Class Blueprints

Contents

1 Making a Blueprint Class 1

2 The Viewport 1

3 The Event Graph (The “Brain”) 2

4 Blueprint Setup: Let’s Create an Interactive Door 3

4.1 Components and Viewport . . . 3

5 Variables and Custom Events 3

5.1 Event Logic . . . 3

6 The Animation: Timeline Node 3

6.1 Timeline Configuration . . . 4

6.2 Applying Rotation . . . 4

7 Quick Summary and Important Terms: 4

7.1 Execution Flow Summary . . . 4

7.1.1 Execution Logic . . . 5

7.1.2 Execution Pins . . . 5

7.2 Animation Summary . . . 5

7.2.1 Timeline . . . 5

7.2.2 Float Track . . . 5

7.2.3 Relative Rotation . . . 5

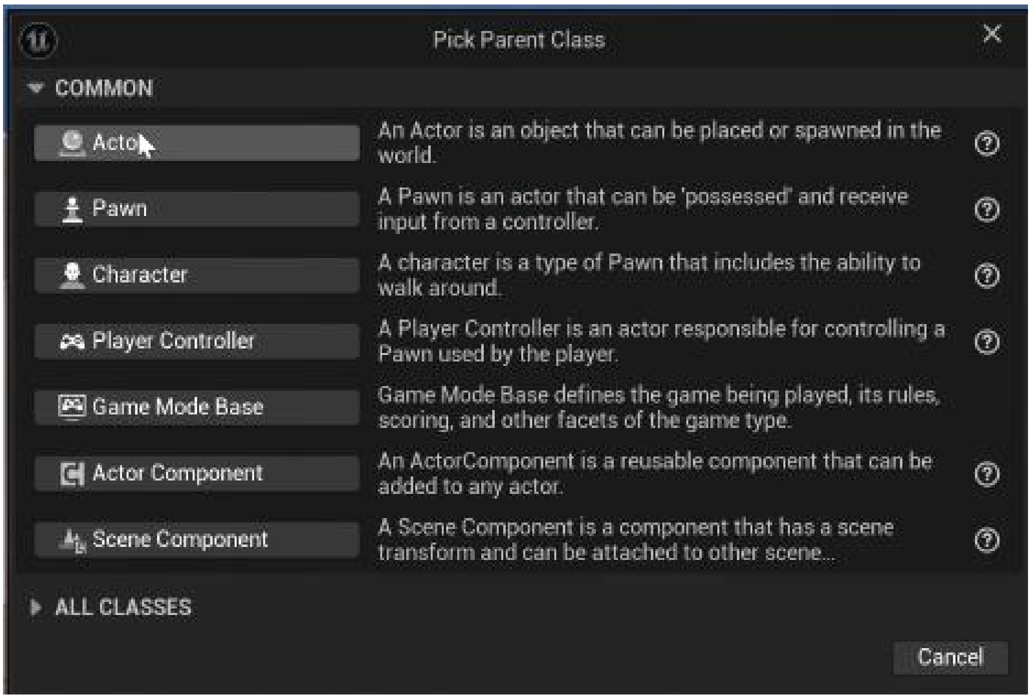
7.2.4 Split Struct Pin . . . 5

8 Optimization Tip 5

A **Class Blueprint** is a reusable template in Unreal Engine that defines both the *structure* (components like meshes, cameras, collisions) and the *behavior* (logic written in the Event Graph) of an object. It acts as a blueprint for creating multiple **instances** of that object in the game world, where each instance shares the same logic but can have different properties.

1 Making a Blueprint Class

1. Just **Right-click** in the content drawer and select for **Blueprint Class**.
2. Select the parent class of the blueprint from options shown in the table below.



Pick Parent Class

COMMON

Actor — An Actor is an object that can be placed or spawned in the world.

Pawn — A Pawn is an actor that can be 'possessed' and receive input from a controller.

Character — A character is a type of Pawn that includes the ability to walk around.

Player Controller — A Player Controller is an actor responsible for controlling a Pawn used by the player.

Game Mode Base — Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

Actor Component — An ActorComponent is a reusable component that can be added to any actor.

Scene Component — A Scene Component is a component that has a scene transform and can be attached to other scene...

ALL CLASSES

Figure 1: Blueprint Classes

2 The Viewport

The **Viewport** is the 3D assembly window where you define the physical structure and spatial hierarchy of the Blueprint Class. Think of this as the “construction site.”

