



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

# Unity 3.x Scripting

Write efficient, reusable scripts to build custom characters, game environments, and control enemy AI in your Unity game

Volodymyr Gerasimov

Devon Kraczla

**[PACKT]**  
PUBLISHING

# Unity 3.x Scripting

Write efficient, reusable scripts to build custom characters, game environments, and control enemy AI in your Unity game

**Volodymyr Gerasimov**

**Devon Kraczla**



BIRMINGHAM - MUMBAI

# Unity 3.x Scripting

Copyright © 2012 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 authors, 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: June 2012

Production Reference: 1140612

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

ISBN 978-1-84969-230-4

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Karl Moore ([karl.moore@ukonline.co.uk](mailto:karl.moore@ukonline.co.uk))

# Credits

**Authors**

Volodymyr Gerasimov  
Devon Kraczla

**Reviewers**

Peter Chan  
Jeff Munde

**Acquisition Editor**

Rashmi Phadnis

**Lead Technical Editor**

Hithesh Uchil

**Technical Editor**

Devdutt Kulkarni

**Project Coordinator**

Alka Nayak

**Proofreader**

Bernadette Watkins

**Indexer**

Monica Ajmera

**Production Coordinator**

Shantanu Zagade

**Cover Work**

Shantanu Zagade

# About the Authors

**Volodymyr Gerasimov** is a level designer and scripter. His major passion is creating modifications for popular games, and developing small, indie projects, with scripting as a main tool. He learned various scripting and programming languages at The Art Institute of Vancouver. Introduced to Unity in 2010, he created and worked on a number of projects, indie games, and prototypes. He has worked as Lead Level Designer and Scripter, on the hack-and-slash action game, *Splik and Blitz: Baked in Blood*, and has also worked on a couple of indie projects for iOS and PC. His latest, finished project is the puzzle platformer game, *Red Rolling Hood*. Currently, he is working at Best Way, as Producer of an action role-playing game.

---

I would like to thank all my friends and teachers who shared their experience with me. They surrounded me with an aura of creativity and art, which kept my passion burning, and my work going. I would also like to thank all who will open this book, and be able to learn something, create, and share.

---

**Devon Kraczla** is an independent game developer. Having an artistic background, Devon came to the gaming industry to explore new ways to surprise people with his creations. Over the last couple of years, having graduated from The Art Institute of Vancouver, Devon has developed multiple, independent projects, both solo and with other enthusiasts, and has worked on the award-winning *Battlefield 3*, as a member of the motion capture team at EA Canada. In his games, Devon focuses on simple and engaging game mechanics, covered with a unique art style that makes his games appealing for hardcore and casual audiences alike. Currently, Devon is working on a new project along with a large group of passionate developers.

---

I would like to thank my teachers and peers of The Art Institute of Vancouver, for helping me pursue the endeavors that I sought after. I would also like to thank my friends and family, outside of my school life, who helped keep me sane, well, as sane as I can be, and for being there when it mattered most. Prost!

---

# About the Reviewer

**Jeff Munde** is a game designer and instructor from New Brunswick, USA, who moved to Vancouver, Canada, a decade ago to produce video games. Since then he has worked on many game projects in various roles, from Motion Capture Specialist at Electronic Arts, to Game Designer for Activision, and all sorts of independent productions in between. He is currently working on a Unity-based game with Holy Mountain Games. He also teaches classes at The Art Institute of Vancouver, about game production using Unity, among other subjects.

---

I would like to thank Vlad and Devon for being leaders in a strong graduating class, by taking the initiative to master Unity. I know they will both go on to make great games.

---

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

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](http://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](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://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.



<http://PacktLib.PacktPub.com>

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

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Diving into Scripting</b>	<b>5</b>
<b>Downloading and installing assets for this book</b>	<b>5</b>
<b>Getting started with the game</b>	<b>8</b>
<b>Available Character Controllers</b>	<b>8</b>
<b>Interactive objects</b>	<b>12</b>
Triggers	12
Buttons	12
Base button script	13
Activating platform status	13
Explosion box	15
The Update function	15
The BOOM function	16
Downloading the Detonator package	17
Pressing the button	19
<b>Dynamic objects</b>	<b>20</b>
Moving boxes	20
Triggered object	23
Moving platform	23
Moving the character with the platform	25
<b>Summary</b>	<b>27</b>
<b>Chapter 2: Custom Character Controller</b>	<b>29</b>
<b>Creating a controllable character</b>	<b>29</b>
<b>Custom Character Controller</b>	<b>31</b>
Setting up the project	32
Creating movement	33
Manipulating character vector	33
Register input from the user	34
The Rigidbody component	35

