Unreal Engine

Friday, May 23, 2025 2:38 PM

SHORTCUTS

- G - Toggle gameview on and off
- Content Drawer - Control + Spacebar
- Dispay Cursor - when in Play mode - SHIFT + F1
- ALT + left click - for rotation
- ALT + right click - for moving closer and further from an object
- In Material editor -1/2/3/4 + left click - create 1/2/3/4 value constant
- CTRL + SHIFT + S - Save all
- CTRL + B + Select any object - shows the object in the Content Drawer
- CTRL + Clicking on an object - selecting multiple objects
- (While selecting an object) + H - To hide an object
- (While selecting an object) + CTRL + H - To unhide an object
- (While selecting an object) + F - Snap the viewpoint to the location of an object

GENERAL

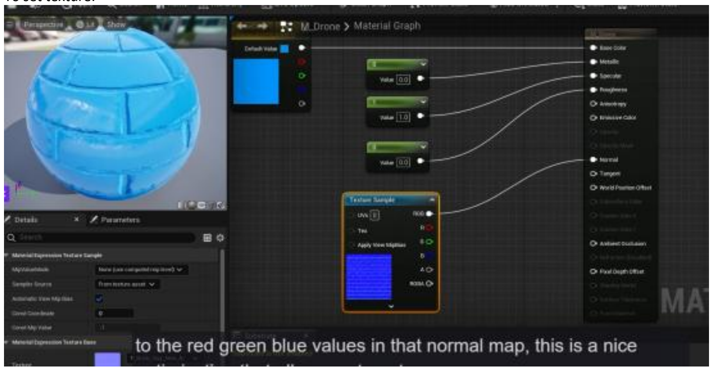
Game engine handles:

- Rendering - generating an image
- Input - input devices
- Audio system - Handling Sounds and Music
- Physics - Physics System
- Assets - Importing and Editing
- UI system - Creating UI
- Scripting - Gameplay Logic and Libraries
    - In Unreal Engine we use Blueprints for Game Logic
- Networking - Multiplayer
- Effects System - Particle Systems

- Hovering over a function in UE will show a tooltip
- We can access any map through the Content Drawer section
- Typically mesh files are either .uasset or .fbx types
- To import an asset, it is possible to just pull the file into content drawer folder
- IF we want to move an object to another project, we have to use the migrate option instead of just copying and pasting a file .
    - ALWAYS migrate to the content folder of another project

- Skeletal mesh allows the moving of some parts in the object relative to other parts

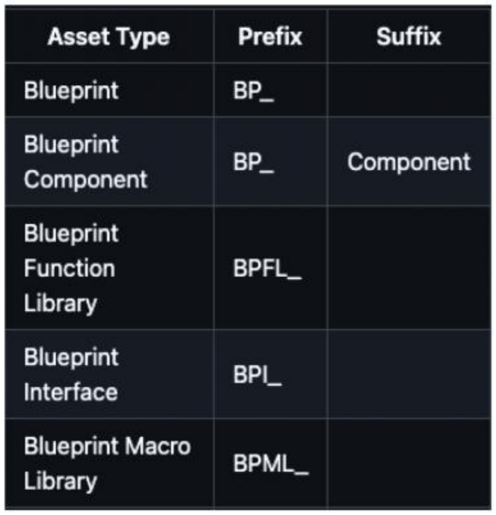
- To set texture:



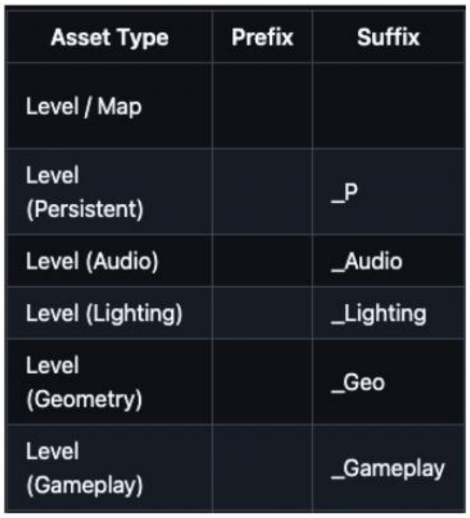
to the red green blue values in that normal map, this is a nice

Best practices

- Stick to one naming convention
- Group and logically structure the project
- Prefixes/suffixes based on the project type:



