Allocation	133
Dynamic memory allocation	135
The delete keyword	135
Memory leaks	136
Regular arrays	137
The array syntax	137
Exercise	138
Solutions	139
C++ style dynamic size arrays (new[] and delete[])	139
Dynamic C-style arrays	141
Summary	142
Chapter 8: Actors and Pawns	143
Actors versus pawns	143
Creating a world to put your actors in	144
The UE4 editor	147
Editor controls	147
Play mode controls	148
Adding objects to the scene	148
Starting from scratch	151
Adding light sources	152
Collision volumes	154
Adding collision detection for the objects editor	154

Adding an actor to the scene	156
Creating a player entity	156
Inheriting from UE4 GameFramework classes	156
Associating a model with the Avatar class	159
Loading the mesh	161
Creating a blueprint from our C++ class	161
Writing C++ code that controls the game's character	169
Making the player an instance of the Avatar class	169
Setting up controller inputs	172
Exercise	174
Solution	174
Yaw and pitch	176
Creating non-player character entities	177
Displaying a quote from each NPC dialog box	181
Displaying messages on the HUD	181
Using TArray<Message>	184
Exercise	186
Solution	187
Triggering an event when it is near an NPC	187
Make the NPC display something to the HUD when something is nearby	189
Exercises	190
Solutions	191
Summary	192
Chapter 9: Templates and Commonly Used Containers	193
Debugging the output in UE4	194
UE4's TArray<T>	194
An example that uses TArray<T>	195
Iterating a TArray	196
Finding whether an element is in the TArray	198
TSet<T>	199
Iterating a TSet	199
Intersecting TSet	200
Unioning TSet	200
Finding TSet	200
TMap<T, S>	200
A list of items for the player's inventory	201
Iterating a TMap	201
C++ STL versions of commonly used containers	202
C++ STL set	203
Finding an element in a <set>	204
Exercise	204
Solution	204

C++ STL map	205
Finding an element in a <map>	206
Exercise	206
Solution	206
Summary	207
Chapter 10: Inventory System and Pickup Items	209
Declaring the backpack	209
Forward declaration	210
Importing assets	211
Attaching an action mapping to a key	216
Base class PickupItem	217
The root component	220
Getting the avatar	222
Getting the player controller	222
Getting the HUD	222
Drawing the player inventory	223
Using HUD::DrawTexture()	224
Exercise	227
Detecting inventory item clicks	227
Dragging elements	228
Exercises	231
Summary	232
Chapter 11: Monsters	233
Landscape	234
Sculpting the landscape	237
Monsters	238
Basic monster intelligence	243
Moving the monster – steering behavior	243
The discrete nature of monster motion	246
Monster SightSphere	248
Monster attacks on the player	250
Melee attacks	251
Defining a melee weapon	251
Sockets	257
Creating a skeletal mesh socket in the monster's hand	258
Attaching the sword to the model	260
Code to equip the player with a sword	261
Triggering the attack animation	263
Projectile or ranged attacks	277
Bullet physics	279
Adding bullets to the monster class	281
Player knockback	285
Summary	286

Chapter 12: Spell Book	287
The particle systems	289
Changing particle properties	291
Settings for the blizzard spell	294
Spell class actor	300
Blueprinting our spells	303
Picking up spells	305
Creating blueprints for PickupItems that cast spells	306
Attaching right mouse click to cast spell	308
Writing the avatar's CastSpell function	309
Instantiating the spell – GetWorld()->SpawnActor()	309
if(spell)	310
spell->SetCaster(this)	310
Writing AMyHUD::MouseRightClicked()	310
Activating right mouse button clicks	312
Creating other spells	314
The fire spell	314
Exercises	315
Summary	315
Index	317

Preface

So, you want to program your own games using Unreal Engine 4 (UE4). You have a great number of reasons to do so:

- UE4 is powerful: UE4 provides some of the most state-of-the-art, beautiful, realistic lighting and physics effects, of the kind used by AAA Studios.
- UE4 is device-agnostic: Code written for UE4 will work on Windows desktop machines, Mac desktop machines, Android devices, and iOS devices (at the time of writing this book – even more devices may be supported in the future).

So, you can use UE4 to write the main parts of your game once, and after that, deploy to iOS and Android Marketplaces without a hitch. (Of course, there will be a few hitches: iOS and Android in app purchases will have to be programmed separately.)

