

# 1

## Introduction to Unreal Engine 4

Welcome to *Unreal Engine 4 by Example*! During the course of this book you are going to learn how to utilize **Unreal Engine 4 (UE4)** to create high-quality games using C++. This chapter will show you how to install the engine, introduce you to the engine's **User Interface (UI)**, show you how to create new projects, and give you a rundown of what to expect from this book. The purpose of this chapter is to give you a general idea of how this book will be formatted and how to get the most out of your *Unreal Engine 4 by Example* experience.

The chapter will cover the following points:

- Navigating this book and what to expect
- Installing UE4
- Creating new UE4 projects
- Navigating UE4's User Interface
- Creating a basic actor and adding components
- Opening and exploring the examples provided by Unreal

### Navigating this book

First off, it should be stated that the use of C++ features heavily in the later stages of this book. It is strongly advised that you already have some experience with writing in C++ or a similar low-level language before embarking on this journey.

This is a *By Example* book, meaning that you will be taught how to use UE4 via the breakdown and explanation of previously created UE4 game projects. These projects have been created specifically for this book with your education in mind! Each project will be taught over two chapters, the first will include the core concepts and features needed to implement the title. The second will delve into the advanced or more complicated components of each project. By the end of each project you will have learnt core UE4 functionality plus some very interesting polish techniques.

By the end of this book you should find yourself well on your way to becoming a skilled Unreal Engine developer capable of using C++ and the provided Blueprint system to utilize Unreal Engine's rich feature set to create polished game titles.

## Game Projects you say?

Yes, game projects! By the end of this book you will have created four unique game projects that will all include their own polish features. These projects culminate with the creation of your very own multiplayer first-person shooter!

During the course of this book you will create the following game projects:

- **Barrel Hopper:** A homage to the original Donkey Kong, this project will be created entirely using Blueprint and will introduce you to the high-level systems used when creating games with UE4.
- **Bounty Dash:** Our take on an endless runner, this project will be your first look into C++ with UE4. We will be expanding on some of our content created for Barrel Hopper and really start to unlock the potential of UE4.
- **Boss Mode:** For this project we will be expanding on your C++ technical skill set by creating an engagement with a game boss controlled by **Artificial Intelligence (AI)**, which will be visually detailed by taking advantage of UE4's in-depth rendering and shading functionality.
- **Network Shooter:** This will be the culmination of your journey with this book and your UE4 education. You will be creating a networked first-person shooter that lets you create a game you can play with your friends through a Local Area Network.

## Installing Unreal Engine 4

The version of Unreal Engine we will be using for this book is Unreal Engine 4.11.0. You may use any 4.11.# engine version as long as your version number exists within the 4.11 family. If you have already installed Unreal Engine 4.11 you can skip this section and jump straight to *Creating your first project*. First things first, you need to sign up as an Unreal Engine developer. Navigate to <https://www.unrealengine.com> in your browser and select the **GET UNREAL** option located in the top right-hand corner of the web page.



This will then redirect you to the login/sign-up page for a UE account. If you do not have an account, follow the steps presented in the page to create one. This account will integrate you into the Unreal Development community. Through this account you can get Unreal Engine news, updates, and developer support.

Once this is done you will be directed to the downloads page, choose the OS that you will be developing on (MAC, Windows) and you should see the download begin for an Epic Games Launcher installer. The Epic Game Launcher is the portal for all Unreal Engine versions and you will use this application to install, manage, and launch the engine version you wish to use.


Once the Epic Games Launcher has been successfully installed, run it and it will prompt you to log in with the account credentials you previously set up with Epic. If everything has gone according to plan you should be presented with the Unreal Engine launcher. From the launcher you will need to navigate to the **Unreal Engine** tab located in the top left-hand side of the window. From here you can see **News**, **Learning Resources**, and **Marketplace Content** for the engine. I advise you to take your time to explore each of the options available to you, however we will be focusing on the **Library** section.

