

03 Gameplay Scripting

Contents

[Overview 4](#_Toc508201207)

[Introduction 5](#_Toc508201208)

[What Is Blueprint? 5](#_Toc508201209)

[How Do Blueprints Work? 5](#_Toc508201210)

[3.1. The Event Graph 6](#_Toc508201211)

[3.2. Basic Scripting 6](#_Toc508201212)

[Variables 7](#_Toc508201213)

[Events and Flow 7](#_Toc508201214)

[Nodes, Pins, and Wires 7](#_Toc508201215)

[3.3. Game Loops 7](#_Toc508201216)

[Delta Seconds 8](#_Toc508201217)

[Demonstration: Simple Day/Night Cycle That Rotates an Actor on Tick 8](#_Toc508201218)

[3.4. Construction Scripts 8](#_Toc508201219)

[Demonstration: Simple Generation of Assets in a Line 8](#_Toc508201220)

[3.5. Call in Editor 8](#_Toc508201221)

[Demonstration: Simple Button to Randomize an Actor’s Color 8](#_Toc508201222)

[3.6. Debugging 8](#_Toc508201223)

[Demonstration: Adding a Breakpoint 9](#_Toc508201224)

[Demonstration: Watching the Flow 9](#_Toc508201225)

[3.7. Framework Overview 9](#_Toc508201226)

[Object 9](#_Toc508201227)

[Actor 9](#_Toc508201228)

[Pawn 9](#_Toc508201229)

[Character 10](#_Toc508201230)

[Game Mode 10](#_Toc508201231)

[Game State 10](#_Toc508201232)

[Game Instance 10](#_Toc508201233)

[3.8. Bird of Prey Project Overview 10](#_Toc508201234)

[Exercises 12](#_Toc508201235)

[Exercise 3A: Rebalancing a Game 12](#_Toc508201236)

[Exercise 3B: Adding Additional Content 14](#_Toc508201237)

[Multiple-Choice Quiz 15](#_Toc508201238)

# Overview

**PURPOSE**

The purpose of this tutorial is to introduce the Blueprints Visual Scripting system.

**SCOPE**

This section will provide an introductory overview of the following:

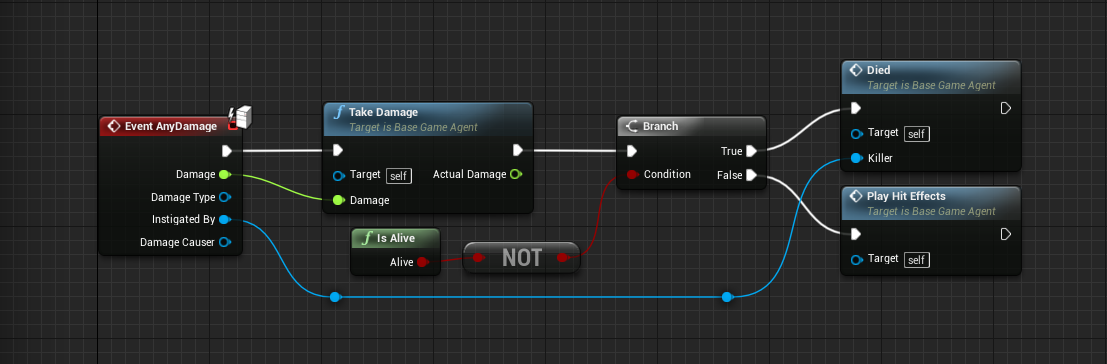
* The Event Graph
* Basic scripting
* Game loops
* Construction Scripts
* The “Call in Editor” feature
* Simple debugging
* The gameplay framework
* Balancing gameplay

**PREREQUISITES**

* An Epic Games account
* A copy of Unreal Engine 4
* A basic understanding of the engine and how to use the editor

## Introduction

### What Is Blueprint?



An example of Blueprint

The Blueprints Visual Scripting system in Unreal Engine is a complete gameplay scripting system that uses a node-based interface to create gameplay elements from within Unreal Editor.