Asset Type	Prefix	Suffix
Blueprint	BP_	
Blueprint Component	BP_	Component
Blueprint Function Library	BPFL_	
Blueprint Interface	BPI_	
Blueprint Macro Library	BPML_	



Asset Type	Prefix	Suffix
Level / Map		
Level (Persistent)		_P
Level (Audio)		_Audio
Level (Lighting)		_Lighting
Level (Geometry)		_Geo
Level (Gameplay)		_Gameplay

- Use comments and comment blocks
- Split logic into small, reusable functions
- Keep logic flow left-to-right, top-to-bottom
- Avoid "Blueprint spaghetti" (complex, tangled graphs)

- Avoid using Tick unnecessarily –use Event Dispatchers or timers
- Avoid GetAllActorsOfClassin large scenes –save instances
- Don't use simple Delay nodes –prefer timers
- Limit casting –use interfaces instead
- Don't import whole asset packages

Version control
- Install Git LFS for large binary files
- Set up proper .gitignoreand .gitattributes
- Close UE before commiting
- Consider saving plugins
- Merge is not possible

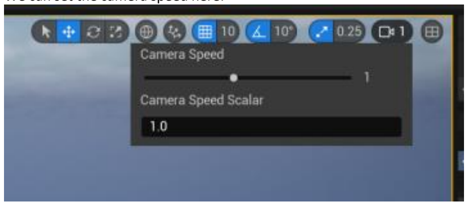
Collaboration
- Install Git LFS for large binary files
- Set up proper .gitignoreand .gitattributes
- Close UE before commiting
- Consider saving plugins
- Merge is not possible

Performance optimization
- Don't use gravity and collision if not needed
- Use level of detail (LOD's)
- LOD's and calculation of collisions
- Use adequate texture quality

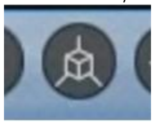
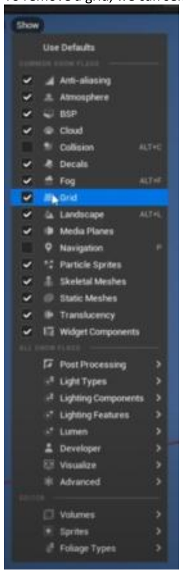
Modern UE technologies
- Lumen and Nanite

NAVIGATION

- Shift + clicking multiple objects - allows you to select (and manipulate) with more objects at the same time
- To duplicate an object, we can either copy and paste it using CTRL + C and CTRL + V OR we can move it while holding ALT to al so duplicate it
- We can set the camera speed here:



Camera Speed: 1
Camera Speed Scalar: 1.0

- This button allows you to navigate an object either through its local or global space
- 
- To remove a grid, we can select the following option:
- 

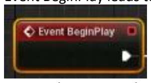
BLUEPRINT
- Blueprint behavior can also be implemented in C++ in UE
- Comparison between blueprints and c++:

C++	**Blueprints**
- Most optimal performance - More access to the code base - Harder to learn	- Easier to learn - Faster to implement behaviors

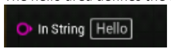
- Construction scripts are mostly used only by artists

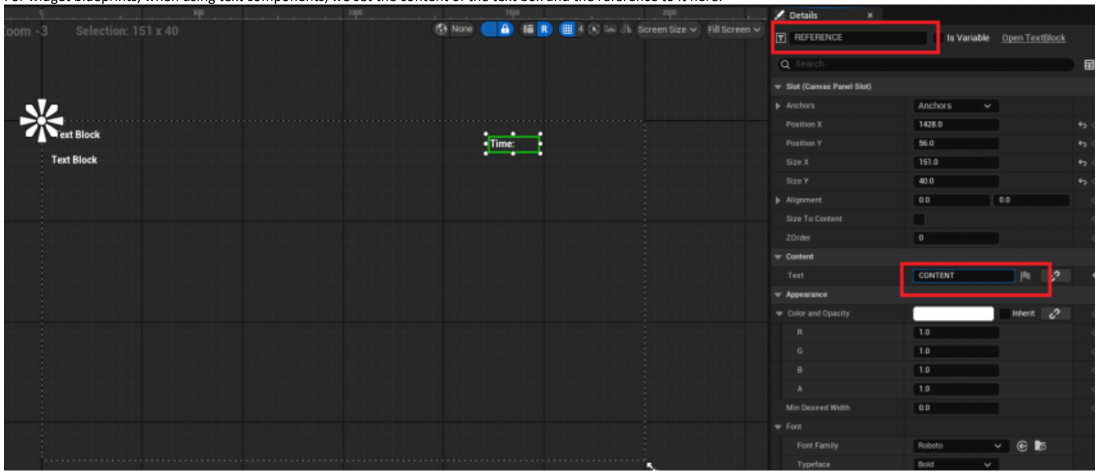
- To observe the logic flow of a blueprint in realtime while playing a game, we can select a debug object and watch as nodes ar e being triggered in the blueprint in real time.