By selecting the **Library** element on the list on the left-hand side of the window you will be presented with any currently installed engines as well as any Unreal projects you have already created. There will be an **Add Versions** button near the top of the window.



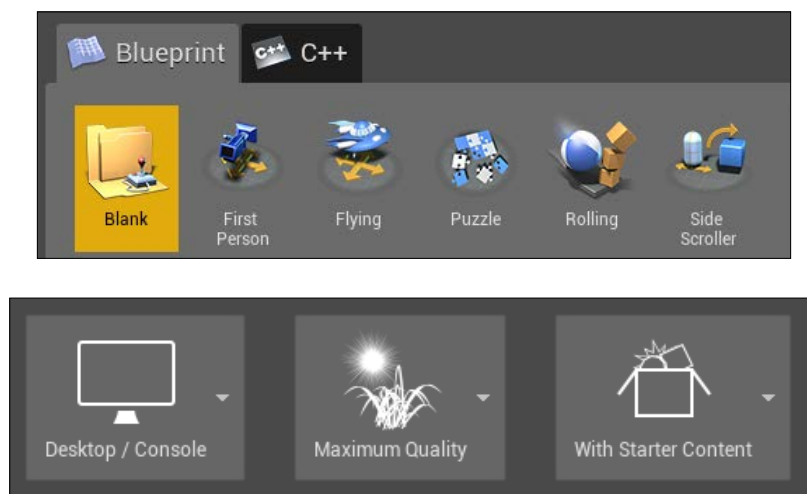
Pressing this button will present you with a light-colored engine node, click the drop-down arrow next to the engine version number and be sure to select 4.11.0. Once this is done, hit **Install**. The launcher will now handle downloading and installing your new engine version!

While the engine is installing I would strongly suggest you spend some time exploring <https://www.unrealengine.com>. The website is a great place to find video tutorials, learning resources, and the Unreal Engine Wiki. You do not need to utilize these resources to follow this book, however the content they have provided is very informative and will definitely help you on your journey to becoming a proficient Unreal Engine developer. Once the engine has been installed, hit **Launch** and you are ready to go!

[  Tip: If you hit the drop-down arrow next to launch you can create a shortcut for the engine version. ]

## Creating your first project!

It is now time for the famous Hello World, but this time it will be Hello Unreal. With your engine version launched you will be presented with the **Unreal Project Browser**. This browser features two tabs, **Projects** and **New Project**, select **New Project**. The first thing you can see in this tab is a large selection of icons under two more tabs, **Blueprint** and **C++**. We will be selecting the **Blueprint** tab and we will be creating a **Blank** project, your selection should appear as follows:



Under this section you will see three dropdown boxes titled **Desktop / Console**, **Maximum Quality**, and **With Starter Content**. We will leave these as they are for now but this is where you can swap your target platform for the project, change the graphics quality, and whether or not you would like to use the starter content provided by Epic. Below these options you will see fields to populate the directory where you wish to create the project and what the name of the project will be. We shall title this project `Hello Unreal`. Once this is done, click the green **Create Project** button and the engine will finally launch.

## Navigating the Unreal Engine UI

Welcome to the Unreal Engine User Interface! It can be a bit overwhelming at first but, fear not, all will be explained. By the end of this book you will be very familiar with the layout and workings of this engine view. Now, for the sake of brevity, I will hand your Unreal education over to the Epic team for a short moment. In the top right-hand corner of the window next to the name of the project you should see a grey academia cap flashing green.



Clicking this cap will start a short tutorial that will familiarize you with the engine's layouts and panels. I am leaving this up to Unreal as the introductory tutorials are succinct and efficient at bringing you up to scratch with the editor layout and camera controls. This gives us more time to get to the juicy learning!