As with many common scripting languages, it is used to define object-oriented (OO) classes or objects in the engine. As you use UE4, you’ll often find that classes defined using Blueprint are colloquially referred to as just “Blueprints.”

|  |  |
| --- | --- |
| C:\Users\Melanie Nikdel\AppData\Local\Microsoft\Windows\INetCache\Content.Word\tips-png-4.png | Need more info?  Head to [http://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html](http://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html%20). |

### How Do Blueprints Work?

In their basic form, Blueprints are visually scripted additions to your game. By connecting Nodes, Events, Functions, and Variables with Wires, it is possible to create complex gameplay elements.

Blueprints work by using graphs of Nodes for various purposes (object construction, individual functions, and general gameplay events) to implement behavior and other functionality.

## 3.1. The Event Graph

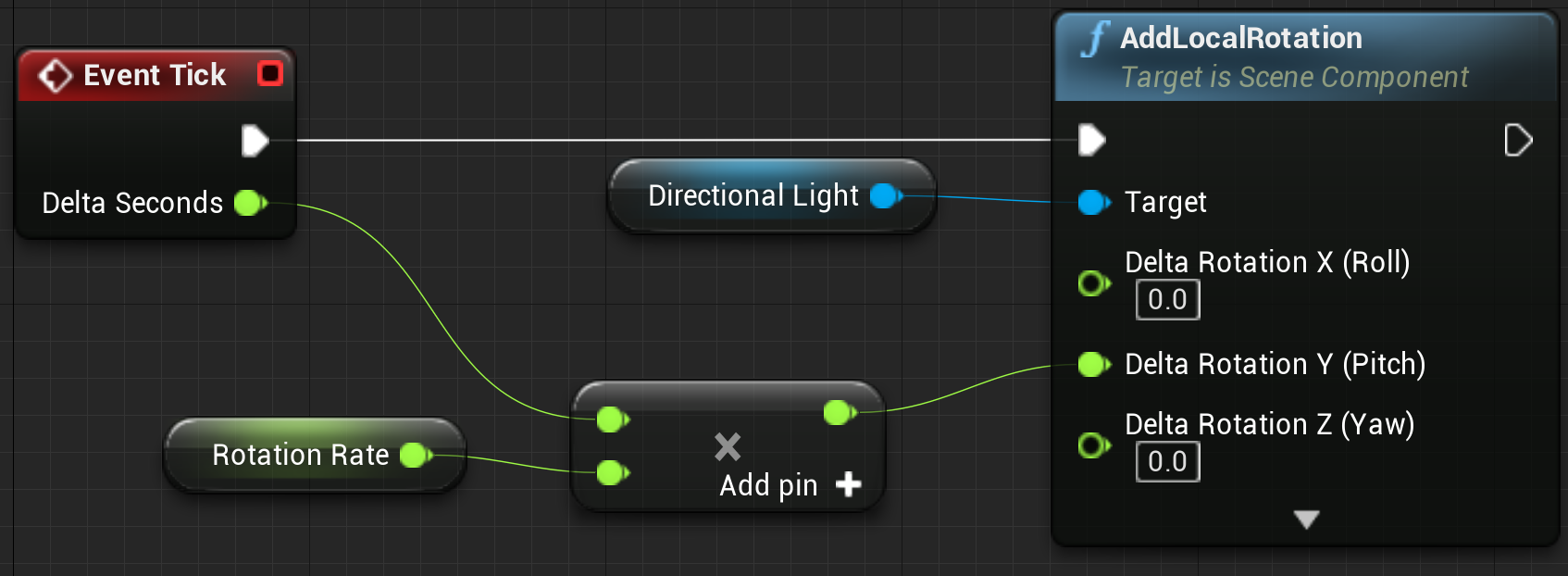
The Event Graph of a Blueprint contains a node graph that uses events and function calls to perform actions in response to gameplay events associated with the Blueprint. The graph is used to add functionality that is common to all instances of a Blueprint. It is where interactivity and dynamic responses are set up. When a developer right-clicks in the graph panel, a context-sensitive menu will appear that shows all options that can be used based on the existing selection.