1. **Purpose:** It allows you to transform components *Move, Rotate, Scale* relative to the **Default Scene Root**.
2. **Hierarchy:** In the Viewport, you establish parent-child relationships.
   For example, if you attach a *Doorknob* mesh to a *Door* mesh, moving the door will automatically move the doorknob.
3. **Spatial Logic:** This is where you place your *Sphere Collision*. Even though the collision is invisible in the game, the Viewport allows you to define exactly where the “interaction zone” exists in 3D space.

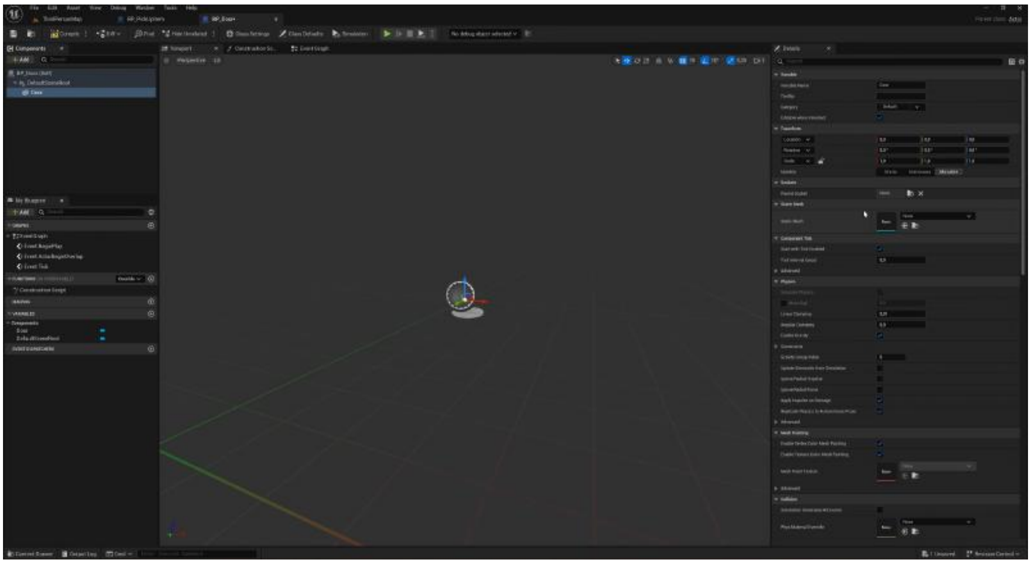


Figure 2: Viewport

3 The Event Graph (The "Brain")

The **Event Graph** is a visual scripting canvas used to define the behavior of the Blueprint during gameplay. It uses a node-and-wire system to create logic.

1. **Purpose:** It responds to **Events** (like a player pressing a button or walking into a trigger) and executes a sequence of instructions.
2. **Non-Linear Logic:** Unlike traditional code that reads strictly top-to-bottom, the Event Graph is modular. Different *Events* can sit on the same graph and wait to be triggered independently.
3. **Visual Debugging:** While the game is running, you can see "pulses" of light moving along the wires in the Event Graph, showing you exactly how data and execution flow in real-time.

