Intro to Unreal

Created by Els Fouché

Originally presented as part of a series of workshops for
Project CGC, CSU Chico's student-run game studio.

Fundamentals of Unreal

- The Unreal Interface
- Level Creation
- Blueprint Basics
- The Gameplay Framework
- The Input System
- The User Interface

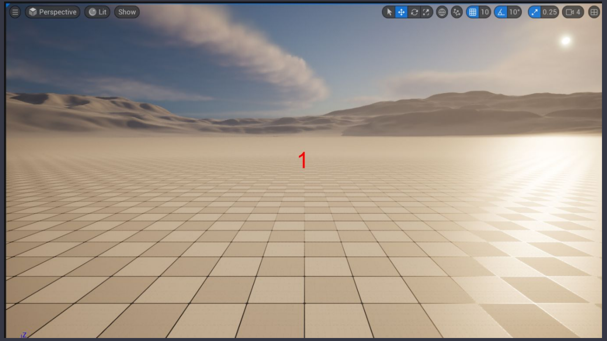
The Unreal Interface

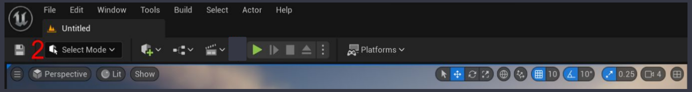
The Unreal Interface

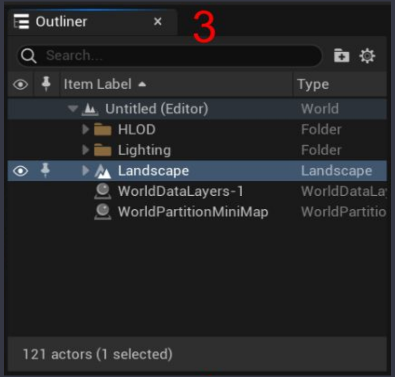
1. <u>Viewport:</u> By default, Unreal Engine 5 makes the view of your level the major focus, the same as Unity. Viewport Controls:
   - Look: right-click.
   - Move: left-click.
   - Pan: middle-click.
   - Orbit: alt+left-click.
   - Group Select: alt+ctrl+left-click.

2. <u>Modes:</u> This panel lets you select between different editor modes. Selection Mode is default. Selection mode is the primary way to manipulate objects in the level. Other modes include Landscape, for creating environments, Foliage, Modeling, and Animation modes among others.

3. <u>World Outliner:</u> Displays all the objects in the current level. You can organize the list by putting related items into folders, and you can search and filter by type.







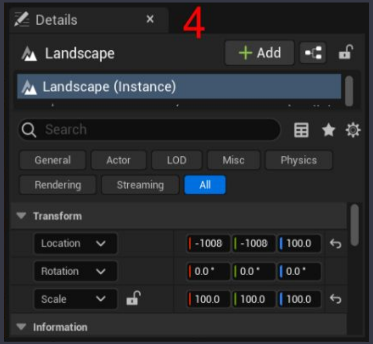
Item Label	Type
Untitled (Editor)	World
HLOD	Folder
Lighting	Folder
Landscape	Landscape
WorldDataLayers-1	WorldDataLa
WorldPartitionMiniMap	WorldPartitio

121 actors (1 selected)

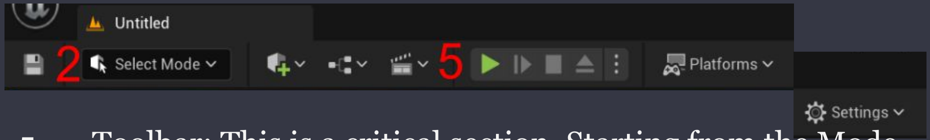
Source: https://www.kodeco.com/31800833-unreal-engine-5-tutorial-for-beginners-getting-started#toc-anchor-004

The Unreal Interface

4. <u>Details:</u> Any object you select will have its properties displayed here. As with Unity, the panel allows you to edit the settings of the object.

