DOWNLOAD PDF INTRODUCTION TO UNREAL ENGINE 4

Chapter 1 : An Introduction to Unreal Engine 4 by Andrew Sanders

Introduction and Project Overview Hi, I'm Josh with Digital Tutors, and welcome to Introduction to Unreal Engine 4. In this tutorial, we'll learn the basics of working in the Unreal Editor. In this tutorial, we'll learn the basics of working in the Unreal Editor.

If you are coming in with no programming experience at all, we have got you covered also! Check out our Blueprint Visual Scripting guide and you will be on your way. You can create entire games using Blueprint scripting! We will talk more about that as we go along. The gameplay API and framework classes are available to both of these systems, which can be used separately, but show their true power when used in conjunction to complement each other. What does that really mean, though? In this case, we are going to create a class that is later extended via Blueprints by a designer or programmer. In this class, we are going to create some properties that the designer can set and we are going to derive new values from those properties. The second step in the process tells the wizard the name of the class you want generated. Once you choose to create the class, the wizard will generate the files and open your development environment so that you can start editing it. Here is the class definition that is generated for you. For more information on the Class Wizard, follow this link. BeginPlay is an event that lets you know the Actor has entered the game in a playable state. This is a good place to initiate gameplay logic for your class. Tick is called once per frame with the amount of elapsed time since the last call passed in. There you can do any recurring logic. However if you do not need that functionality, it is best to remove it to save yourself a small amount of performance. If you remove it, make sure to remove the line in the constructor that indicated ticking should occur. The constructor below contains the line in question. You can turn this off to improve performance if you do not need it. There are more ways to control how and where it is edited. For instance, if you want the TotalDamage property to appear in a section with related properties, you can use the categorization feature. The property declaration below shows this. This is a great way to place commonly used settings together for editing by designers. Now let us expose that same property to Blueprints. There are quite a few options available for controlling how a property is exposed to the engine. To see more options, follow this link. Before continuing to the section below, let us add a couple of properties to this sample class. There is already a property to control the total amount of damage this actor will deal out, but let us take that further and make that damage happen over time. The code below adds one designer settable property and one that is visible to the designer but not changeable by them. The VisibleAnywhere flag marks that property as viewable, but not editable in the Unreal Editor. The image below shows the properties as part of the class defaults. Below are two examples of setting default values in a constructor and are equivalent in functionality. In order to support per instance designer set properties, values are also loaded from the instance data for a given object. This data is applied after the constructor. You can create default values based off of designer set values by hooking into the PostInitProperties call chain. Even though these are designer specified, you can still provide sensible default values for them, as we did in the example above. If you do not provide a default value for a property, the engine will automatically set that property to zero or nullptr in the case of pointer types. There are two ways to do this: With the editor still running, go ahead and build from Visual Studio or Xcode like you normally would. The editor will detect the newly compiled DLLs and reload your changes instantly! You can use this feature in the sections below as we advance through the tutorial. Let us now take a look at how a designer can start creating unique classes from our humble beginnings here. First thing we are going to do is create a new Blueprint class from our AMyActor class. Notice in the image below that the name of the base class selected shows up as MyActor instead of AMyActor. This is intentional and hides the naming conventions used by our tools from the designer, making the name friendlier to them. Once you choose Select, a new, default named Blueprint class is created for you. In this case, I set the name to CustomActor1 as you can see in the snapshot of the Content Browser below. This is the first class that we are going to customize with our designer hats on. First thing we are going to do is change the default values for our damage properties. In this case, the designer changed the TotalDamage to and the time it takes to deliver that damage to 2 seconds. This is how the