What is a game engine anyway?

A game engine is analogous to a car engine: the game engine is what drives the game. You will tell the engine what you want, and (using C++ code and the UE4 editor) the engine will be responsible for actually making that happen.

You will build your game around the UE4 game engine, similar to how the body and wheels are built around an actual car engine. When you ship a game with UE4, you are basically customizing the UE4 engine and retrofitting it with your own game's graphics, sounds, and code.

What will using UE4 cost me?

The answer, in short, is \$19 and 5 percent of sales.

"What?" you say. \$19?

That's right. For only \$19, you get full access to a world class AAA Engine, complete with a source. This is a great bargain, considering the fact that other engines can cost anywhere from \$500 to \$1,000 for just a single license.

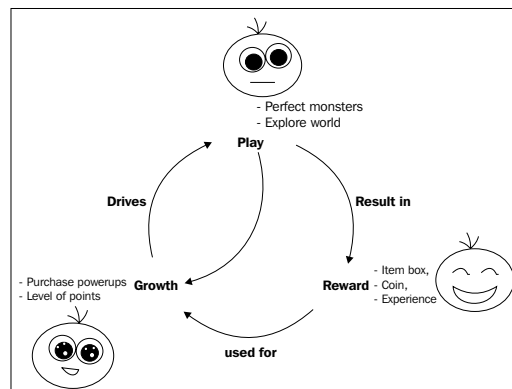
Why don't I just program my own engine and save the 5 percent?

Take it from me, if you want to create games within a reasonable time frame and you don't have a large team of dedicated engine programmers to help you, you'll want to focus your efforts on what you sell (your game).

Not having to focus on programming a game engine gives you the freedom to think only about how to make the actual game. Not having to maintain and bug-fix your own engine is a load off your mind too.

A game's overview – the Play-Reward-Growth loop

I want to show you this diagram now because it contains a core concept that many novice developers might miss when writing their first games. A game can be complete with sound effects, graphics, realistic physics, and yet, still not feel like a game. Why is that?



Starting at the top of the loop, Play actions committed during the game (such as defeating a monster) result in rewards for the player (such as gold or experience). These rewards, in turn, can be used for in-game Growth (such as stats increases or new worlds to explore). This Growth then drives the gameplay in new and interesting ways. For example, a new weapon can change the basic mechanics of fighting, new spells let you take on groups of monsters with a completely different approach, or new modes of transportation can let you reach areas that were previously inaccessible.


This is the basic core loop that creates interesting gameplay. The key is that Play must result in some kind of Reward — think of glittering gold pieces popping out of nasty baddies. For rewards to have a point, it must result in some kind of Growth in the gameplay. Think about how many new locations were unlocked with the hook shot in *The Legend of Zelda*.

A game that is only Play (without Rewards or Growth) won't feel like a game: it will feel only like a really basic prototype of a game. For example, imagine a flight simulator with just an open world and no goals or objectives as well as without the ability to upgrade your plane or weapons. It wouldn't be much of a game.

A game with only Play and Rewards (but no Growth) will feel primitive and simple. The rewards will not satisfy the player if they cannot be used for anything.

A game with only Play and Growth (without Rewards) will just be seen as a mindless increasing challenge, without giving the player a sense of gratification for his achievements.

A game with all three elements will keep the player engaged with an entertaining Play. The Play has a rewarding result (loot drops and story progression), which results in the Growth of the game world. Keeping this loop in mind while you are devising your game will really help you to design a complete game.

 A prototype is the proof of concept of a game. Say, you want to create your own unique version of *Blackjack*. The first thing you might do is program a prototype to show how the game will be played.

Monetization

Something you need to think about early in your game's development is your monetization strategy. How will your game make money? If you are trying to start a company, you have to think of what will be your sources of revenue from early on.

Are you going to try to make money from the purchase price, such as *Jamestown*, *The Banner Saga*, *Castle Crashers*, or *Crypt of the Necrodancer*? Or, will you focus on distributing a free game with in-app purchases, such as *Clash of Clans*, *Candy Crush Saga*, or *Subway Surfers*?

A class of games for mobile devices (for example, builder games on iOS) make lots of money by allowing the user to pay in order to skip Play and jump straight to the rewards and Growth parts of the loop. The pull to do this can be very powerful; many people spend hundreds of dollars on a single game.