## 3.2. Basic Scripting

When scripting, functions and events are used to process data and execute tasks while variables are used to store data and provide references to objects within the world.

In the example below, the Tick event calls the AddLocalRotation Function which rotates a Directional Light according to the Rotation Rate variable.



### Variables

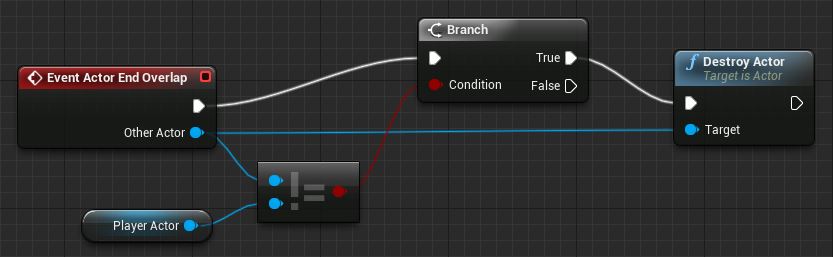
Variables are objects that hold a *value* or *reference* an Object or Actor in the world. Variables can be accessible internally to the Blueprint containing them, or they can be made accessible externally so that their values can be modified by designers working with instances of the Blueprint placed in a level.

Create a variable in Blueprint by either selecting the “+” next to “Variables” or by dragging off of a function output and selecting “Promote to Variable.



### Events and Flow

Events and Functions can be used to coordinate process flow, alter variables and process actions in the world. In the example below, the Event “Actor End Overlap” passes in a reference to an Actor that was overlapping the current Actor. In turn, this reference is compared to a variable that holds a reference to a Player. The branch node redirects the flow of events if the reference is not referencing the Player Actor. If this is true, the “Other Actor” is destroyed.



### Nodes, Pins, and Wires

Nodes, Pins and Wires are the core of coordinating and directing functionality within Blueprint.

* Pins are the inputs and outputs of nodes. Nodes can represent Functions, Events, Macros, Subgraphs or Variables.
* White wires represent the execution flow of functionality. In the above example the order of events goes from the End Overlap to the Branch to the Destroy Actor.
* Colored wires represent the passing of variable information between nodes. When something flows from one pin to another information is being communicated from the output of one node to the input of the next.

## 3.3. Game Loops

Games process information in what are commonly known as “frames”. By updating many frames in a short period time the actions of a game appear continuous. When a frame is updated, the information from the previous frame is combined with the time since the previous update to update the Actors in the world. This process of continuously iterating through objects in the world and updating them is the core subsystem in what is commonly known as the “Game Loop”.

In Unreal each Actor has what is known as a “Tick” event. By default Tick is executed every frame on each Actor with Tick enabled; allowing the Actor to continuously update variables and execute tasks.

### Delta Seconds

The time between the previous update and the current update is known as Delta Seconds. Picture a ball flying through the air in a game. The new location of the ball is equal to the previous location plus the distance travelled since the last update. The distance travelled is equal to the velocity of the ball multiplied by the time since the last update. When an update is called on the ball the delta time is read in, combined with the velocity and previous location to work out the new position of the ball.

This logic of Delta Seconds being used for a velocity change exemplifies the importance of Delta Seconds in any functionality that requires continuous change over time.

### Demonstration: Simple Day/Night Cycle That Rotates an Actor on Tick

**Reference in Example Content: Blueprint'/Game/Examples/BPTimeOfDay.BPTimeOfDay'**

## 3.4. Construction Scripts

The Construction Script is a type of graph within Blueprint Actor classes that executes when that Actor is placed or updated in the editor, but not during gameplay.

It is useful for creating easily customizable props that allow environment artists to work faster, such as a light fixture that automatically updates its material to match the color and brightness of its point light component, or a Blueprint that randomly scatters foliage meshes over an area.

### Demonstration: Simple Generation of Assets in a Line

**Reference in Example Content: Blueprint'/Game/Blueprints/BP\_Fence.BP\_Fence'**

## 3.5. Call in Editor

Events and Functions can be marked as “Call in Editor” within code or Blueprint. This automatically creates a button on Actors of that class that can be clicked to perform functionality within the editor. An example of this would be randomly selecting a color for the object or moving the object to another nearby relevant Actor.