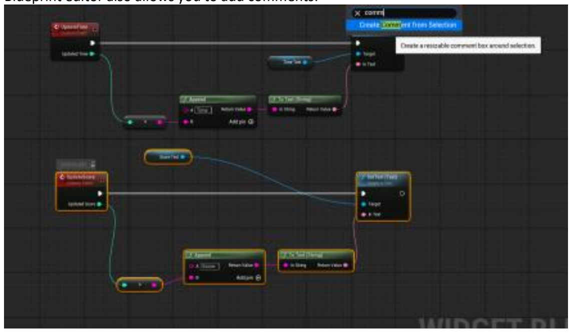
- UPON CONSTRUCTING A BLUEPRINT IT HAS TO BE PULLED ONTO THE LEVEL WE WANT TO APPLY THE BLUEPRINT TO!!!!!!

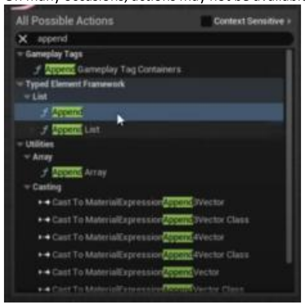
- Event BeginPlay leads to any nodes connected to it being executed:


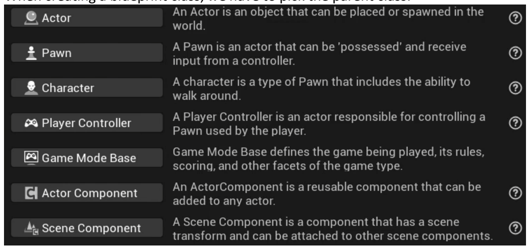
- EventTick is triggered every time a new frame is generated
- We can also use button events which are triggered upon getting a specified user input

- Every node has its input data type
- PrintString Node is very useful for verifying that some of our code is actually working
- The hello area defines the default value
- 

- For widget blueprints, when using text components, we set the content of the text box and the reference to it here:


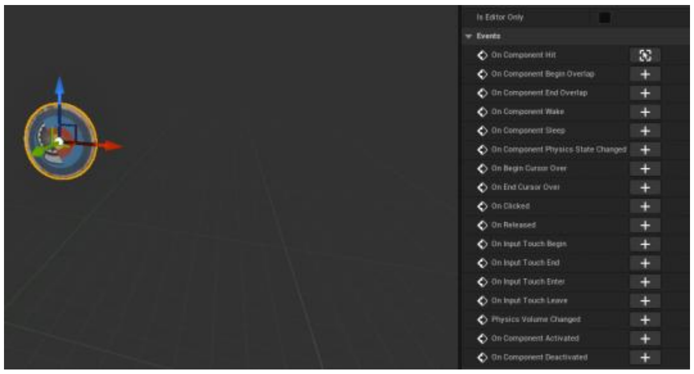
- Blueprint editor also allows you to add comments:

comm
Create Comment from Selection
Create a resizable comment box around selection

- On many occasions, actions may not be available through the search bar in blueprint editor, s we have to untick the Context S ensitive toggle button
- 
All Possible Actions
Context Sensitive
append
Gameplay Tags
Append Gameplay Tag Containers
Typed Element Framework
List
Append
Append List
Utilities
Array
Append Array
Casting
Cast To MaterialExpressionAppend3Vector
Cast To MaterialExpressionAppend3Vector Class
Cast To MaterialExpressionAppend4Vector
Cast To MaterialExpressionAppend4Vector Class
Cast To MaterialExpressionAppendVector