Why C++

UE4 is programmed in C++. To write code for UE4, you must know C++.

C++ is a common choice for game programmers because it offers very good performance combined with object-oriented programming features. It's a very powerful and flexible language.

What this book covers

Chapter 1, Coding with C++, talks about getting up and running with your first C++ program.

Chapter 2, Variables and Memory, talks about how to create, read, and write variables from computer memory.

Chapter 3, If, Else, and Switch, talks about branching the code: that is, allowing different sections of the code to execute, depending on program conditions.

Chapter 4, Looping, discusses how we repeat a specific section of code as many times as needed.

Chapter 5, Functions and Macros, talks about functions, which are bundles of code that can get called any number of times, as often you wish.

Chapter 6, Objects, Classes, and Inheritance, talks about class definitions and instantiating some objects based on a class definition.

Chapter 7, Dynamic Memory Allocation, discusses heap-allocated objects as well as low-level C and C++ style arrays.

Chapter 8, Actors and Pawns, is the first chapter where we actually delve into UE4 code. We begin by creating a game world to put actors in, and derive an Avatar class from a customized actor.

Chapter 9, Templates and Commonly Used Containers, explores UE4 and the C++ STL family of collections of data, called containers. Often, a programming problem can be simplified many times by selecting the right type of container.

Chapter 10, Inventory System and Pickup Items, discusses the creation of an inventory system with the ability to pick up new items.

Chapter 11, Monsters, teaches how to create monsters that give chase to the player and attack it with weapons.

Chapter 12, Spell Book, teaches how to create and cast spells in our game.

What you need for this book

To work with this text, you will need two programs. The first is your integrated development environment, or IDE. The second piece of software is, of course, the Unreal Engine itself.

If you are using Microsoft Windows, then you will need Microsoft Visual Studio 2013 Express Edition for Windows Desktop. If you are using a Mac, then you will need Xcode. Unreal Engine can be downloaded from <https://www.unrealengine.com/>.

Who this book is for

This book is for anyone who wants to write an Unreal Engine application. The text begins by telling you how to compile and run your first C++ application, followed by chapters that describe the rules of the C++ programming language. After the introductory C++ chapters, you can start to build your own game application in C++.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The `variableType` is going to tell you what type of data we are going to store in our variable. The `variableName` is the symbol we'll use to read or write that piece of memory".

A block of code is set as follows:

```
struct Player
{
    string name;
    int hp;
    // A member function that reduces player hp by some amount
    void damage( int amount ) {
        hp -= amount;
    }
    void recover( int amount ) {
        hp += amount;
    }
};
```

New terms and **important words** are shown in bold. Text that appears on the screen appears like this: From the **File** menu, select **New Project...**



Extra information that is relevant, but kind of a side note, appears in boxes like this.



Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from https://www.packtpub.com/sites/default/files/downloads/65720T_ColoredImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Coding with C++

You're a first-time programmer. You have a lot to learn!

Academics often describe programming concepts in theory but like to leave implementation to someone else, preferably someone from the industry. We don't do that in this book—in this book, we will describe the theory behind C++ concepts and implement our own game as well.

The first thing I will recommend is that you do the exercises. You cannot learn to program simply by reading. You must work with the theory with the exercises.

We are going to get started by programming very simple programs in C++. I know that you want to start playing your finished game right now. However, you have to start at the beginning to get to that end (if you really want to, skip over to *Chapter 12, Spell Book*, or open some of the samples to get a feel for where we are going).

In this chapter, we will cover the following topics:


- Setting up a new project (in Visual Studio and Xcode)
- Your first C++ project
- How to handle errors
- What are building and compiling?

Setting up our project

Our first C++ program will be written outside of UE4. To start with, I will provide steps for both Xcode and Visual Studio 2013, but after this chapter, I will try to talk about just the C++ code without reference to whether you're using Microsoft Windows or Mac OS.


Using Microsoft Visual C++ on Windows

In this section, we will install a code editor for Windows, Microsoft's Visual Studio. Please skip to the next section if you are using a Mac.

 The Express edition of Visual Studio is the free version of Visual Studio that Microsoft provides on their website. Go to <http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx> to start the installation process.