### Demonstration: Simple Button to Randomize an Actor’s Color

**Reference in Example Content: Blueprint'/Game/Blueprints/ColorCube.ColorCube'**

## 3.6. Debugging

Through the Blueprint Editor the execution of code can be visualized and, when problems arise, debugged. Within this view, the execution wires that are currently being run will highlight and the execution path will be visible. In addition to this, ‘breakpoints’ can be added with the F9 key that will automatically pause the game and highlight a node when it is executed.

### Demonstration: Adding a Breakpoint

Add a breakpoint on any existing Blueprint by pressing F9 on a given Node.

### Demonstration: Watching the Flow

Demonstrate the visual Blueprint flow of execution while the game is running.

## 3.7. Framework Overview

### Object

The base building blocks in the Unreal Engine are called Objects and contain a lot of the essential “under the hood” functionality for your game assets. Just about everything in Unreal Engine 4 inherits (or gets some functionality) from the Object class.

### Actor

An Actor is any Object that can be placed into a level. Actors are a generic class that support 3D transformations, such as translation, rotation, and scale. Actors can be created (sPawned) and destroyed through gameplay code. There are several different types of Actors. Examples include StaticMeshActor, CameraActor, and PlayerStartActor.

### Pawn

Pawns are a subclass of Actors and serve as an in-game avatar or persona (for example, the characters in a game). Pawns can be controlled by a Player Controller or by the game’s AI, in the form of non-player characters (NPCs).

If the Pawn is the controllable ‘body’, the Controller is the ‘brain’ that feeds information back to the Pawn for direction. Controllers come in two key types: **Player Controllers** and **AI Controllers**. A Player Controller receives input from the player and feeds that information through to the currently possessed Pawn. An AI Controller processes the world and gives information to its own controlled Pawn.

This split in functionality between the Controller exists for multiple reasons, however the most important are:

* When a Pawn is destroyed, the Controller continues to exist. This allows, for instance, a player in a multiplayer game to *unpossess* their destroyed body and *possess* a new one in a single continuous process.
* Transitioning between Pawns to allow for modularized functionality. An example of this is a player Controller directing it’s desired movement direction through to its controlled Pawn body. This Pawn body knows how it should be moving through its own movement component. If the player was to try and enter a vehicle they would begin to play an animation of entering the vehicle, attach their Pawn to the vehicle, unpossess the Pawn and instead possess the vehicle. Upon possessing the vehicle Pawn the Player Controller will now pass the same movement information as before through to the vehicle which will now process that information using its own movement component to react accordingly.

### Character

Character is a subclass of the Pawn class that is intended to be used as a player character. The Character class includes a collision setup, input bindings for bipedal movement, and additional code for movement controlled by the player.

### Game Mode

The GameMode class is responsible for setting the rules of the game that is being played. The rules can include how players join the game, whether or not a game can be paused, and level transitions, as well as any game-specific behavior, such as win conditions.

### Game State

The GameState class contains the information that you want replicated to every client in a game—or, more simply, it is “the state of the game” for everyone connected.

### Game Instance

The Game Instance class can be used to store information that needs to persist for as long as the application is running. Unlike other game objects, the Game Instance is not destroyed between level loads.

## 3.8. Bird of Prey Project Overview

Bird of Prey is a demo project of an action vertical scroller based on the arcade titles of the early to mid-1990s. Let’s look at the Game Framework and structure of the key elements of the project and which parts of the game are built within which areas.