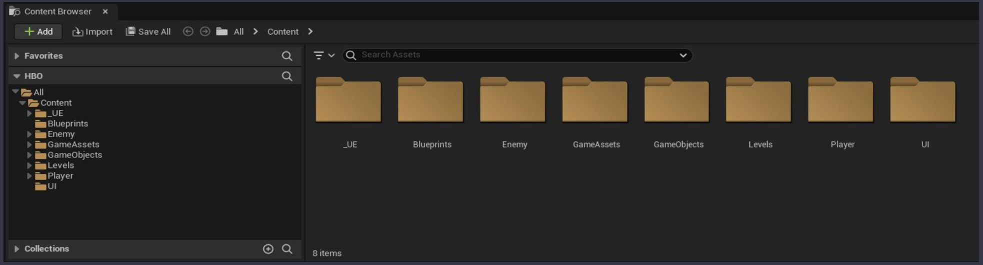


Location	-1008	-1008	100.0
Rotation	0.0°	0.0°	0.0°
Scale	100.0	100.0	100.0

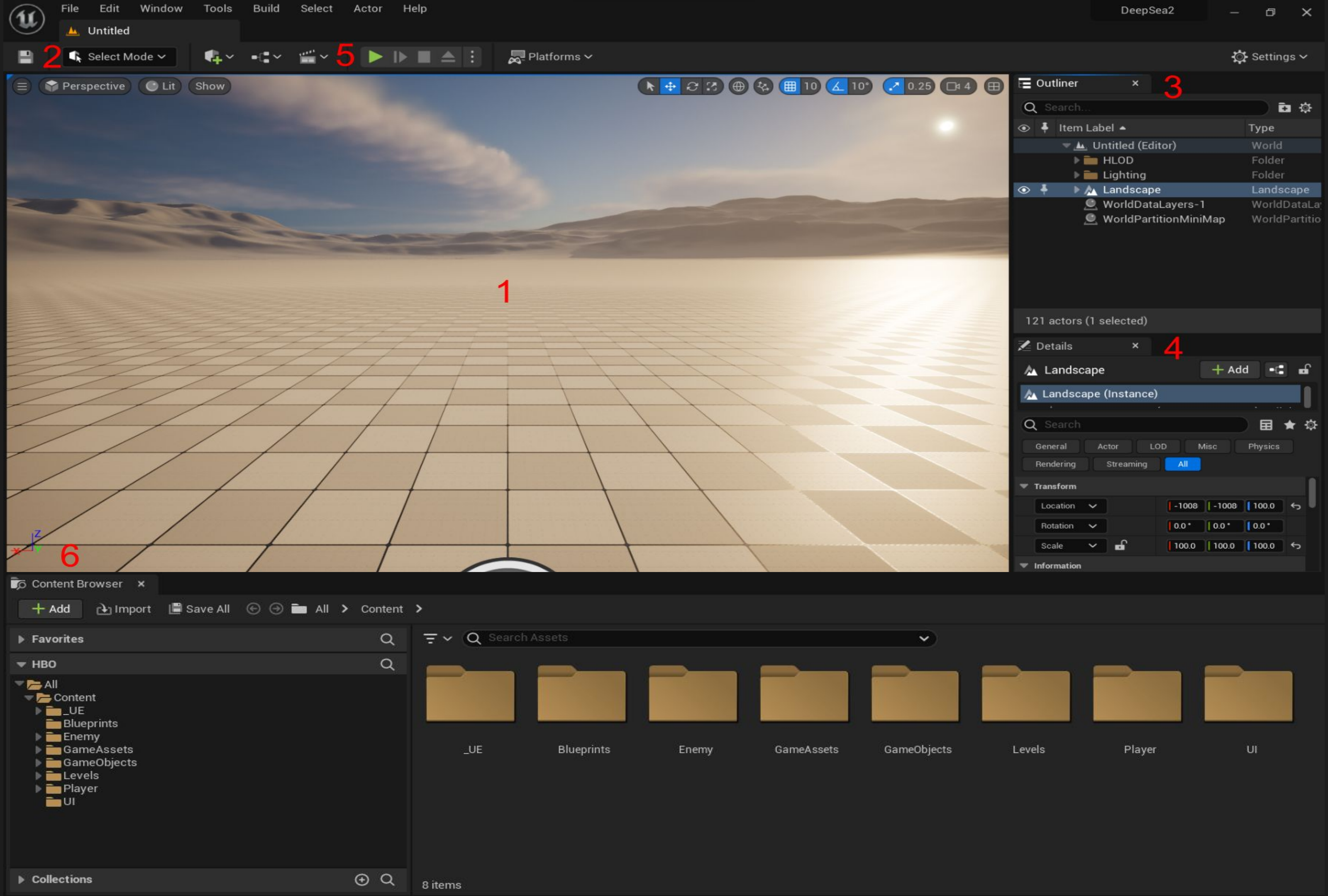


5. <u>Toolbar:</u> This is a critical section. Starting from the Mode Selection dropdown, the toolbar has these buttons:
   - Add to Project. Includes basic actors, lights, etc.
   - Blueprint selector, most useful for accessing the Level blueprint.
   - Level Sequence dropdown.
   - Play-in-editor button (changes to pause).
   - Single Frame Advance.
   - Stop (to exit play-in-editor mode).
   - Eject, which allows you to release control of a pawn.