Once you have completed this introductory tutorial you will be guided to the tutorial home panel. Here you will be able to find a list of simple tutorials that will help familiarize you with the different features of the engine, could you please complete the two tutorials under the **Basics** category. Feel free to utilize the other tutorials during the course of the book. I will likely be covering almost all of the topics covered in the tutorials but they may prove useful as an additional learning resource. Now that Epic has taught you how to view its engine, it is now time to teach you how to use the engine!

## Creating a basic actor


At this point you should have a world that looks similar to this:



What we are going to do now is remove the table, statue, and chairs from our level and replace them with our own actor. The purpose of this is to show you how we can create new actors and add components from within the editor without having to navigate to other areas of the engine. You can remove actors by clicking on them and hitting the *Delete* key, right-clicking on them and selecting **Edit | Delete** from the dropdown menu, or by finding the actor in the **World Outliner** and performing the same steps.

Once the table and chairs have been removed, we need to add an actor to the world that we wish to build upon. To do this we need to place an element from the **Modes Panel** in the top left-hand side of the window. As stated in the introductory tutorial, the **Modes Panel** is where you can find all of the tools that you will need to perform construction actions within the editor. The five tabs you can interact with are **Place**, **Paint**, **Landscape**, **Foliage**, and **Geometry Editing**.

These options will be explored during the course of this book but, for now, we are concerned with the **Place** tab. The **Place** tab is where you can find all of the editor objects that can be dragged and dropped into the level. Here you can find things such as basic geometry primitives, BSPs, visual effects, and numerous processing volumes.

[  If you wish to see detailed documentation for each of the editor objects, click the small question mark on the right-hand side of the object name. ]

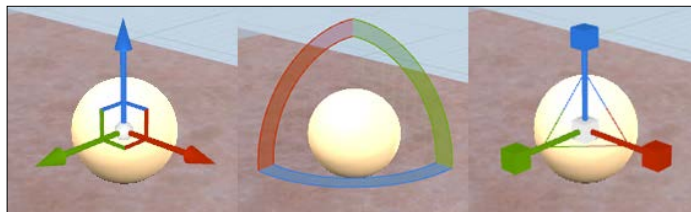
From the list on the left-hand side of the panel, select **Basic**; this will show you a selection of simple objects that can be dragged into the world. We want to drag an **Empty Actor** into the world.



## Objects, Actors, Pawns, and Characters

Unreal Engine utilizes a specific naming convention for the objects that can be created and built upon within the editor. There are **Objects**, **Actors**, **Pawns**, and **Characters**. Each name represents an expanded level of functionality. In general, you can think of Actors as whole items or entities such as a power up or boost pad, where an object is usually a small part or **Component** with specialized functionality. For example, a whole car could be an Actor but the engine within the car would be considered an Object. Pawns and Characters I will detail in later chapters. The term object(s) will be used to refer to things in our levels but when required I will use the capitalized Actor or Object when specification is required.

Select our new actor now by clicking on it within the viewport. With the **Empty Actor** selected in our world we are presented with a few new visual elements. The first is the transform widget; the transform widget provides us with the ability to translate, rotate, and scale objects within our world. You can swap the functionality of your transform widget by pressing the *W* key for translation, the *E* key for rotation, and the *R* key for scale. Alternatively, in the top right-hand side of the **Viewport** panel you can click and select the appropriate transform icon. We can also select the globe icon in the same corner to swap between world and local space transformation. Simply put, world space transformation will present a transform axis in line with the world. Local space will present a transform axis in line with the object (based on the objects arbitrary forward).





If you are ever curious to see the world space axis, it can be found in the bottom left-hand side of the screen, denoted by a simple line widget. This can be used to orient yourself while using the 3D camera/building levels.

## Adding components to your Actor

You will notice that, if you try to scale or rotate the empty actor, nothing happens. That is because, at this point, the empty actor only contains a `DefaultSceneRoot` component. A component is an Object that is owned by other Unreal Engine 4 objects (usually Actors, Pawns, or Characters) that provide different functionalities/features to the object. In this case our actor owns a `DefaultSceneRoot` component allowing us to translate, rotate, and scale the object.