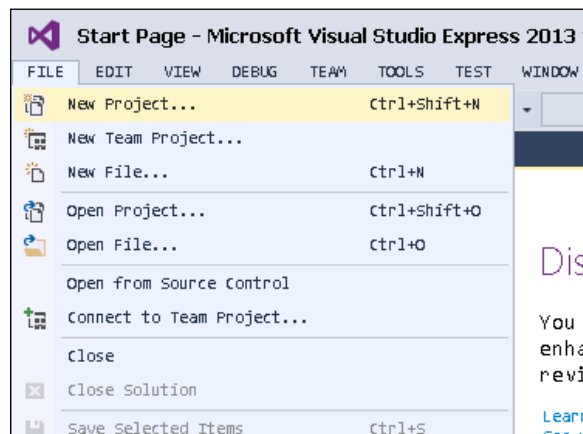
To start, you have to download and install **Microsoft Visual Studio Express 2013 for Windows Desktop**. This is how the icon for the software looks:



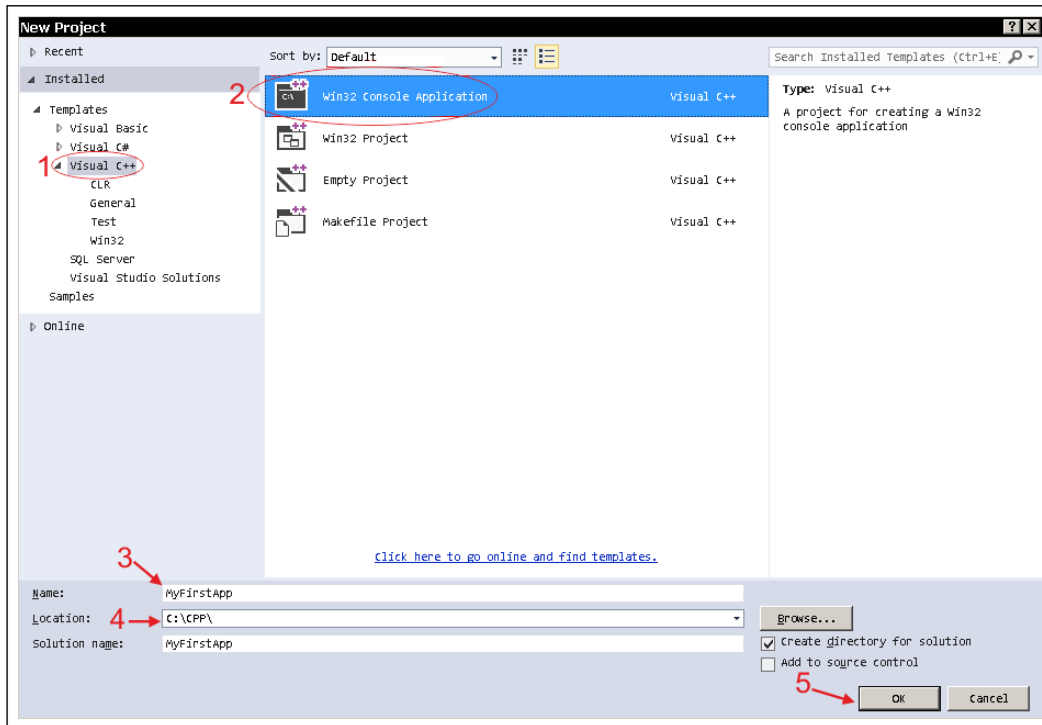
 Do not install **Express 2013 for Windows**. This is a different package and it is used for different things than what we are doing here.

Once you have Visual Studio 2013 Express installed, open it. Work through the following steps to get to a point where you can actually type in the code:

1. From the **File** menu, select **New Project...**, as shown in the following screenshot:



2. You will get the following dialog:

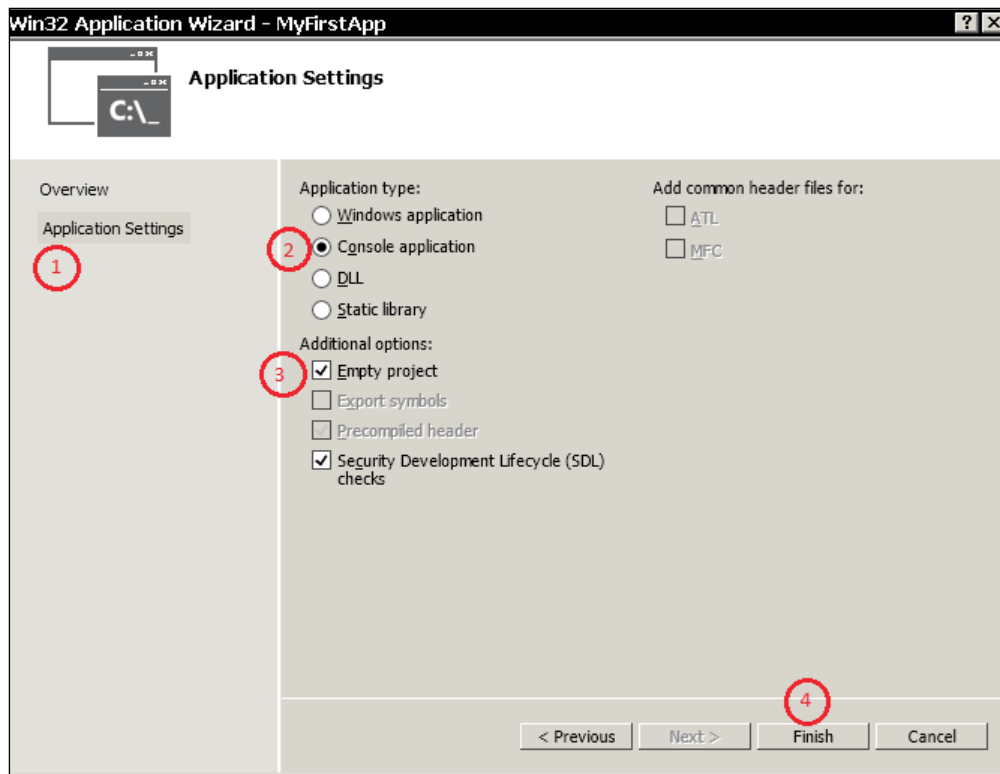


Note that there is a small box at the bottom with the text **Solution name**. In general, **Visual Studio Solutions** might contain many projects. However, this book only works with a single project, but at times, you might find it useful to integrate many projects into the same solution.

3. There are five things to take care of now, as follows:

1. Select **Visual C++** from the left-hand side panel.
2. Select **Win32 Console Application** from the right-hand side panel.
3. Name your app (I used MyFirstApp).
4. Select a folder to save your code.
5. Click on the **OK** button.

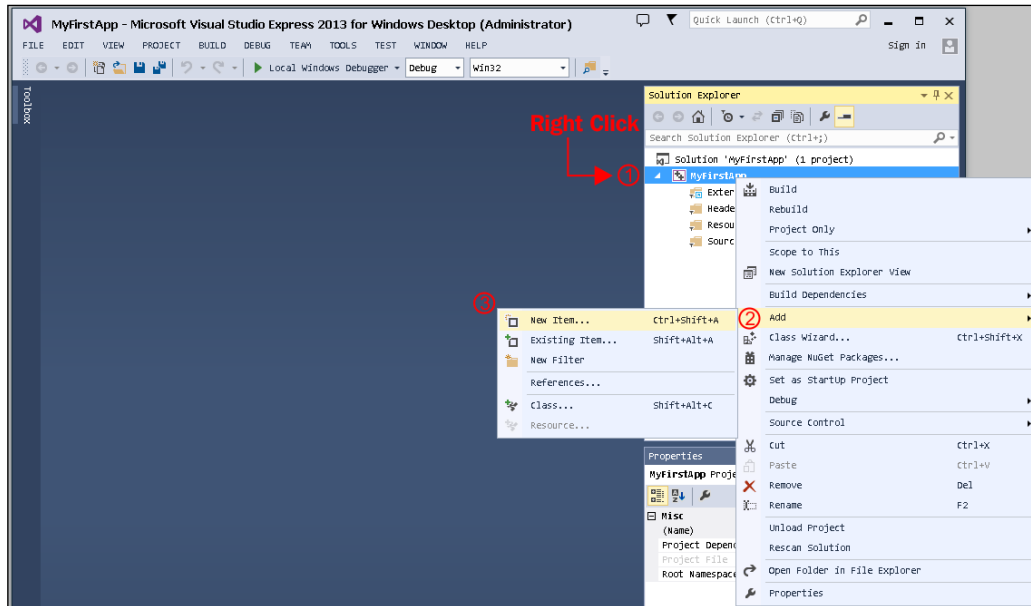
4. After this an **Application Wizard** dialog box opens up, as shown in the following screenshot:



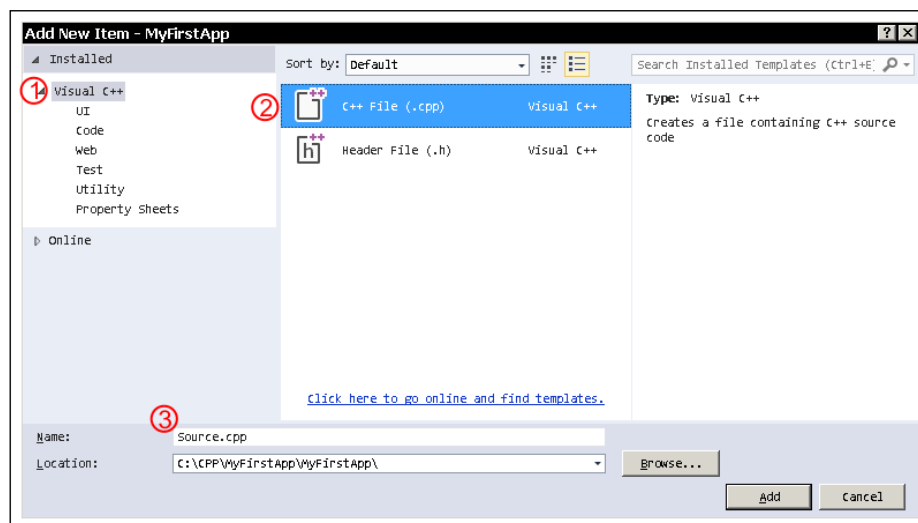
5. We have four things to take care of in this dialog box, as follows:
 1. Click on **Application Settings** in the left-hand side panel.
 2. Ensure that **Console application** is selected.
 3. Select **Empty project**.
 4. Click on **Finish**.

Now you are in the Visual Studio 2013 environment. This is the place where you will do all your work and code.

However, we need a file to write our code into. So, we will add a C++ code file to our project, as shown in the following screenshot:



Add your new source code file as shown in the following screenshot:



You will now edit `Source.cpp`. Skip to the Your First C++ Program section and type in your code.

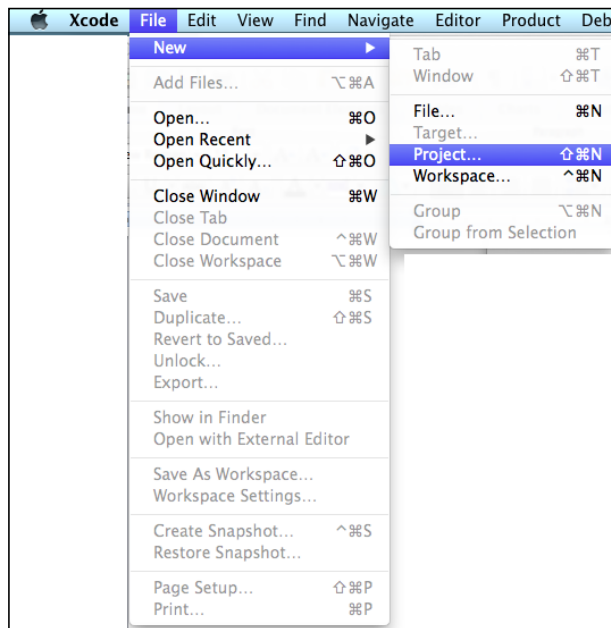
Using XCode on a Mac

In this section, we will talk about how to install Xcode on a Mac. Please skip to the next section if you are using Windows.