**Pawns:** Player ships, enemy ships, bosses, and turrets

**Actors:** Projectiles and pickups

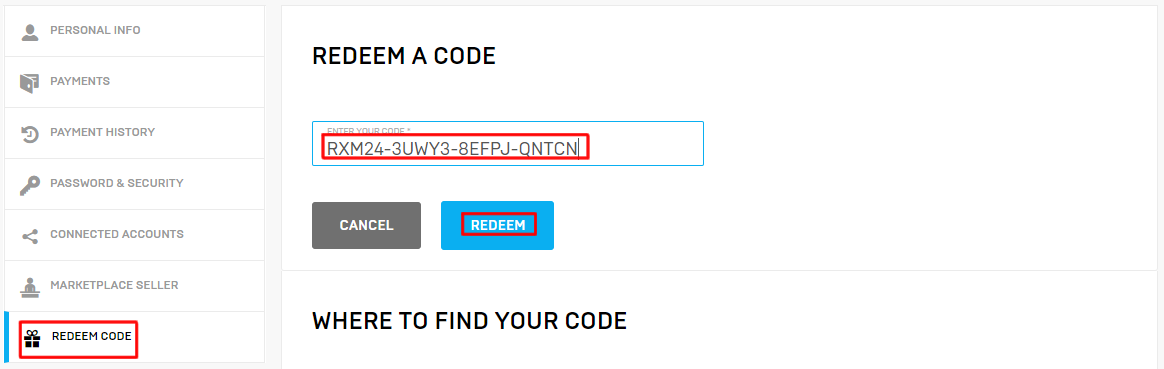
**Game Mode:** Handles primary gameplay decisions and logic. When ships are destroyed, their points are evaluated here.

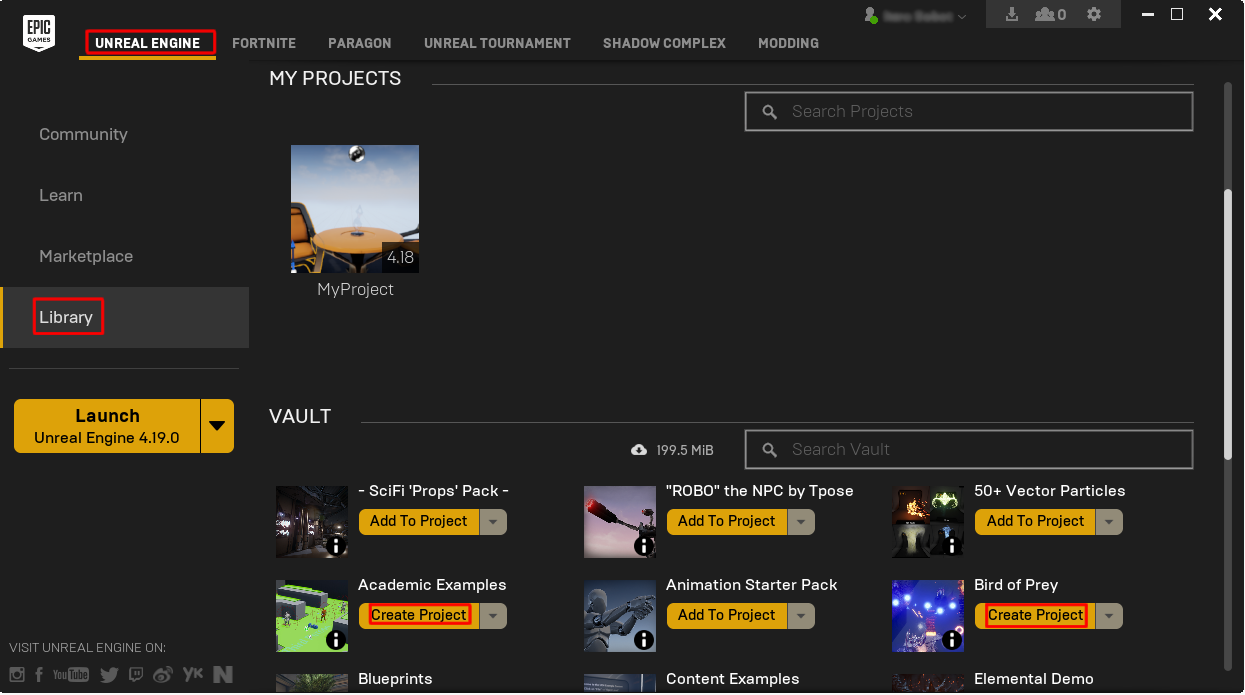
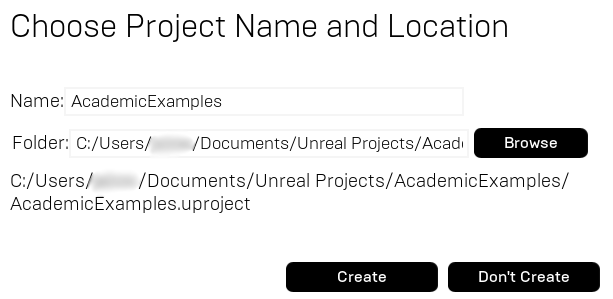
**Game State:** Stores the high scores and other information