properties now appear. Our calculated value does not match what we would expect. It should be but it is still at the default value of The reason for this is that we are only calculating our damage per second value after the properties have been initialized from the loading process. Runtime changes in the Unreal Editor are not accounted for. There is a simple solution to this problem because the engine notifies the target object when it has been changed in the editor. The code below shows the added hooks needed to calculate the derived value as it changes in the editor. This is so that building your game only the code that you need for the game, removing any extra code that might increase the size of your executable unnecessarily. Now that we have that code compiled in, the DamagePerSecond value matches what we would expect it to be as seen in the image below. Let us start by first making the CalculateValues function callable from Blueprints. Exposing a function to Blueprints is just as simple as exposing a property. It takes only one macro placed before the function declaration! The code snippet below show what is needed for this. Every Blueprint exposed function requires a category associated with it, so that the right click context menu works properly. The image below shows how the category affects the context menu. As you can see, the function is selectable from the Damage category. The Blueprint code below shows a change in the TotalDamage value followed by a call to recalculate the dependent data. This uses the same function that we added earlier to calculate our dependent property. We often use the approach to notify the designer of an event that they can respond to as they see fit. Often that includes the spawning of effects or other visual impact, such as hiding or unhiding an actor. The code snippet below shows a function that is implemented by Blueprints. This is commonly referred to as a Thunk. The code snippet below shows the changes needed in the header to achieve this. So how do you provide the default implementation? You must provide this version of the function or your project will fail to link. Here is the implementation code for the declaration above. In any version 4. Now that we have walked through the common gameplay programmer workflow and methods to work with designers to build out gameplay features, it is time for you to choose your own adventure. Diving Deeper I see you are still with me on this adventure. The next topics of discussion revolve around what our gameplay class hierarchy looks like. Objects, Actors, and Components There are 4 main class types that you derive from for the majority of gameplay classes. Each of these building blocks are described in the following sections. Of course, you can create types that do not derive from any of these classes, but they will not participate in the features that are built into the engine. Typical use of classes that are created outside of the UObject hierarchy are: This class, coupled with UClass, provides a number of the most important base services in the engine: Reflection of properties and methods Serialization of properties.

Chapter 2 : Unreal Engine 4 Blueprints Tutorial | www.nxgvision.com

Welcome to Unreal Engine 4 Game Development www.nxgvision.com this chapter, you will learn how to download Unreal Engine's source version and launcher version. After that, we will get familiar with the Unreal Engine 4 UI and Content Browser.

To open the project, go to the project folder and open BananaCollector. You can either choose the option to open a copy, or the option to convert in place. You will see the scene below. This is where the player will move around and collect the items. I have categorized the project files into folders for easier navigation, like you see here: Click the Add New button and select Blueprint Class. Since you want the actor to be able to receive inputs from the player, the Pawn class is fitting. The Character class would also work. It even includes a movement component by default. However, you will be implementing your own system of movement so the Pawn class is sufficient. You will create a camera that looks down towards the player. To create a camera, go to the Components panel. Click Add Component and select Camera. For the camera to be in a top-down view, you need to place it above the player. With the camera component selected, go to the Viewport tab. Activate the move manipulator by pressing the W key and then move it to , 0, Alternatively, you can type the coordinates into the Location fields. It is located under the Transform section in the Details panel. If you have lost sight of the camera, press the F key to focus on it. Next, activate the rotation manipulator by pressing the E key. Rotate the camera down to degrees on the Y-axis. Representing the Player A red cube will represent the player so you will need to use a Static Mesh component to display it. First, deselect the Camera component by left-clicking an empty space in the Components panel. Click Add Component and select Static Mesh. To display the red cube, select the Static Mesh component and then go to the Details tab. Click Compile and go back to the main editor. Spawning the Player Before the player can control the Pawn, you need to specify two things: The Pawn class the player will control Where the Pawn will spawn You accomplish the first by creating a new Game Mode class. For example, in a multiplayer game, you would use Game Mode to determine where each player spawns. More importantly, the Game Mode determines which Pawn the player will use. Go to the Content Browser and make sure you are in the Blueprints folder. Now, you need to specify which Pawn class will be the default. Go to the Details panel and look under the Classes section. Before you can use your new Game Mode, the level needs to know which Game Mode to use. You can specify this in World Settings. Click Compile and close the Blueprint editor. Each level has their own settings. A new World Settings tab will open next to the Details tab. Finally, you need to specify where the player will spawn. You do this by placing a Player Start actor into the level. If the Game Mode finds one, it will attempt to spawn the player there. Left-click and drag Player Start from the Modes panel into the Viewport. Releasing left-click will place it. You can place this wherever you like. You will spawn where you placed the Player Start. To exit the game, click the Stop button in the Toolbar or press the Esc key. Your next task is to set up the input settings. Setting Up Inputs Assigning a key to an action is called key binding. In Unreal, you can set up key bindings that will trigger an event when you press them. Events are nodes that execute when certain actions happen in this case, when you press the bound key. When the event is triggered, any nodes hooked up to the event will execute. This method of binding keys is useful because it means you do not have to hard code keys. For example, you bind left-click and name it Shoot. Any actor that can shoot can use the Shoot event to know when the player has pressed left-click. If you want to change the key, you change it in the input settings. If you had hard coded it, you would have to go through each actor and change the keys individually. On the left, select Input under the Engine section. The Bindings section is where you will set up your inputs. Unreal provides two methods to create key bindings: These can only be in two states: Action events will only trigger once you press or release the key. These output a numerical value called an axis value more on that later. Axis events will fire every frame. Generally used for actions that require a thumbstick or mouse. For this tutorial, you will use axis mappings. Creating Movement Mappings First, you will create two axis mapping groups. Groups allow you to bind multiple keys to one event. Create two groups and name them MoveForward and MoveRight. MoveForward will handle moving forward and backwards. MoveRight will handle moving left