Why then does scaling and rotating do nothing? Well, right now we have no 'visual' components included in our actor and by that I mean any components that contain geometry that will visually represent the object in the game world. Without a visual reference, the orientation or size of the empty actor has no context, only the location of the object in 3D space could be of any use, thus we have the ability to translate.

This brings me to the second visual element; on the right-hand side of the window is the **Details** panel. This panel is used to show all information about any selected object within our world. From the **Details** panel we can manipulate exposed member data, add and remove components, and edit component properties. There are a few other tricks the **Details** panel offers us but I will expand upon those later on in the book.

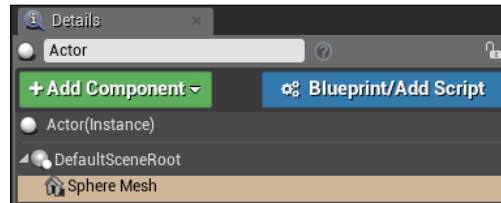
You can also change the name of objects in the details panel. Right now our object is called **Actor**, we should change this to a more specific name, let's use `Hello Sphere`. Now we need to add a visual reference to our `Hello Sphere`. Under the **Details** panel you can see a green **Add Component** button; clicking this button will bring up a list of pre-built components we can add to our actor. We would like to add a **Sphere** to our actor, it can be found under the **Basic Shapes** section of the dropdown. You can either use the search bar provided to find specific components or scroll down the list until you find what you require.

## The component hierarchy

Once you have added the sphere mesh to our `Hello Sphere` actor you will see a new element in the **Details Panel, component hierarchy**. We still have our `DefaultSceneRoot` component but now, directly under it, we have our sphere mesh. This brings up a very important point: Component parenting.



Within the **Details** panel you will notice that the sphere mesh component is indented slightly and directly beneath our `DefaultSceneRoot` component.



This is called parenting; our sphere mesh is now a child component of our `DefaultSceneRoot` component. This means that any scaling, rotation, or translation we apply to our `DefaultSceneRoot` will be applied to our sphere as well. However, any transformations we enact on our sphere mesh will not be applied to our `DefaultSceneRoot`. You can change the hierarchy of an object's components by clicking and dragging them up and down the list. The top-most component will be the Root Component and will dictate the transform of all subsequent components.

Ignoring this can lead to disjointing of child components and can be fairly troublesome. I encourage you to now translate the `Hello Sphere` with the `Sphere` component selected in the details panel, then again with the `DefaultSceneRoot` component selected instead. You will notice that, upon the second translation, the `Sphere Mesh` maintains its relative distance from the location of the `DefaultSceneRoot` component. It is because of this that I would strongly advise to only apply transformations to child objects when you wish those transformations to remain constant, relative to the parent.



If you wish to move everything within an object instance you may select the `ObjectName (Instance)` element within the hierarchy.

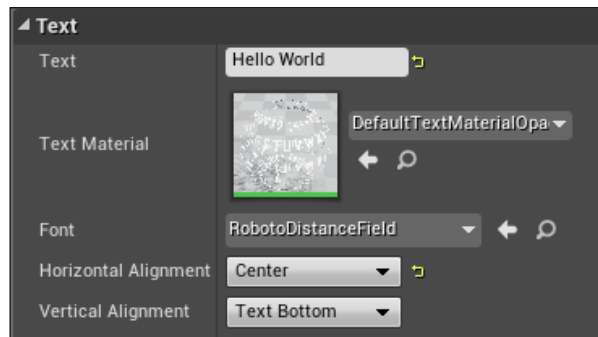
## Modifying components

Now translate the `Hello Sphere` to the same location that the table and chairs originally occupied. We are going to add some `Flash` to our `Hello Sphere`. We are going to add two more components to the actor; a **Text Render** component and a **Particle System** component. We will then use the **Details** panel to modify the properties of these components to achieve our desired visual result.