**Game Instance:** Stores player data to store information between the Main Menu and the game.

## 3.9 Supplementary Resources

You will need these files to complete the Exercises on this page. These files are hosted on the Unreal Engine Marketplace and can be accessed from your Launcher by following the directions below.

1. On the [Code Redemption tab](https://www.unrealengine.com/dashboard/code-redemption) (<https://www.unrealengine.com/dashboard/code-redemption>) of your Epic personal dashboard, enter the code RXM24-3UWY3-8EFPJ-QNTCN, then click REDEEM and CONFIRM  


2. Open the Epic Games Launcher (you can download the [Launcher](https://www.unrealengine.com/download) [here](https://www.unrealengine.com/download)if you need it), navigate to the Vault section of the Unreal Engine Library tab, and click Create Project on Academic Examples or Bird of Prey  
  
  
3. Choose a project name and location and click Create  


And that's it! If you have any further questions about how to download these products from the Epic Games Launcher, please contact [marketplace-support@unrealengine.com](mailto:marketplace-support@unrealengine.com).

# Exercises

## Exercise 3A: Rebalancing a Game

**Deliverables: Project files (uassets), In-class observation**

Have the students modify the Bird of Prey project by tweaking the properties of the existing ships. Each student should then produce a summary (about a paragraph) of how the changes affect gameplay.

The purpose of this exercise is to validate the students’ understanding of gameplay systems and the importance of game balance.



Bird of Prey

**Directions**

1. Download the Bird of Prey sample and create a new project with the latest downloaded version of the engine. The sample can be downloaded from the Marketplace. Please see the instructions above for more information.
2. After loading the project, play through the game and get a feel for how each element interacts.
3. Once you have played the game a few times, locate the “Enemy ships” folder within the Content Browser and select the “simple” enemy.
4. Modify the properties of the ship (speed, armor, health, base weapon) and play through the game again, taking note of the influence each change has on the gameplay.

## Exercise 3B: Adding Additional Content

**Deliverables:** Project files (uassets), In-class observation

**Instructor Task:** Demonstrate the functionality of gameplay elements using the Bird of Prey example game.

**Student Task:** Create a new pickup from the Bird of Prey example and add functionality.



**Directions**

1. Download the Bird of Prey sample from the Marketplace. See above for more detailed information.
2. Add a new ship with a unique characteristic (speed, armor, etc.).
3. Create a new pickup with unique functionality. (Try causing radial damage when picked up! Don’t forget to ignore yourself when dealing radial damage or you’ll blow yourself up.)
4. Create a new weapon type with a unique behavior or function.
5. Test out your newly created items in the game.
6. Save and submit your work in a zipped file with the following naming convention: “*yourname*\_EX3B.zip”.

# Multiple-Choice Quiz

|  |  |  |  |
| --- | --- | --- | --- |
| Name: |  | Date: |  |

1. Where would you typically handle Begin Play code?
   1. Construction Script
   2. Variable
   3. Event Graph
   4. Game Array
2. Which of the following are primitive variables in Unreal?
   1. Float
   2. Name
   3. Integer
   4. All of the above
   5. None of the above
3. A branch directs the flow of execution based on the value of a binary input.
   1. True
   2. False
4. Which color denotes an event node in Blueprint?
   1. Green
   2. Purple
   3. Blue
   4. Red
   5. Orange