Jumping	36
User input verification	36
Raycasting	38
Additional jump functionality	40
Running	42
<b>Cameras</b>	<b>42</b>
Camera scripting	42
Creating camera script	43
Creating an enumeration list	44
Writing functions	44
Writing camera switching controls	47
Character movement and camera positioning	48
Updating camera type changing	49
Influencing camera with a mouse	50
Clamping angles	51
Camera's late update	53
Rotating character with a camera	53
<b>Animation controls</b>	<b>55</b>
Playing simple animations	55
Start function versus Awake function	56
Animation component and playing speed	57
Animation scripting	59
Walk, run, and idle animations	61
<b>Summary</b>	<b>63</b>
<b>Chapter 3: Action Game Essentials</b>	<b>65</b>
<b>Programming weapons and pickables</b>	<b>65</b>
Creating the base	66
Programming the weapon	68
The Shooting function	71
Shooting cooldown	72
Alternative shooting function	73
<b>Advanced animation system</b>	<b>74</b>
Working of an animation	75
Animation mixing	75
Animation script overview	78
<b>Weapon pickup</b>	<b>80</b>
<b>Adding ammo and health pickups</b>	<b>82</b>
<b>Creating a treasure chest</b>	<b>85</b>
<b>Applying projectile fixes</b>	<b>89</b>
<b>Tethering and soft body</b>	<b>90</b>
Tethering	90
Creating a tether	90
Creating assets	92
Tether manager	93

Creation of tether	94
The StickySegment script	98
Tether scripts overview	101
<b>Summary</b>	<b>103</b>
<b>Chapter 4: Drag-and-Drop Inventory</b>	<b>105</b>
<b>GUI basics</b>	<b>105</b>
GUI.Box	106
GUI.Button	106
GUI.Label	107
GUI.TextField	107
GUI.TextArea	108
GUI.Toggle	108
GUI.Toolbar and GUI.SelectionGrid	109
GUI.HorizontalSlider and GUI.VerticalSlider	110
GUI.HorizontalScrollBar and GUI.VerticalScrollBar	110
GUI.BeginGroup and GUI.EndGroup	111
GUI.BeginScrollView, GUI.EndScrollView, and ScrollTo	111
Other GUI classes	112
<b>Drag-and-drop inventory</b>	<b>112</b>
Basics	113
Inventory slots and draggable objects	114
Working with GUI windows	118
Inventory slots	121
Patching the inventory	126
<b>Character customization</b>	<b>127</b>
3D character avatar	128
Dealing with a camera	128
Adjusting the camera	130
Window dragging limits	131
Customization	132
Setting up items	132
Adding items	133
Modifying character	135
Reloading and inventory	141
Finishing adjustments	142
<b>Summary</b>	<b>144</b>
<b>Chapter 5: Dynamic GUI</b>	<b>145</b>
<b>Radial health display</b>	<b>146</b>
The Health script	146
Health display script	148
Revisiting the Health script	151
Hooking up objects to Inspector	152

<b>Creating items</b>	<b>153</b>
The Change_Item script	154
Setting up the code	154
Changing items	155
Addition and removal	155
Displaying items	156
Increment controls	157
Creating the UseItem script	159
Revisiting the Change_Item script	161
The PlayerStats script	162
The TextManager script	164
The textMesh script	165
Revisiting the UseItem script	167
Revisiting the Health script	169
<b>Creating armor</b>	<b>169</b>
The Armor script	170
Revisiting the HealthBar script	172
Revisiting the Health script	173
Revisiting the UseItem script	174
<b>Creating the weapons</b>	<b>174</b>
The Change_Weapon script	175
The UseWeapon script	176
Revisiting PlayerStats	178
Revisiting the textMesh script	179
<b>Scripting and displaying the score system</b>	<b>180</b>
The Score script	180
Reading from the text file	182
Writing to the text file	183
The timer script	184
Revisiting the textMesh script	185
<b>Displaying the objectives</b>	<b>186</b>
Revisiting TextManager	186
Revisiting textMesh	187
Hooking up HUD	188
Game manager	189
Health	190
Item_Pic	191
ItemMultiplier, highScoreDisplay, ObjectiveDisplay, scoreDisplay, and weaponDisplay	191
saveDisplay	192
Weapon_Pic	192

---

<b>Creating the targeting system</b>	<b>193</b>
Creating the Bezier equation script	194
ArcBehaviour	195
The moveObject script	196
Hooking it up in the editor	197
<b>Summary</b>	<b>197</b>
<b>Chapter 6: Game Master Controller</b>	<b>199</b>
<b>Game manager theory</b>	<b>200</b>
<b>Creating game managers</b>	<b>200</b>
Level streaming	201
Mission creation	204
Managing levels	207
Save/load system	208
Loading with checkpoints	214
<b>GameLoader</b>	<b>217</b>
<b>Dynamic camera</b>	<b>218</b>
<b>Audio</b>	<b>218</b>
<b>Audio manager</b>	<b>221</b>
<b>Summary</b>	<b>222</b>
<b>Chapter 7: Introduction to AI Pathfinding and Behaviors</b>	<b>223</b>
<b>Simple waypoint pathfinding</b>	<b>224</b>
Setting up the hierarchy	225
Writing the waypoint display script	225
Setting up the path arrays	226
Creating the aiSimplePath script	227
Declaring variables	227
Starting up functions	228
Traversing the path	229
Shutting down the robot	232
Hooking up the aiSimplePath script on Inspector	233
<b>Enemy statistics, shooting, and behaviors</b>	<b>233</b>
The enemyStats script	233
Setting up variables	234
Setting up functions	234
Retrieving functions	234
Manipulation functions	234
Hooking up the enemyStats script on Inspector	236
The Shoot script	236
Setting up the script	236
Writing shooting functionality	237
Hooking up the Shoot script on Inspector	239