and right. You will map movement to four keys: Currently, there are only two slots to map keys. To map a key, click the drop-down to bring up a list of keys. Map the W and S keys to MoveForward. Map the A and D keys to MoveRight. Next, you will set the Scale fields. Axis Value and Input Scale Before you set the Scale fields, you need to learn about how they work with axis values. An axis value is a numerical value that is determined by the type of input and how you use it. Buttons and keys output 1 when pressed. Thumbsticks output a value between -1 and 1 depending on the direction and how far you push it. For example, if you push the thumbstick to the edge, the axis value will be 1. If you push it halfway, it will be 0. By multiplying the axis value with a speed variable, you can adjust the speed with the thumbstick. You can also use the axis value to specify a direction along an axis. Using a negative axis value will result in a negative offset. Since keyboard keys can only output an axis value of 1 or 0, you can use scale to convert it to a negative. It works by taking the axis value and multiplying it by the scale. If you multiply a positive the axis value with a negative the scale , you will get a negative. Set the scale of the S and A keys by clicking on the Scale field and entering Next, comes the fun part: Moving the Player First, you need to get the events for your movement mappings. Right-click an empty space in the Event Graph to get a list of nodes. From the menu, search for MoveForward. Add the MoveForward node listed under Axis Events. Repeat the process for MoveRight. Now, you will set up the nodes for MoveForward. Using Variables To move, you need to specify how fast the Pawn is moving. An easy way to specify the speed is by storing it in a variable. With your new variable selected, head over to the Details tab.

Chapter 3 : Video: Introduction to C++ for Unreal Engine 4 - Tom Looman

This book serves as an introduction to the level design process in Unreal Engine 4. By working with a number of different components within the Unreal Editor, readers.

Chapter 4 : An Introduction to Unreal Engine 4 | PDF Free Download

Unreal Engine 4 Beginner Tutorial Series - #1 Series Introduction This is episode 1 of my Unreal Engine 4 Beginner Tutorial Series, In this video I will be going over an introduction to the unreal.

Chapter 5 : Skillshare â€“ Introduction To Character Animation In Unreal Engine 4 | CG Persia

Once you have signed in the launcher's window will load and the first thing that you will see, under the Unreal Engine tab [1], is a huge Launch button with the selected default Unreal Engine 4 version[2] and below that the Community section [3] with all the latest news, tutorials and community spotlights.

Chapter 6 : Skillshare - Introduction To Character Animation In Unreal Engine 4 - GFXDomain Blog

Recently a new version of Unreal Engine, that being , was released! This version comes with an insane about of new features which will make your work a lot more easier!

Chapter 7 : Introduction to C++ Programming in UE4

This book serves as an introduction to the level design process in Unreal Engine 4. By working with a number of different components within the Unreal Editor, readers will learn to create levels using BSPs, create custom materials, create custom Blueprints complete with events, import objects.

Chapter 8 : Introduction to Unreal Engine 4 - Unreal Engine 4 Game Development Essentials [Book]

Unreal Engine 4 Sequencer Next up is the Unreal Engine Sequencer, which is very similar to a non-linear editor. We will

set up your scene, complete with spaceship interior, Exile alien characters, lights, smoke effects and cameras and get it ready for animation.

Chapter 9 : An Introduction to Unreal Engine 4 - CRC Press Book

Introduction to Unreal Engine 4 9 torrent download locations www.nxgvision.com Introduction to Unreal Engine 4 DigitalTutors p MP4 Other Other 1 day www.nxgvision.com Introduction to Unreal Engine 4 DigitalTutors p MP4 Other.