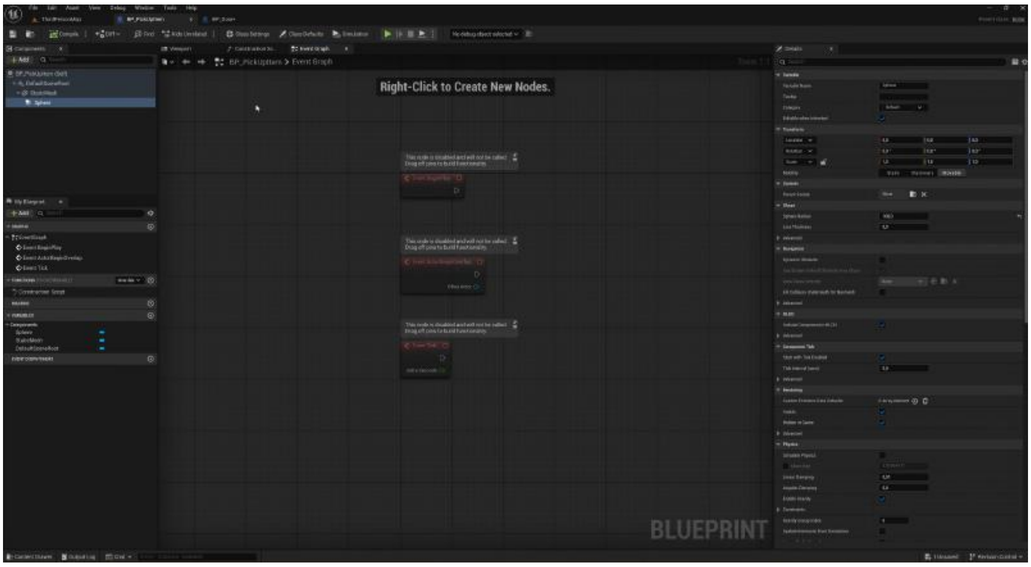


Figure 3: Event Graphs

4 Blueprint Setup: Let's Create an Interactive Door

Unlike Level Blueprints, a **Blueprint Class** (BP_Door) allows us to create a reusable interactive object.

4.1 Components and Viewport

1. **Static Mesh:** Add a door mesh. Ensure its pivot point is at the hinge, not the center, for correct rotation.
2. **Sphere Collision:** Add a Sphere Collision component. Scale it to define the "Interaction Radius" where the player can trigger the door.

5 Variables and Custom Events

In the **My Blueprint** panel, create a Variable:

- **Name:** `IsOpen`
- **Type:** Boolean

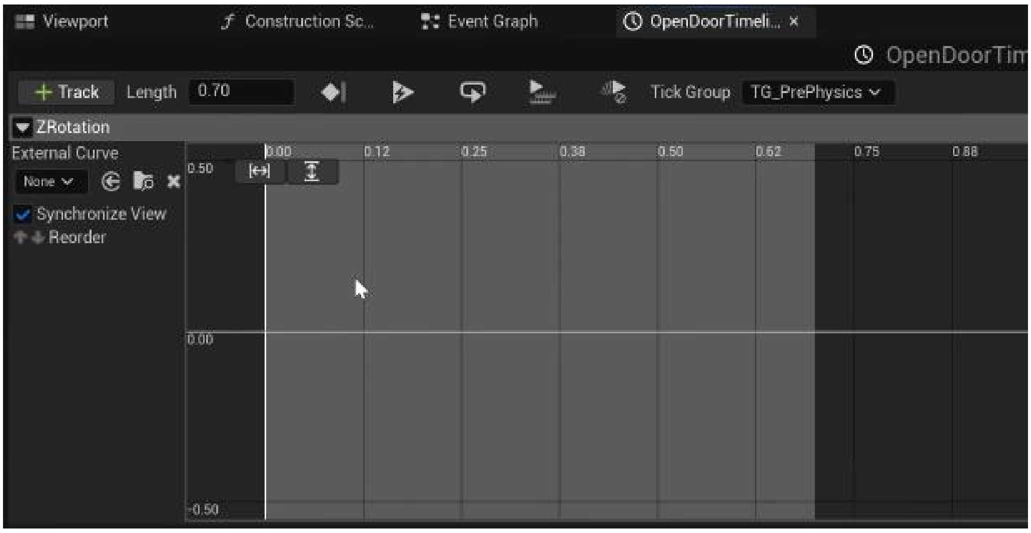
5.1 Event Logic

We define two core events in the Event Graph:

- **OpenDoor:** Sets `IsOpen` to **True** and triggers the animation.
- **CloseDoor:** Sets `IsOpen` to **False** and reverses the animation.

6 The Animation: Timeline Node

The **Timeline** node creates a smooth transition between 0° and 90°.



Length 0.70

Tick Group TG_PrePhysics

External Curve None

☑ Synchronize View

Figure 4: Timeline

6.1 Timeline Configuration

Double-click the Timeline node to open the editor (right click to add keys):

- **Length:** 1.0 second.
- **Track:** Add a Float Track named `Z-Rotation`.
- **Key 1:** Time: 0, Value: 0.
- **Key 2:** Time: 1.0, Value: 90.0 (Degrees of rotation).

6.2 Applying Rotation

To apply the animation to the mesh:

1. Drag the **Static Mesh** into the graph.
2. Drag from the mesh and call `SetRelativeRotation`.
3. **Split Struct Pin:** Right-click the *New Rotation* pin and select *Split* to reveal individual X, Y, and Z inputs.
4. **Connect:** Connect the `Z-Rotation` track output to the *New Rotation Z* input.
5. **Update:** Connect the Timeline's *Update* execution pin to the `SetRelativeRotation` node.