The aiSimpleBehaviour script	240
Setting up the script	240
Behavior functions	241
Additional functions	247
Hooking up the aiSimpleBehaviour script on Inspector	248
<b>Returning to the aiSimplePath script</b>	<b>249</b>
Pursue functionality	249
Revisiting the EnemyPath function	250
<b>The bulletCollision, ammoCollision, and AmmoInfo scripts</b>	<b>252</b>
Creating the bulletCollision script	252
Hooking up the bulletCollision script on bullet's Inspector	253
Creating the ammoCollision script	254
Hooking up the ammoCollision script on enemy's Inspector	255
Creating the AmmoInfo script	255
Hooking up the AmmoInfo script on ammo's Inspector	257
<b>Summary</b>	<b>258</b>
<b>Appendix: Object-oriented Programming in Unity</b>	<b>259</b>
<b>Object-oriented programming – basics</b>	<b>259</b>
Encapsulation	259
Classes	260
Constructors	260
Code	260
Inheritance	261
Preparations	261
Code	261
Polymorphism	262
Code	263
Nested classes	263
<b>Summary</b>	<b>263</b>
<b>Index</b>	<b>265</b>

---

# Preface

If you are an enthusiastic gamer who is ready to seriously get into game development, this book will give you a great head start for your journey. We will guide you through the step-by-step process of creating your first playable game prototype, which you will be able to further extend into a full-scale game. This book contains examples of the most important features that can be found in games, and much more; it will help you to understand Unity better, and increase your programming skills.

## What this book covers

*Chapter 1, Diving into Scripting*, will teach you how to set up the project and take advantage of built-in character controllers. We will talk about dynamic objects and their collision, as well as investigate creating a moving platform and explosions.

*Chapter 2, Custom Character Controller*, will show you how to create your own character controllers, camera rigs, and animation systems.

*Chapter 3, Action Game Essentials*, will introduce programming of basic gameplay features, such as shooting, picking up items, and opening treasure boxes, as well as soft bodies and tethering.

*Chapter 4, Drag-and-Drop Inventory*, will give you an example on how to create your own inventory and character customization with the help of Unity GUI.

*Chapter 5, Dynamic GUI*, will take you step by step, through the creation of the HUD and targeting system.

*Chapter 6, Game Master Controller*, will teach you how to design and program systems to run and manage your game.

*Chapter 7, Introduction to AI Pathfinding and Behaviors*, will give you a sneak peek of AI programming, and talk about the basic theory behind it.

*Appendix, Object-oriented Programming in Unity*, will cover some basics of programming that will help you to continue learning.

## What you need for this book

You need to be comfortable in an editor's environment, and have a very basic knowledge of Unity's JavaScripts, or any other object-oriented programming language.

## Who this book is for

This book is for passionate game developers, students who are preparing to make their first project, or people who think they are ready to learn something new.

## 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 are shown as follows: "After the `Start` function, we will create the `MoveButton` function."