First, let's modify the text. After adding a Text Render component, select it in the component hierarchy. You will see that the bottom section of the **Details** panel will change. This section is where you can modify the component's properties. You will notice that you can also get access to the component's transform properties here, this is so you can perform exact transformation changes to an object without having to use the transform widget.

Underneath the **Transform** and **Materials** sections we will see a section titled **Text**. It is here that the main properties for our Text Render component exist. The first thing we need to do is change the text so it says something meaningful. Let's change the **Text** field to Hello World then change the **Horizontal Alignment** field to Center.

Your Text section of the details panel should look something like this:

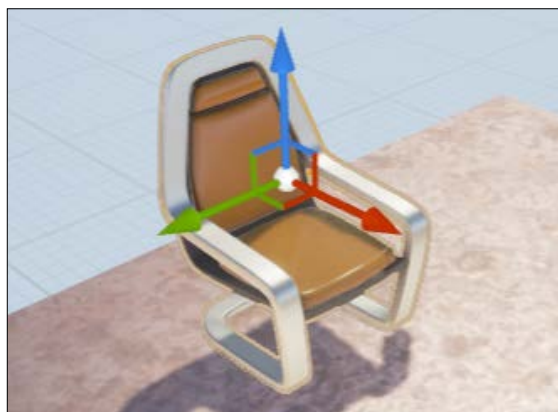


## 3D transformation and axis

Unfortunately the 3D text is partially penetrating the sphere mesh. To fix this we will be applying a translation to the Text Render component. We wish to apply this translation to the Text Render child component as we want this translation to remain the same regardless of the position of our actor. Instead of using the widget we can use the Transform section of the **Details** panel. You will notice a 3x3 field matrix under the Transform section. The top row represents the location of the component in 3D space, the second row the rotation, and the bottom row the scale. We want to move our 3D text vertically by a small amount and reduce the scale of our sphere so that our text sits above our sphere mesh.

3D game engines will use an arbitrary 3D co-ordinate system to accurately gauge the transformation of objects in 3D space. These co-ordinate systems are made up of three axis—X, Y, and Z. We gauge the direction of the co-ordinate axis by the direction in which the axis increases in a positive manner.

To better understand how axis work, we will take your position as a developer as a reference. If you were to see your own Unreal axis set, you would see a red axis (the X axis) extend outwards from your chest towards the screen. This is because the X axis in Unreal Engine increases in a positive manner forwards into the screen. In a similar fashion you would see a green axis extend out of your right-hand side (the Y axis), and finally you would have a blue axis extending upwards from your head (the Z axis). As an example, imagine the chair in the following screenshot is the chair you are sitting in:



We need to translate the text component 30cm positively along the Z axis. To do this, ensure that you have the Text Render component selected in the component hierarchy and simply change the third field in the top row of the **Transform** section of the **Details** panel to 30.0. However, our text render component is still intersecting with the sphere, we now need to scale down or shrink our sphere mesh component.

To scale down the sphere, make sure the Sphere Mesh component is selected in the component hierarchy, then press *R* to bring up the Scale Transform widget. You'll notice, as you hover your mouse over the different sections of the widget, that they will change to a yellow color. This means that if you were to press and hold the left mouse button, any movement of the mouse will result in a scale change on that axis. To scale the sphere along all axis uniformly, there is a small white box in the widget where all the axis meet. If your mouse is situated over this box, all of the axis will change to a yellow color. With this white box selected, click and drag the mouse to scale the sphere to half the size it was before. Or, alternatively, change all of the values in the bottom row of the transform matrix to 0.5.

This will move our text render component up and out of our smaller sphere mesh, resulting in something that looks like this:



## Adding the Unreal factor

Our little sphere has come a long way, however we need to add something to our actor to make it stand out. This brings our other new component into focus, the **Particle system**. Particle systems are components that allow us to attach previously created particle effects to our actors. The creation of these particle effects and a more in-depth description will be detailed in the *Boss Mode Activated* chapter.