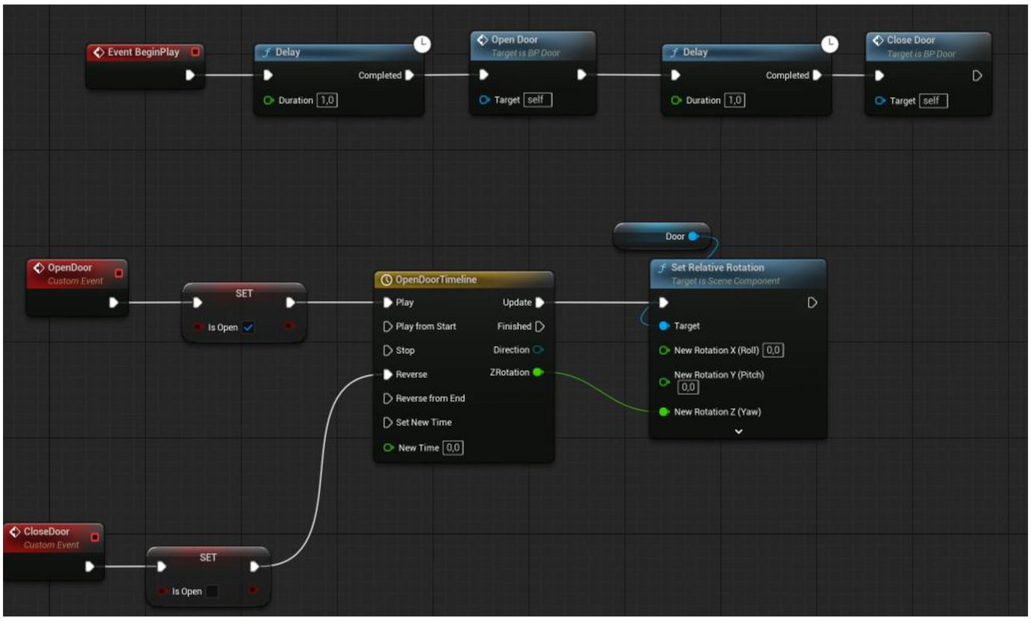


Figure 5: Blueprint Event Graph for Door Movement

7 Quick Summary and Important Terms:

7.1 Execution Flow Summary

The logic operates as a state machine:

$$\text{Action} = \begin{cases} \text{Play Timeline} & \text{if } \texttt{IsOpen} = \text{False} \to \text{Set } \texttt{IsOpen} = \text{True} \\ \text{Reverse Timeline} & \text{if } \texttt{IsOpen} = \text{True} \to \text{Set } \texttt{IsOpen} = \text{False} \end{cases} \tag{1}$$

7.1.1 Execution Logic

1. **Event Graph** of the Blueprint where logic is written using nodes.
2. **Custom Events** are user-defined node that triggers a specific sequence of actions. For example, *OpenDoor* and *CloseDoor* are custom events created to organize logic.
3. **Branch** is the most common flow-control node. It checks a Boolean variable; if the result is **True**, it follows the "True" execution path; otherwise, it follows "False."

7.1.2 Execution Pins

The white arrows that dictate the sequence of code. If an execution pin is not connected, that logic will never run.

7.2 Animation Summary

7.2.1 Timeline

A specialized node that generates a float (number) over a set period. It is used to create smooth, frame-rate independent animations (e.g., a door swinging open over 1 second).

7.2.2 Float Track

The internal curve of a Timeline that maps Time (x-axis) to a Value (y-axis).

For a door animation:

$$\text{Time: } 0 \Rightarrow \text{ Value: } 0$$

$$\text{Time: } 1 \Rightarrow \text{ Value: } 90$$

7.2.3 Relative Rotation

Rotating a component relative to its parent. This is preferred over "World Rotation" so that if the entire house rotates, the door still swings correctly relative to the frame.

7.2.4 Split Struct Pin

A command used to break a complex variable (such as a 3D Rotation vector) into its individual components:

$$(X, Y, Z)$$

This allows manipulation of a single axis at a time.

8 Optimization Tip

To ensure the door doesn't "snap" when interrupted, connect the `CloseDoor` event to the **Reverse** input of the Timeline rather than *Reverse from End.* This ensures the door starts closing from its current position.