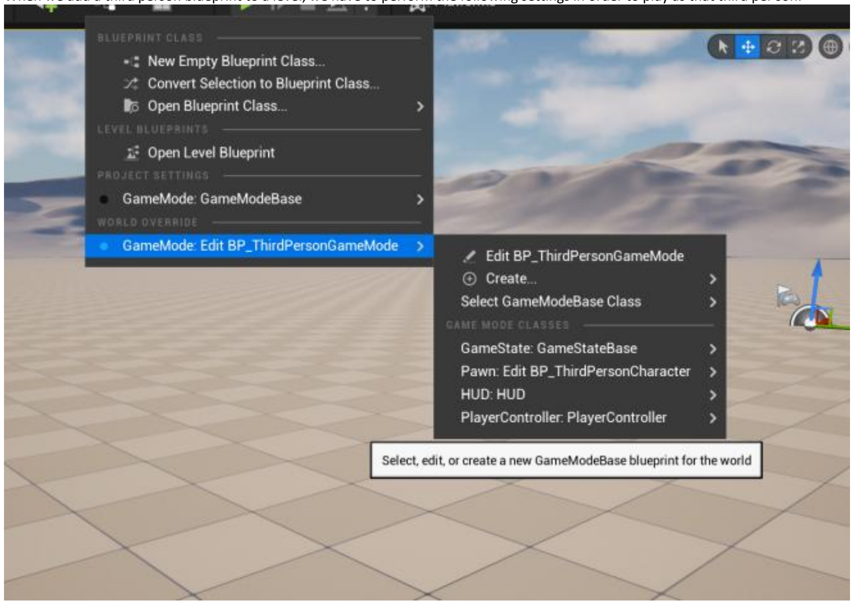
- To break the wire, hold ALT and click on the pin in a node in the blue print editor
- When creating a blueprint class, we have to pick the parent class:


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 components.

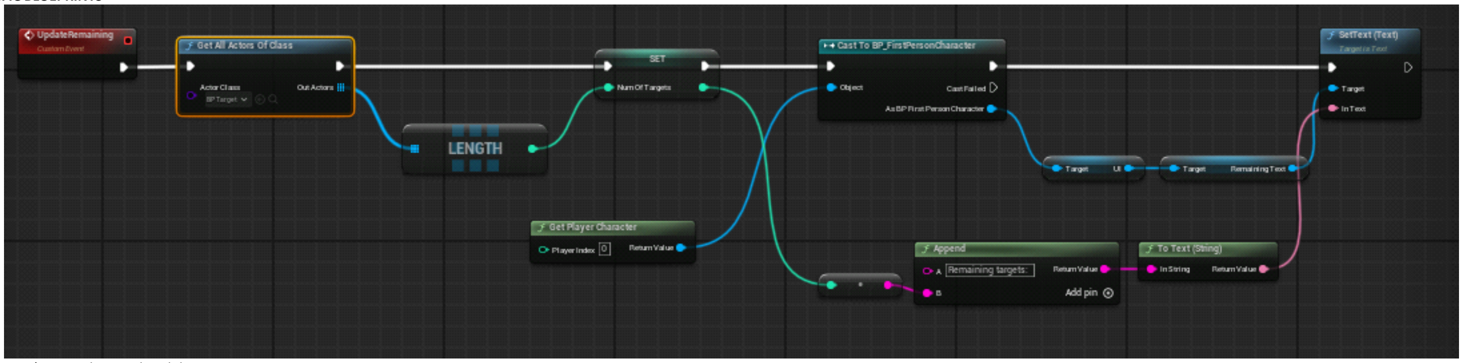
- When we tie a mesh to a blue print, we can select the message and create the logic through the events available in relation t o that mesh:



- When we add a third person blueprint to a level, we have to perform the following settings in order to play as that third per son:


BLUEPRINT CLASS
New Empty Blueprint Class...
Convert Selection to Blueprint Class...
Open Blueprint Class...
LEVEL BLUEPRINTS
Open Level Blueprint
PROJECT SETTINGS
GameMode: GameModeBase
WORLD OVERRIDE
GameMode: Edit BP_ThirdPersonGameMode
Edit BP_ThirdPersonGameMode
Create...
Select GameModeBase Class
GAME MODE CLASSES
GameState: GameStateBase
Pawn: Edit BP_ThirdPersonCharacter
HUD: HUD
PlayerController: PlayerController
Select, edit, or create a new GameModeBase blueprint for the world