   - Platforms dropdown, where you actually build the project.

6. <u>Content Browser:</u> This panel displays all your project files. Use this to create folders and organize files. You can search for files by using the search bar or filters.



Source: https://www.kodeco.com/31800833-unreal-engine-5-tutorial-for-beginners-getting-started#toc-anchor-004



Item Label	Type
Untitled (Editor)	World
HLOD	Folder
Lighting	Folder
Landscape	Landscape
WorldDataLayers-1	WorldDataLa
WorldPartitionMiniMap	WorldPartitio

121 actors (1 selected)

Landscape

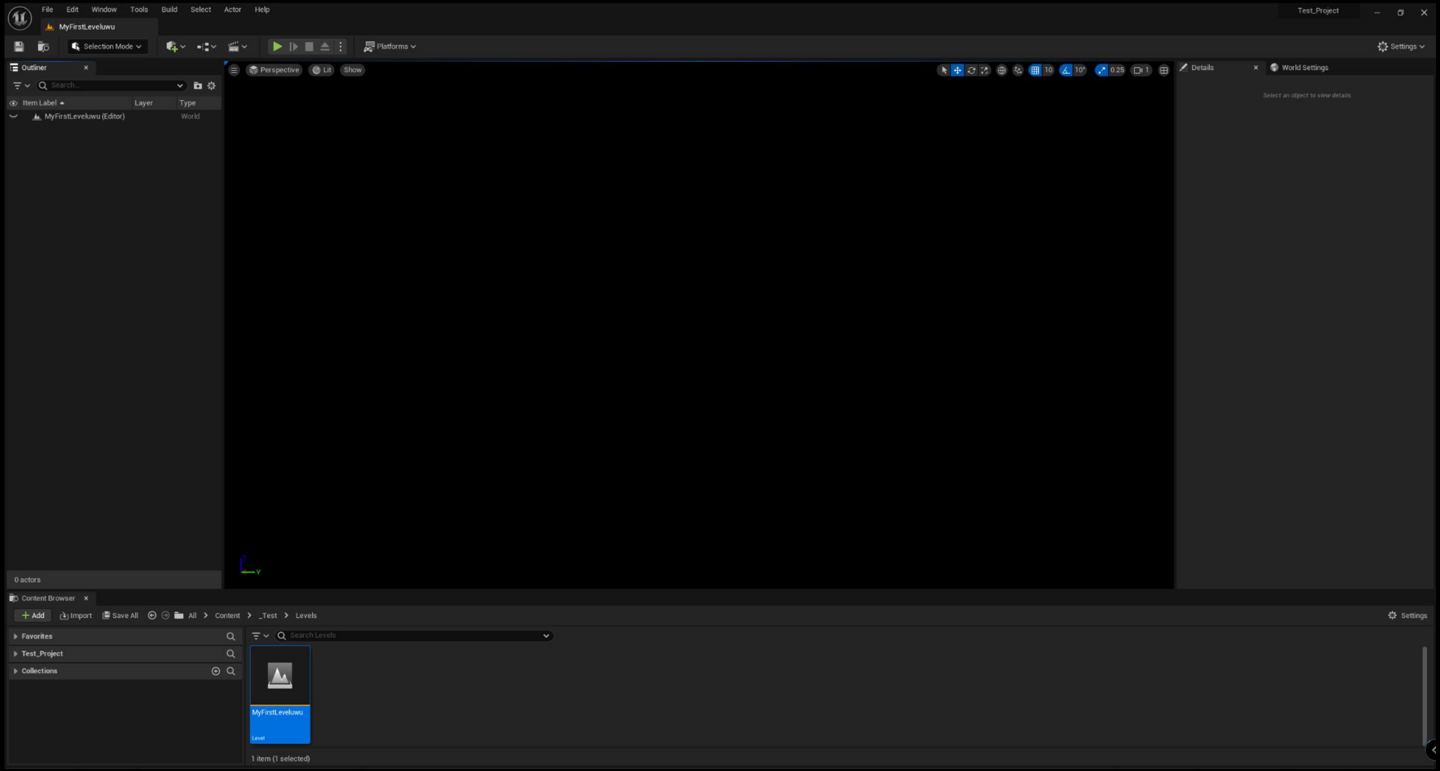
Landscape (Instance)