For now, we can use the new component's **Details** panel to modify which particle effect we would like to use and how it is transformed. With the Particle System component selected in the hierarchy, select the **Template** dropdown field in the **Particles** section and select `P_Fire`. This will use the `P_Fire` particle system template for the particle component. You may notice that the particle system is quite large initially; I scaled the system to about 1/3 of the original size. You can do this by using the scale widget or the transform field in the **Details** panel with the particle system component selected.

We are not done yet, Unreal Engine boasts a very flexible and feature-rich renderer. One of the most commonly used features is the **Material system**. Materials are pre-built visual effects we would like to apply to our surfaces. Materials encompass all surface details such as textures, color, and physical properties such as roughness and luminescence. The creation of Materials along with an in-depth description is covered in the *Boss Mode Activated* chapter. For now we can use a pre-made material for our Sphere Mesh. With the Sphere mesh component selected, click the dropdown field under the **Materials** section of the **Details** panel. Search in the list for `M_Metal_Burnished_Steel`. You'll notice that when the material is selected, the appearance of the sphere will change. You now see something similar to this:

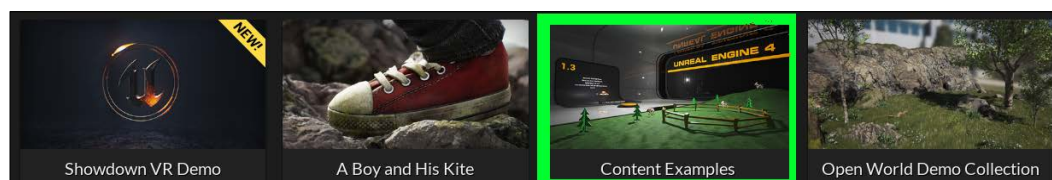


And that is it! We are done setting up our game world for the Hello World project. Now you can click the large play button in the **Toolbar** at the top of the Editor window and fly around your new game world! You will notice at the top of the screen there are a few options, these in-editor play options will be explained in the next chapter *Blueprints and Barrels*. Congratulations on making your first Unreal Engine project `Hello Unreal!` All of the concepts presented in this project will be expanded on in much more detail in the following chapters. What you have learned here is only the beginning; there are many more interesting features and techniques yet to come.

## Pre-Built projects as a learning resource

Now that you are comfortable creating and modifying your own projects, it is important to learn how to interact with the Unreal Learning tools to download pre-built projects so you may learn from other people's work.

Epic releases example projects that you can navigate and explore to learn about the different faces of the engine. If you open the **Epic Games Launcher** application and return to the **Unreal Engine** tab, you will notice in the list on the left-hand side of the screen a **Learn** element. Selecting this element will open up the **Learn** page of the **Epic Game Launcher**. From this page you can get access the engine documentation, video tutorials, community wiki and, most importantly, the example projects. If you scroll down, you will see a section titled Engine Feature Samples. This section holds all of the example projects that you can download to see how Epic Games implements the different feature sets of the Unreal Engine. The one in particular I would like to draw your attention to is the **Content Examples** project.



The **Content Examples** project features a set of levels that demonstrate the use of many of the Engine's features. If at any point the uses of the Engine's components seem unclear, you may be able to find an example of that component's implementation in this project. Once the project has been downloaded it will appear in the **Library** section of the **Unreal Engine** tab.

## Summary

Well done on taking your first steps towards becoming an Unreal Engine developer! I hope you look forward to learning from this book and continuing your game development adventures with Unreal Engine 4. In this chapter you have learned how to install the engine, navigate its unique interface, and you have made your first Unreal Actor! The next chapter *Barrels and Blueprints* will expand on the skills you just have learned to create your first UE4 game project! You will be recreating a version of the classic arcade game Donkey Kong©. The chapter will have you learn more about interfacing with the engine editor, how to create a workflow with Unreal Engine, and how to work with the Blueprint visual scripting system.