SPECIFIC BLUEPRINTS

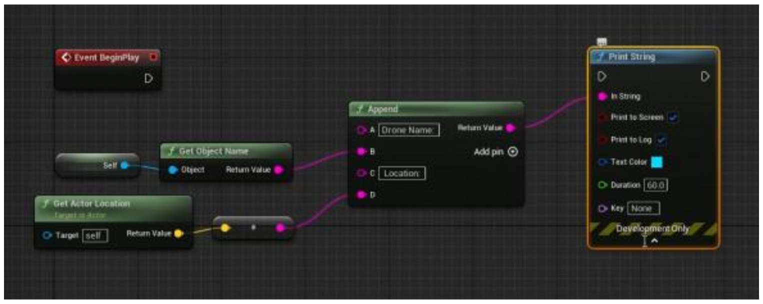


Here's a step-by-step breakdown:

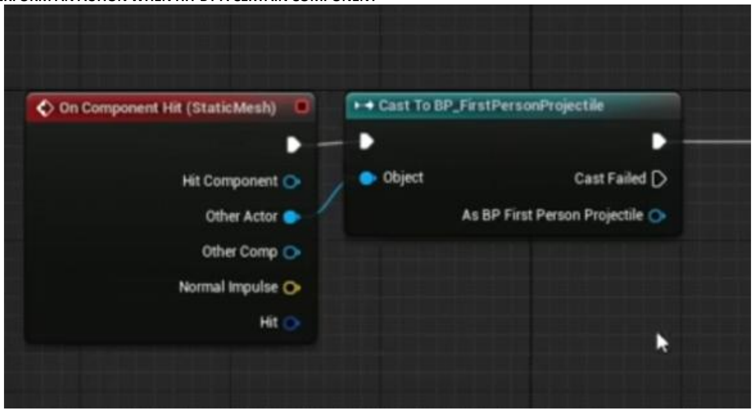
i. **UpdateRemaining (Custom Event):** This is the starting point. This event is likely triggered whenever the game needs to check and display how many targets are left – perhaps when a target is destroyed or when the level starts.
ii. **Get All Actors Of Class:** This node searches the entire game world for all active actors (objects) that belong to the BP_Target class. It gathers them into a list (an array). BP_Target is likely a custom Blueprint class representing the targets in your game.
iii. **Length:** This node takes the list of BP_Target actors and simply counts how many items are in it. This gives you the total number of t argets currently in the level.
iv. **SET MainOfTargets:** The count (the length of the array) is then stored in a variable named MainOfTargets. This variable now holds the number we w ant to display.
v. **Get Player Character:** This node gets a reference to the character currently being controlled by the player (Player 0).
vi. **Cast To BP_FirstPersonCharacter:** It then tries to confirm that the Player Character is specifically of the type BP_FirstPersonCharacter. This is often done to access custom variables or functions within that specific character Blueprint.
vii. **Accessing UI Elements:** If the cast is successful, it accesses a variable named UI (presumably a reference to a UI widget) and then another variable named RemainingText (likely the specific text block within that UI widget) from the BP_FirstPersonCharacter.
viii. **Append:** This node constructs the final text string. It takes the literal text "Remaining targets: " and adds the value of the MainOfT argets variable (converted to a string) to the end of it.
ix. **SetText (Text):** Finally, this node takes the combined string from the Append node and updates the RemainingText UI element, making it display the current count (e.g., "Remaining targets: 5").

In short, whenever the UpdateRemaining event runs, this script counts all BP_Target actors, formats a string showing that cou nt, and updates a specific text field in the player's UI.