Location	-1008	-1008	100.0
Rotation	0.0 °	0.0 °	0.0 °
Scale	100.0	100.0	100.0

All
- Content
  - _UE
  - Blueprints
  - Enemy
  - GameAssets
  - GameObjects
  - Levels
  - Player
  - UI

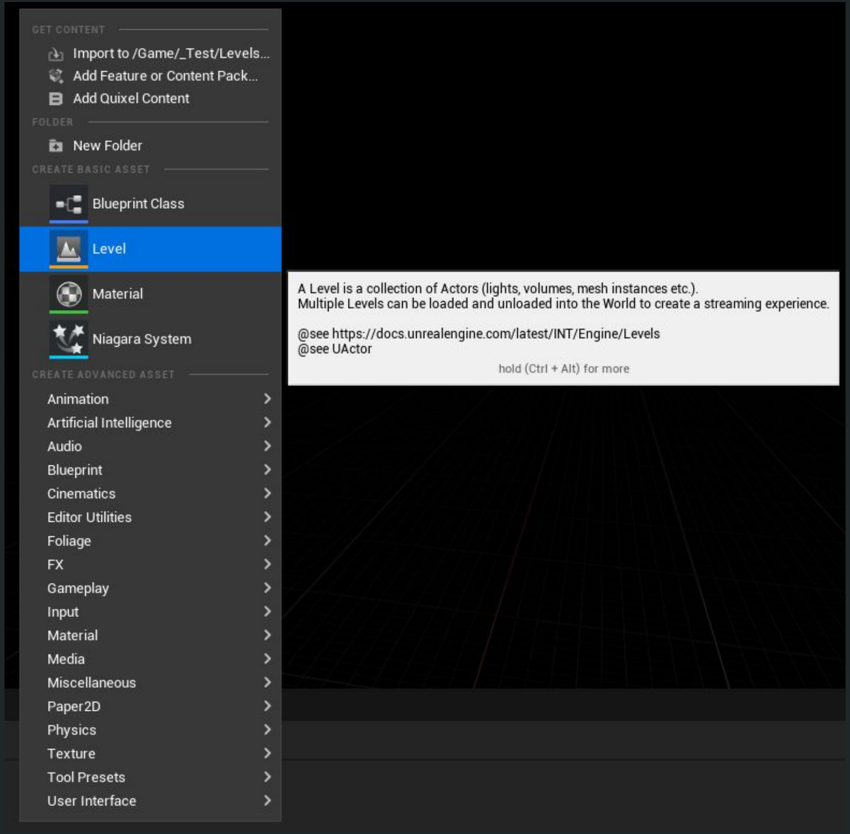
Level Creation

Help I created a new level and it's really dark and I'm scared



Level Creation

- Right-clicking inside the content browser allows you to create new assets.
- Here, we've highlighted the 'Level' selection which automatically displays a tooltip.
- Selecting 'Level' will create a default level asset in the current folder in the content browser.
- By default, a new level has nothing in it. No lights, no actors, no geometry.
- Unlike Unity, the viewport displays black by when there are no lights in the scene.