A block of code is set as follows:

```
function Update() {
    if( tnt != null ) {
        If(trigObj.GetComponent("Button").ReturnButtonStatus()) {
            BOOM();
        }
    }
}
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "To gain access to the package data, open **Unity** and go to **Assets | Import Package | Custom Package...**, as shown in the following screenshot".

 Warnings or important notes appear in a box 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](mailto:feedback@packtpub.com), and mention the book title through 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](http://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.

## 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/support>, 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 to our website, or added to any list of existing errata, under the Errata section of that title.

## 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](mailto: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](mailto: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

## Diving into Scripting

Welcome to advanced Unity scripting! In this book, we will cover interesting information about scripting in Unity's built-in scripting language—JavaScript for Unity. We believe that this book, and included material, has the fundamentals needed to create a game that you always dreamed of creating.

In order to start working with this book, you need to have a basic understanding of what Unity3D is; navigate freely inside Unity, and have basic knowledge of JavaScript and **object-oriented programming (OOP)** in general.

In this chapter, we will:

- Set up a project and a third-person Character Controller
- Talk about dynamic objects and collision detection
- Create moving platform and explosion box

## Downloading and installing assets for this book

In Unity3D, there is the ability to download pre-made packages or import assets. These packages/assets can be of 3D models in the form of raw art assets, game objects, prefabs, particles, scripts, animations, sounds, and so on. Packages are identified by having a `.package` extension.

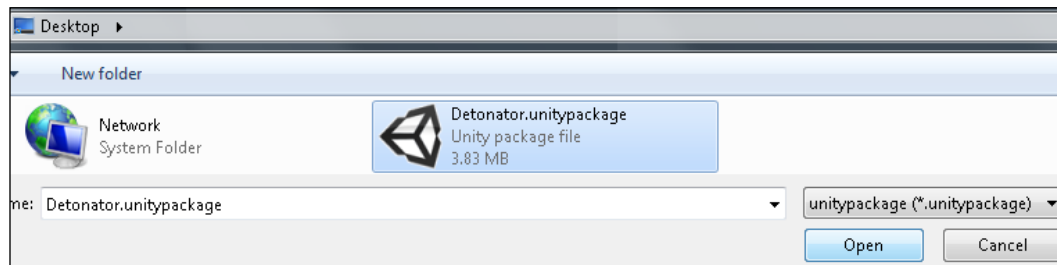
In order for the reader to be able to follow along with the examples in the book, get the greatest amount of experience, and practice out scripting in Unity, pre-made packages have been made available for the reader's convenience.



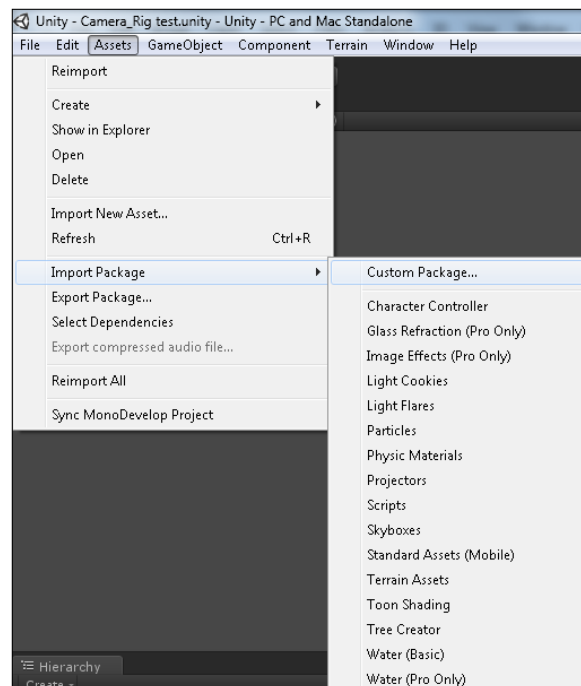
### Downloading the example code

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.

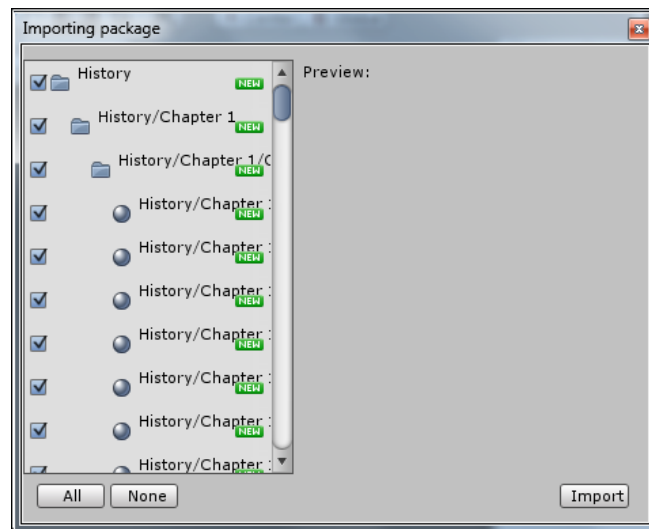
These packages are available for download on the book's website underneath the **Packages** heading. There is only one package here and it is called **Unity\_Scripting.unpackage**. The downloaded file will be a ZIP file.



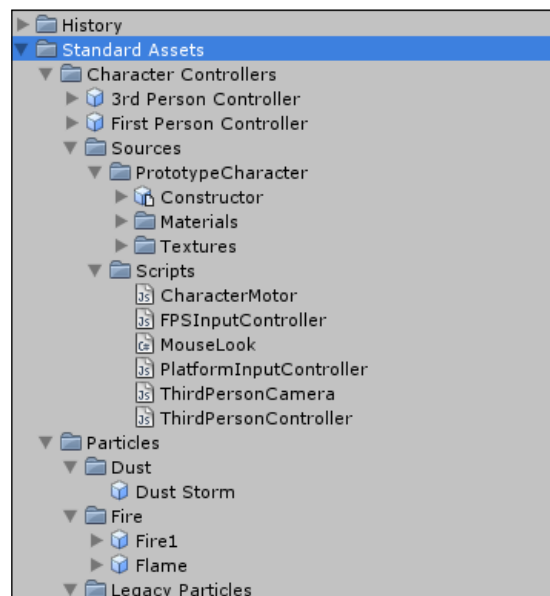
Extract the data and put the package where you would like it to be in your Unity project. To gain access to the package data, open **Unity** and go to **Assets | Import Package | Custom Package...**, as shown in the following screenshot:



Search for the location of your project and open your package. A small interface comes up showing a list of all the assets on the left-hand side and a prompt asking if you would like to install all assets. Click on **All**, as shown in the following screenshot:



This will open up the **Unity\_Scripting** package. The default path for the downloaded assets is **Standard Assets** in the Unity project. If a `Standard Assets` folder does not exist, it will create one and download your package into it.



Congratulations, you have now downloaded and successfully installed the assets required for this book. Now, let's start building!

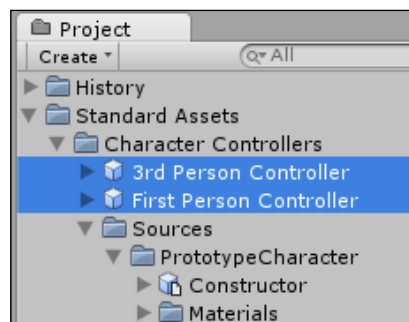