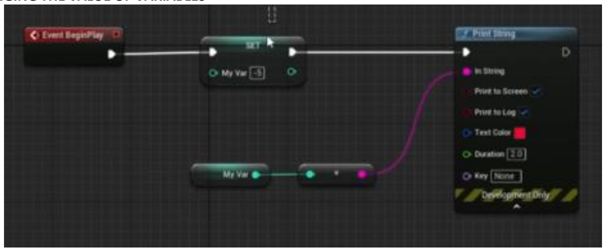
PRINT PLAYER LOCATION



TO PERFORM AN ACTION WHEN HIT BY A CERTAIN COMPONENT

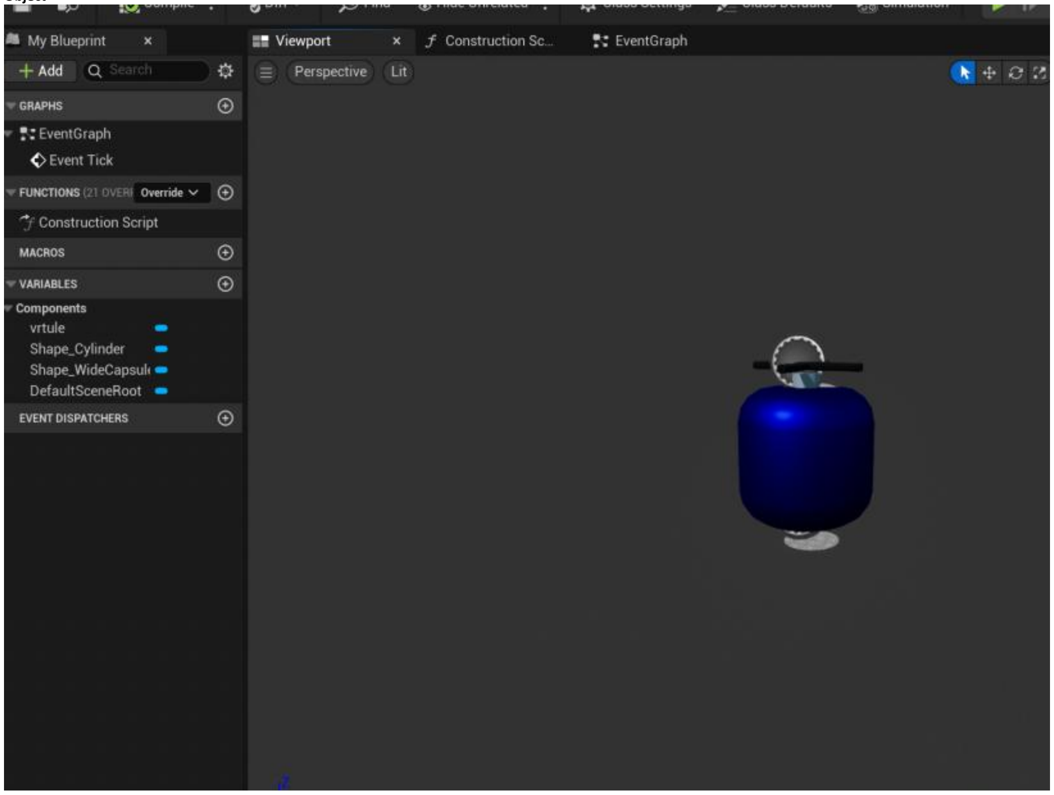

On Component Hit (StaticMesh)
Hit Component
Other Actor
Other Comp
Normal Impulse
Hit
Cast To BP_FirstPersonProjectile
Object
Cast Failed
As BP First Person Projectile

CHANGING THE VALUE OF VARIABLES

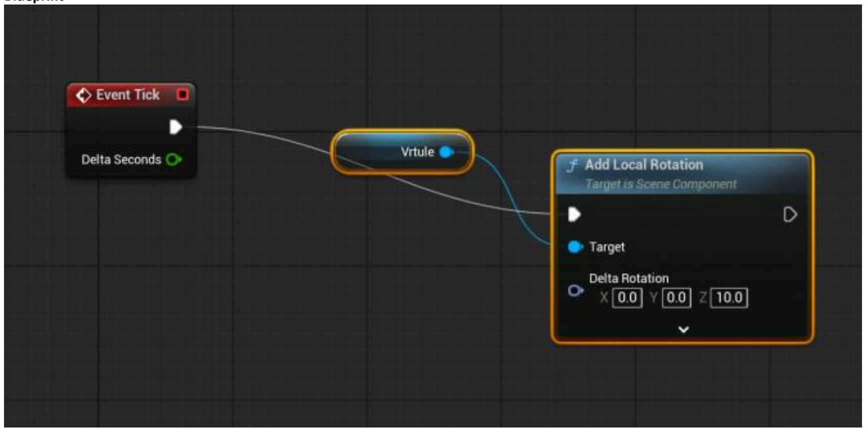


ROTATION OF AN OBJECT

Object



Blueprint


Event Tick
Delta Seconds
Vrtule
Add Local Rotation
Target is Scene Component
Target
Delta Rotation
X 0.0 Y 0.0 Z 10.0