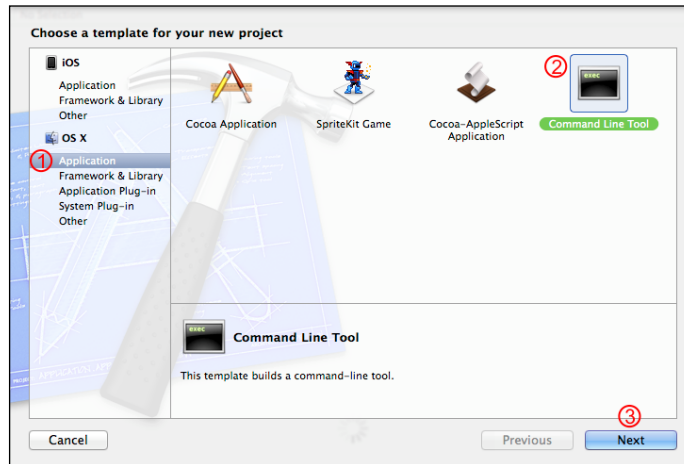
Xcode is available on all Mac machines. You can get Xcode using the Apple App Store (it's free), as shown here:




1. Once you have Xcode installed, open it. Then, navigate to **File | New | Project...** from the system's menu bar at the top of your screen, as shown in the following screenshot:

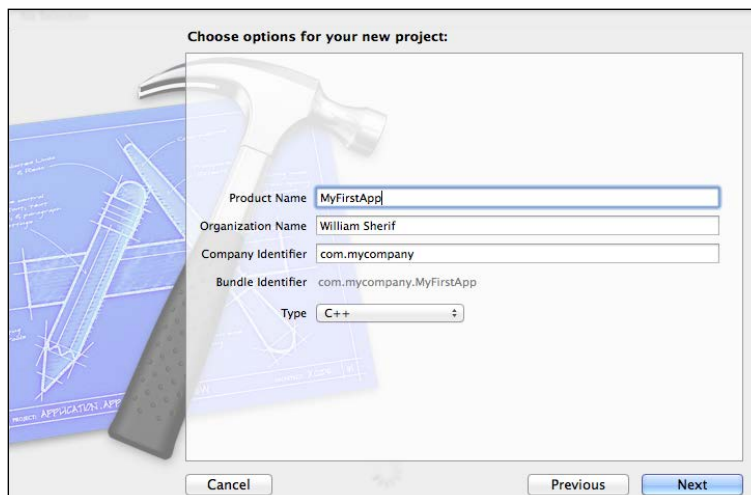


2. In the New Project dialog, select **Application** under **OS X** on the left-hand side of the screen, and select **Command Line Tool** from the right-hand side pane. Then, click on **Next**:

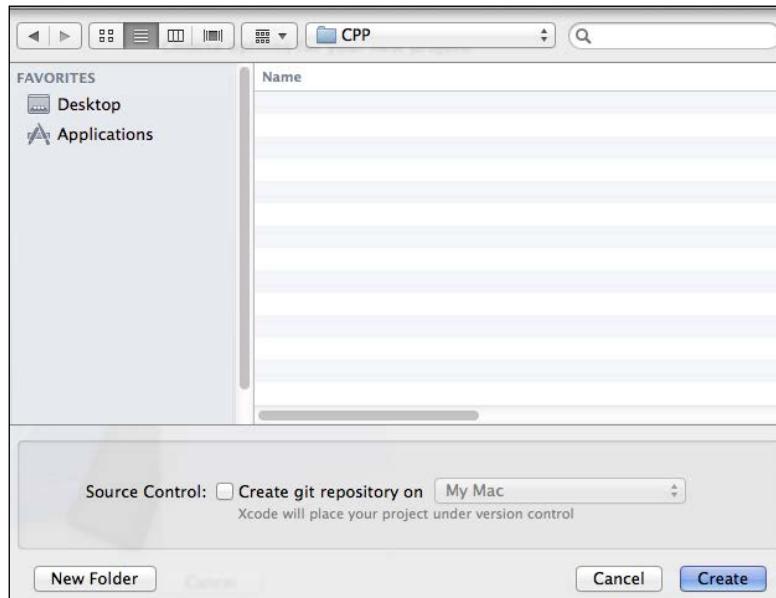


 You might be tempted to click on the **SpriteKit Game** icon, but don't click on it.