## Getting started with the game

From now on, we will start to script our own game and dive into uncharted depths of JavaScript. The first chapter is dedicated to creating a simple platform game. We will learn to use the built-in functionality of Unity to set up our character, and use the Character Controller component to make that character move and be controlled with our commands. Later in the chapter, we will get into creating a playground for our character. We will also get into teaching him to move boxes around, script moving platforms, create custom triggers, and make huge explosions.

## Available Character Controllers

Now, let's get into the fun part and set up a controllable character. Let's open the project that comes with the book and start coding.

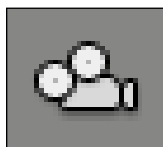
There are two kinds of Character Controllers that are available with a Pro version of Unity3D – **3rd Person Controller** and **First Person Controller**. Default Character Controllers can be found in **Project** view | **Standard Assets** | **Character Controllers**, as shown in the following screenshot. To use any of those Character Controllers, just drag-and-drop them on a scene using the left mouse button. Now, we can click **Play** and start the game, and see our character following orders when we press control buttons.



Now, let's take a look at what these Character Controllers consist of.

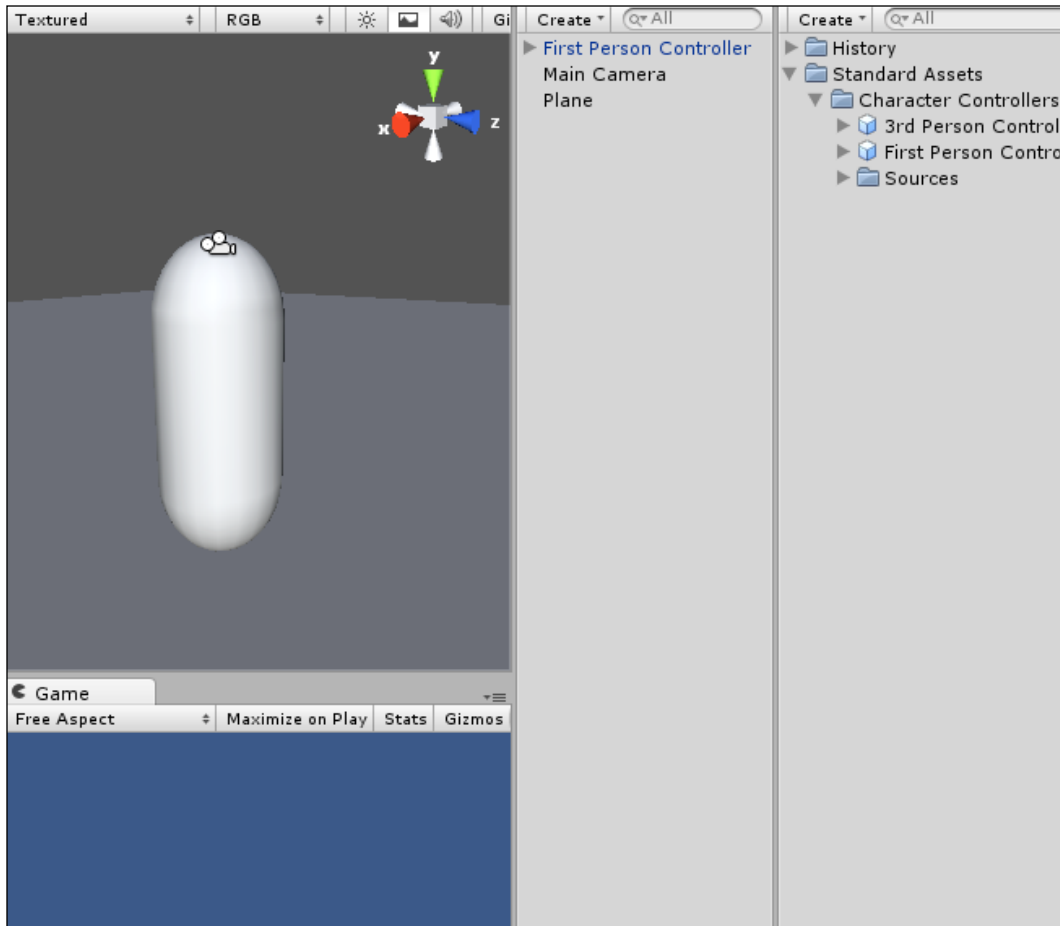
**Character Controller** is a default physics component that does all the necessary collision calculations for us but, at the same time, doesn't follow rules of physics and isn't affected by external forces. However, that doesn't mean that it can't push Rigidbodies if scripted. In general, if we are trying to create a controllable humanoid and don't wish bothering with tons of code, Character Controller will be our best choice. If we are planning to create a character that is being influenced by external forces (like physics) or interacting with objects that are influenced by physics, we will see Character Controller becoming our worst enemy that will break game functionality for no reason. Supplementary to Character Controller are pure physics objects – Rigidbodies. They allow us to create almost anything that is physics related and consist of many hard edges that we will go around in future chapters.

From now on, we will look into both Character Controllers separately and start with First Person Controller. By dragging **First Person Controller** prefab on the screen, we will see a simple cylinder with a camera icon above it. Let's take a look at what's inside:



- **Character Controller:** This is attached to the cylinder with the camera icon above it, at the very top of the list. To attach the Character Controller to the object, select the object, go to **Component** at the top of the screen, and click on **Physics | Character Controller**.
- **Mouse Look (Script):** This handles the camera rotation based on mouse manipulations. This script is written in C# and is beyond this book's scope, but it has a fair amount of description inside, which can be used to tweak mouse controls. To attach a script, go to **Component | Camera Control | Mouse Look**.
- **Character Motor (Script):** This is a script that is responsible for registering all the inputs and controlling **Movement, Jumping, Sliding**, and so on. It is available at **Component | Character | Character Motor**. Some of the functionality can be tweaked from the **Inspector** view, but most of it has been purposely hidden and is accessible only through scripts.

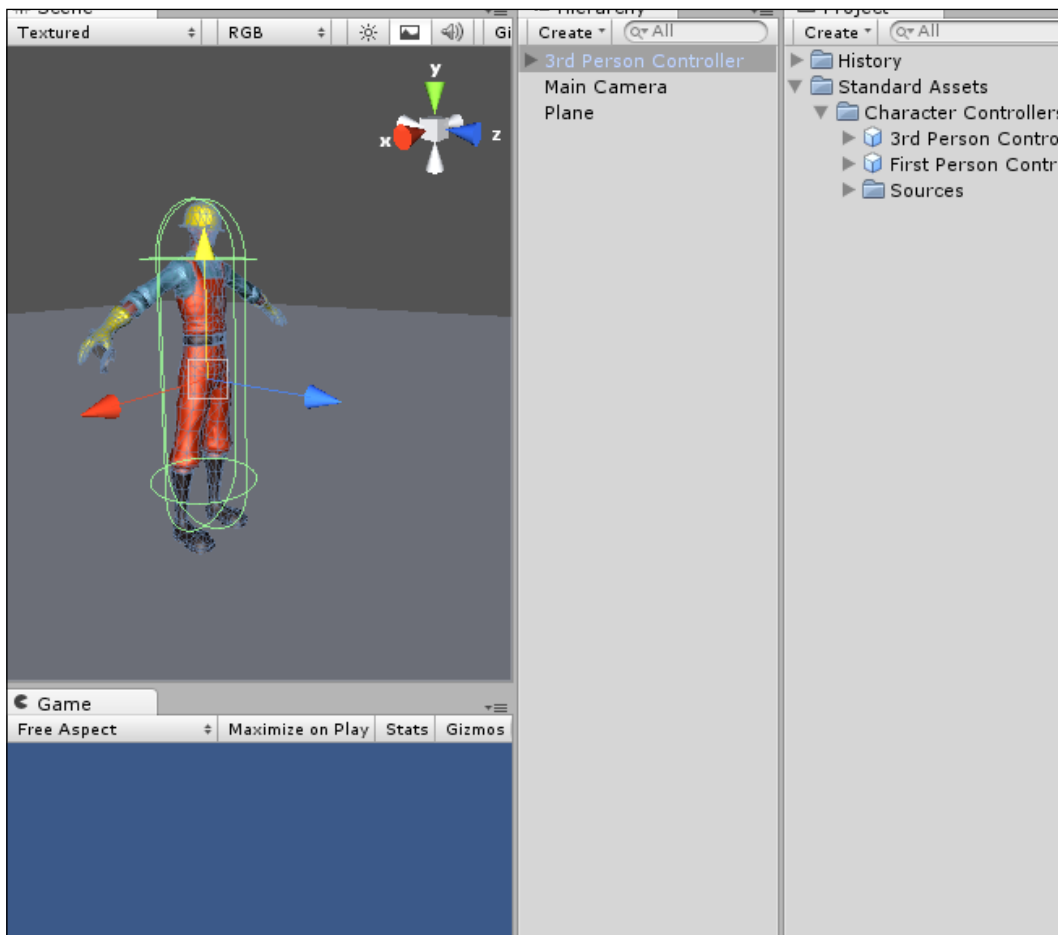
- **FPSInput Controller (Script):** This works together with **Character Motor (Script)**. Its main purpose is to control the functionality of previous scripts (**Component | Character | FPSInput Controller**).



Now that we are done with the **First Person Controller**, let's switch to **3rd Person Controller**. There are few things that make it stand apart. They are as follows:

- **Animation:** Unlike First Person Camera, we are expecting to visually observe our character and watch it playing various types of animations. This is what **Animation** does; we simply attach it to the object (**Component | Miscellaneous | Animation**) and add baked animations to the animation array. The rest is done through code and will be covered in future chapters.

- **Third Person Controller (Script)** and **Third Person Camera (Script)**: They are self-explanatory. The first one controls character, registers inputs from the keyboard, handles animation synchronization, and so on. The latter one adjusts the camera according to character position and actions. Both scripts can be found in **Component | Scripts**.
- **Character Motor (Script)**: This is a script that is responsible for registering all the inputs and controlling **Movement, Jumping, Sliding**, and so on. It is available at **Component | Character | Character Motor**. Some of the functionality can be tweaked from the **Inspector** view, but most of it has been purposely hidden and is accessible only through scripts.
- **FPSInput Controller (Script)**: It works together with **Character Motor (Script)**. Its main purpose is to control the functionality of previous scripts (**Component | Character | FPSInput Controller**).



## Interactive objects

So, you want to interact with objects in the environment now? Interactive objects are usually the objects, which the player has to interact with in order to continue their progression through a level and/or environment. In deciding which interactive items to include as examples, we have chosen to pick objects that show a variety of player interactions. The following is an overview of the type of interactive objects, which will be covered in this chapter:

- Buttons/plunger
- Explosion box
- Moving boxes
- Platform

The list of interactive items can be quite extensive but luckily, once you have thought of the logic behind one, scripting another becomes easier. For a better understanding of the preceding interactive objects, we can split them into two categories – **Triggers** and **Triggered Objects**. TNT plunger, targets, buttons, levers, and volumes fall under the **Triggers** category, whereas TNT box, triggered door, item required/event door, breakable door, and raft fall under **Triggered Objects**. For more information on other interactive items such as pickups, treasure chests, and weapons, see *Chapter 3, Action Game Essentials*. All assets for this chapter can be found in the **History | Resources | Chapter 1** folder.

## Triggers

As stated previously, these objects are used to trigger events in the environment. Through interacting with them, doors can be opened, non-interactive events can be triggered, and enemies can be spawned. These are only a couple of examples of the infinite number of tasks that can be done by interacting with a trigger. Here is a breakdown of the mentioned triggers. Due to the limited number of pages, we will dive right into the description and breakdown of code for each project.

## Buttons

In our case, a button will be described as an object, which the character directly has to interact with in order for it to be triggered. What we will write is a base script, which when used triggers an event. This script, once written, will be used to open a door and explode a box of TNT.

## Base button script

So let's script a button. Go grab the **Button** prefab from the **Chapter 1 prefabs** folder and drag it into the **Hierarchy** view. Once that is done, there will be two game objects in the prefab asset. In **buttonTrigger**, there is a default script on the asset called **Button**.

In the **Start** function of this script, we want to get the initial position of the button.

Declare a variable for initial position, make its type a `Vector3` and default it to `Vector3.zero`. To get the position, have the variable equal to `transform.localPosition` in the **Start** function:

```
var initPos : Vector3;
function Start() {
    initPos = transform.localPosition;
}
```

After the **Start** function, we will create the **MoveButton** function.

## Activating platform status

This next function will move the button to the move position and set the activated status for the platform.

Create a private variable for the button pressed, set its type as `Boolean` and default it to `false`. Inside the **MoveButton** function, create an `if` statement. Have the `if` statement check to see if the button pressed variable is equal to `false`. Inside the `if` statement, we want to send the activation information to triggered object.

To send the information to the appropriate platform in the level, we will have to create a new variable called `Platform`, or something along those lines, with the type of `gameObject` and defaulted to `null`. In the **MoveButton** function, we need to call the **Activated** function in the `platform` script (this script will be created later in this chapter). The following is an example of what it could look like:

```
Platform.GetComponent( platform ).Activated();
```

Now, we need to move the button to give visual indication to the player that the button has been pressed. To get the move position, create another variable for move position, set its type as a `float` and default its value to `0.1` (this value can be adjusted later in the inspector).

```
var movePos : float = 0.1;
```

To move the button from its current position to the new position, we will take the local Z position of the button, subtract the move position value and apply it to the current local position of the button (we will use the Z axis for the example due to the world having Z as depth and the button being mounted on a wall).

The last thing to add to this `if` statement before we close it is to turn the button pressed variable `true`. That's it for this script. We just need to add the collision check function to the built-in Character Controller script and we will have functionality.

Inside of this function, we will do a name check to identify what object the character has collided with. In order to get the name information from the collided object, we have to access the name component, which is a property of `gameObject`. We will then compare this to one that we want, which in this case is `Button`:

```
function OnControllerColliderHit ( Hit : ControllerColliderHit){
    if ( Hit.gameObject.name == "Button" ){
        }
    }
```

If the name matches what we want, we need to access the `MoveButton` function in the `Button` script. To do this, use `GetComponent` to grab the `Button` script and access the desired function. The following statement shows roughly what it should look like:

```
Hit.gameObject.GetComponent("Button").MoveButton();
```

Then in the `if` statement for the detonator plunger, we want to access the `GetPressed` function in the `Button` script.

You have finished writing the base `Button` script. The following is a sample of what that script could look like:

```
var initPos : Vector3;
var movePos : float = 0.1;
var Platform : Transform = null;
var isPressed : Boolean = false;
function Start(){
    initPos = transform.position;
}
function MoveButton(){
    if(!isPressed){
        Platform.GetComponent( platform ).Activated();
        transform.position.z = transform.position.z - movePos;
        isPressed = true;
    }
}
```

Remember that this is a base script and much, much more functionality can be scripted into it.

## Explosion box

It's time to make things explode. Let's script a little bit of explosion box. When the player applies pressure to a detonator box, it triggers the explosion box, and the explosion box explodes! There are just six steps to achieve that, as follows:

1. Prepare objects.
2. Write `Update` function.
3. Write `BOOM` function.
4. Download and install **Detonator** package.
5. Write functionality for button pressing.
6. Preparation.

In this section, we will handle the entire preparation of available resources.



Grab the **Detonator\_Box** prefab out of the Chapter 1 prefabs folder and drag it into the **Hierarchy** view.

If you open the `gameObject` of the detonator box, you will see that it is made up of two pieces – **Detonator\_Box** and **Explosion Box**. We want to drag the **Button** script, made in the last example, to the inspector of the **Detonator\_Plunger** asset located underneath the **Detonator\_Box** group. As the plunger is essentially a button, and the base `Button` script is generic, it can be used for many purposes, such as triggering the explosion box to explode. This script will be the master control for the explosion box as well as the detonator box. It will determine the explosion created when the explosion box explodes, what object is used as the trigger, and the object triggered. You will notice that the prefab parent of **Detonator\_Box** has the **TNT** script in its **Inspector** menu.

## The Update function

The next function that we will write is the `Update` function. In this function, we will do a check for getting the trigger object's pressed status.

First, we have to create a couple of variables – the first one for a trigger and the second one for the explosion box. We want the trigger variable to be of `Transform` type and defaulted to `null` and the `tnt` variable to be of a `Transform` type as well and defaulted to `null`. Create an `Update` function. We will have an `if` statement to make sure that the `tnt` variable has an object associated with it.

To do the trigger check, we will have to write an `if` statement that gets the `Button` script component from the trigger object. To do this, we will have to declare a new variable, make it `public` and call it something along the lines of `trigObj`. We should declare its type as `gameObject`, and default it to `null`.

The value we need for this statement is the `return` function located in the `Button` script. To access this, we get the script component of the trigger object using `GetComponent`. We then declare the script by the name that we wish to access and then the name of the function which has the value to check. The following is an example.

```
function Update() {  
    if( tnt != null ) {  
        If(trigObj.GetComponent("Button").ReturnButtonStatus()) {  
            BOOM();  
        }  
    }  
}
```

As you can see, we have added the `BOOM` function in the name of the next function, which we will be writing.

## The BOOM function

The `BOOM` function will create an explosion at the location of the explosion box and destroy the explosion box game object from the **Hierarchy** view. Before we do anything, let's declare two more variables. The first variable is `explosion` and the second one is `collidedObj`. Make sure that `explosion` is `public`, its type declaration is `Transform`, and it is defaulted to `null`. The `collidedObj` variable should be `private`, and the type declaration should be as a `Collider` array.

In the `BOOM` function, we want to create a collision sphere that will detect all colliders within a given area from a given point. To accomplish this, we will use the `Physics.OverlapSphere` function. Have the `collidedObj` variable equal to the `Physics.OverlapSphere` function with the parameters of the `tnt` variables—`position` for position and the size of the collision sphere set to 1. The following is an example of how it should look:

```
collidedObj = Physics.OverlapSphere(tnt.transform.position, 1);
```

After this, we need to go through the `collidedObj` array and for each object in that array, create an explosion at its position and then destroy the object. To do this use a `for` loop to loop through the array. Call Unity's built-in creation function—`Instantiate` inside of the loop.

The `Instantiate` parameters are the explosion variables, `Obj` in the `collidedObj` array position and then a rotation. The rotation of the current `gameObject` will perform `transform.rotation`. The following is a sample:

```
for (var obj in collidedObj) {
    Instantiate(explosion, obj.transform.position, transform.
rotation);
}
```

Lastly, we will destroy the `gameObject` in the array. To do that, after the instantiation code, type the following line:

```
destroy(obj.gameObject);
```

## Downloading the Detonator package

Now, return to the **Inspector** of **Detonator\_Box**. Under the **TNT** script, you will see the variables that were `public`. These variables are, for example, `trigger`, `explosion`, and `TNT`.

In the `trigger` variable, drag your detonator trigger into it. For the `explosion` variable, we are going to do something different. For the explosion, we will utilize the **Detonator** package that can be downloaded off of Unity's website. You can find it in the **Support | Resources** section at <http://unity3d.com/support/resources/>.

Unity Extensions	Articles
Extend the power of Unity with these ready-made frameworks.	Detailed pieces of knowledge ahead.
 <b>Terrain Toolkit</b> Create good looking, realistic terrains easier.	 <b>Casual Games as a Business</b> Unity is uniquely suited for use as a casual game development tool.
 <b>Explosion Framework</b> Create good looking, scalable explosions in less time.	<a href="#">See all</a>
<a href="#">See all</a>	