3. In the next dialog, name your project. Be sure to fill in all the fields or Xcode won't let you proceed. Make sure that the project's **Type** is set to **C++** and then click on the **Next** button, as shown here:

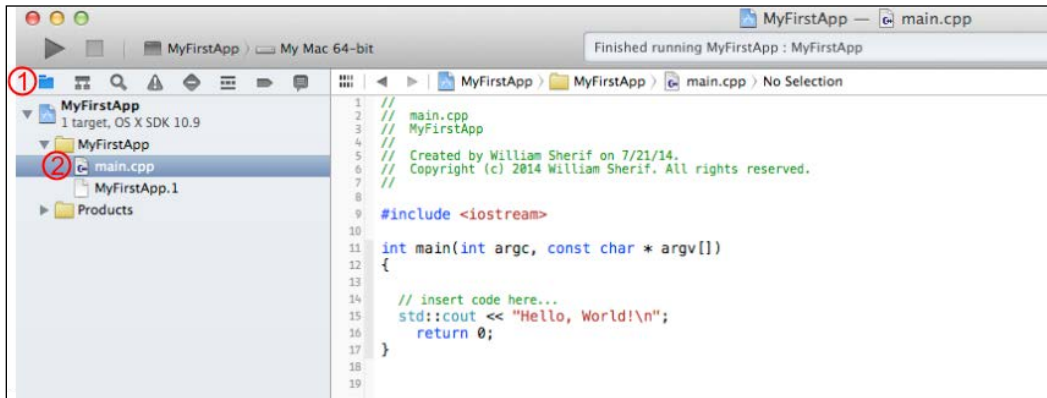


- The next popup will ask you to choose a location in order to save your project. Pick a spot on your hard drive and save it there. Xcode, by default, creates a Git repository for every project you create. You can uncheck **Create git repository** — we won't cover Git in this chapter — as shown in the following screenshot:



Git is a **Version control system**. This basically means that Git keeps the snapshots of all the code in your project every so often (every time you *commit* to the repository). Other popular **source control management** tools (**scm**) are Mercurial, Perforce, and Subversion. When multiple people are collaborating on the same project, the scm tool has the ability to automatically merge and copy other people's changes from the repository to your local code base.

Okay! You are all set up. Click on the **main.cpp** file in the left-hand side panel of Xcode. If the file doesn't appear, ensure that the folder icon at the top of the left-hand side panel is selected first, as shown in the following screenshot:



Creating your first C++ program

We are now going to write some C++ source code. There is a very good reason why we are calling it the source code: it is the source from which we will build our binary executable code. The same C++ source code can be built on different platforms such as Mac, Windows, and iOS, and in theory, an executable code doing the exact same things on each respective platform should result.

In the not-so-distant past, before the introduction of C and C++, programmers wrote code for each specific machine they were targeting individually. They wrote code in a language called assembly language. But now, with C and C++ available, a programmer only has to write code once, and it can be deployed to a number of different machines simply by sending the same code through different compilers.



In practice, there are some differences between Visual Studio's flavor of C++ and Xcode's flavor of C++, but these differences mostly come up when working with advanced C++ concepts, such as templates. One of the main reasons why using UE4 is so helpful is that UE4 will erase a lot of the differences between Windows and Mac. The UE4 team did a lot of magic in order to get the same code to work on both Windows and Mac.




A real-world tip

It is important for the code to run in the same way on all machines, especially for networked games or games that allow things such as shareable replays. This can be achieved using standards. For example, the IEEE floating-point standard is used to implement decimal math on all C++ compilers. This means that the result of computations such as $200 * 3.14159$ should be the same on all the machines.

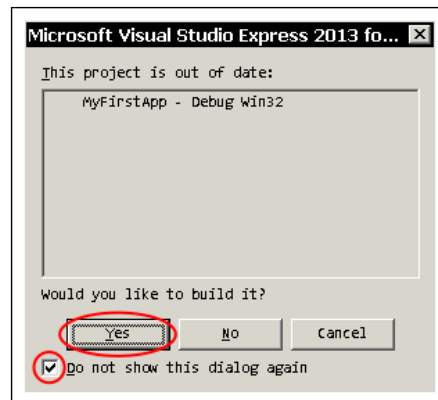
Write the following code in Microsoft Visual Studio or in Xcode:

```
#include <iostream> // Import the input-output library
using namespace std; // allows us to write cout
                      // instead of std::cout

int main()
{
    cout << "Hello, world" << endl;
    cout << "I am now a C++ programmer." << endl;
    return 0;        // "return" to the operating sys
}
```

Press *Ctrl* + *F5* to run the preceding code in Visual Studio, or press  + *R* to run in Xcode.

The first time you press *Ctrl* + *F5* in Visual Studio, you will see this dialog:



Select **Yes** and **Do not show this dialog again** – trust me, this will avoid future problems.