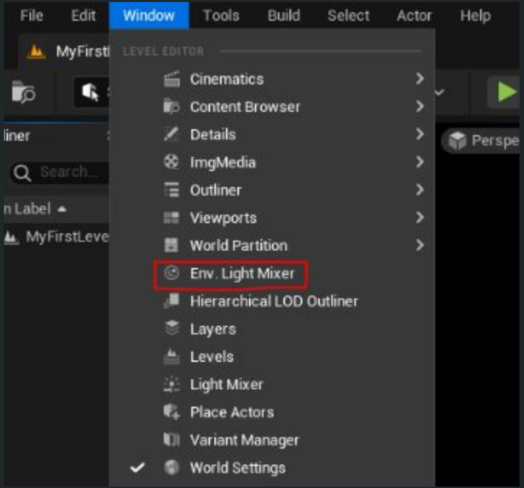
A Level is a collection of Actors (lights, volumes, mesh instances etc.).
Multiple Levels can be loaded and unloaded into the World to create a streaming experience.

@see https://docs.unrealengine.com/latest/INT/Engine/Levels
@see UActor

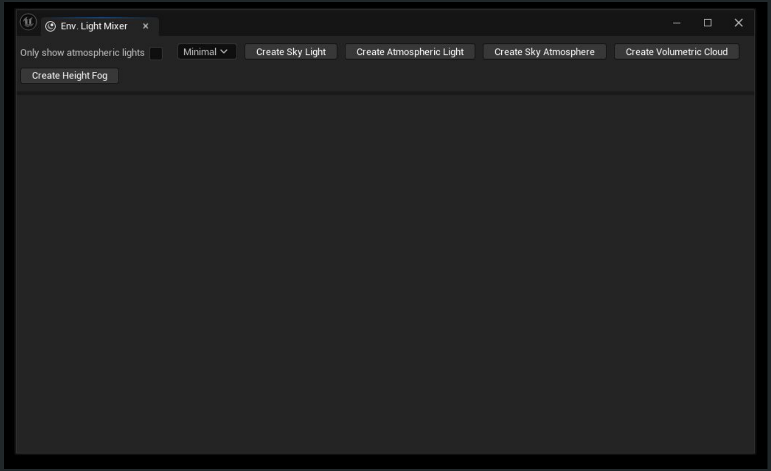
hold (Ctrl + Alt) for more

Level Creation

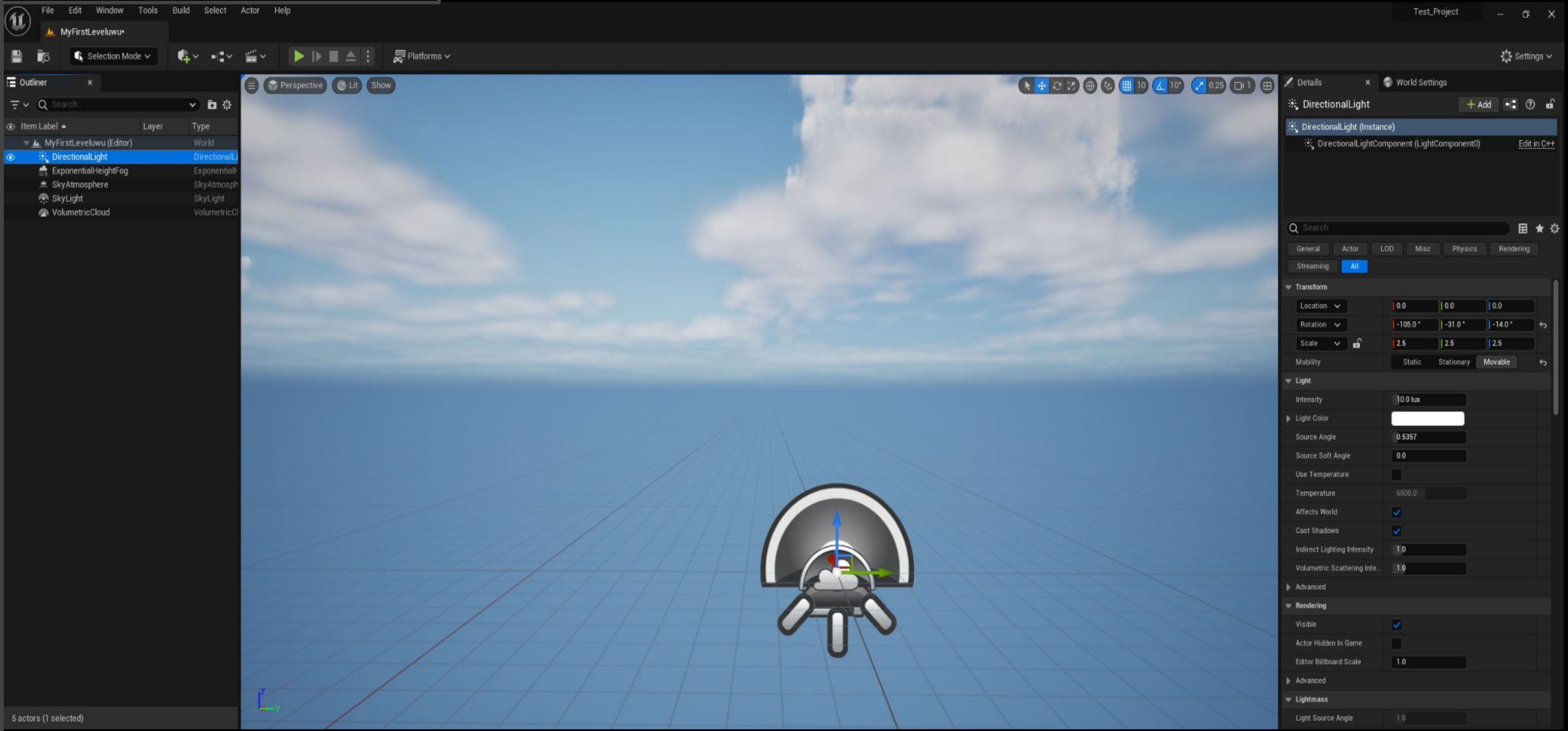
- We can quickly add basic lights to our scene using the Environmental Light Mixer tool found in Main Editor -> Window dropdown -> Env. Light Mixer
- The Environmental Light Mixer allows us to quickly add the set of lights and light utilities that are most commonly used in a Level.



- Selecting 'Create Sky Light' etc. will add the associated object to your current Level.
- Directional lights (referred to as 'Atmospheric Light' in the light mixer) are a type of light source that act as though they are infinitely distant and are often used as the sun or moon in exterior Levels.
- The rest of these lights and utilities are outside of the scope of this tutorial but I encourage you to experiment with and read about them further.



Basic Light Setup



Item Label	Layer	Type
MyFirstLeveluwu (Editor)		World
DirectionalLight		DirectionalLi
ExponentialHeightFog		Exponentiall
SkyAtmosphere		SkyAtmosph
SkyLight		SkyLight
VolumetricCloud		VolumetricCl

5 actors (1 selected)

DirectionalLight

DirectionalLight (Instance)

DirectionalLightComponent (LightComponent0)

General | Actor | LOD | Misc | Physics | Rendering | Streaming | All

Transform

Location	0.0	0.0	0.0
Rotation	-105.0 °	-31.0 °	-14.0 °
Scale	2.5	2.5	2.5
Mobility	Static	Stationary	[✓ Movable]

Light

Intensity	10.0 lux
Light Color	⬜
Source Angle	0.5357
Source Soft Angle	0.0
Use Temperature	☐
Temperature	6500.0
Affects World	✓
Cast Shadows	✓
Indirect Lighting Intensity	1.0
Volumetric Scattering Inte...	1.0

Advanced

Rendering

Visible	✓
Actor Hidden In Game	☐
Editor Billboard Scale	1.0

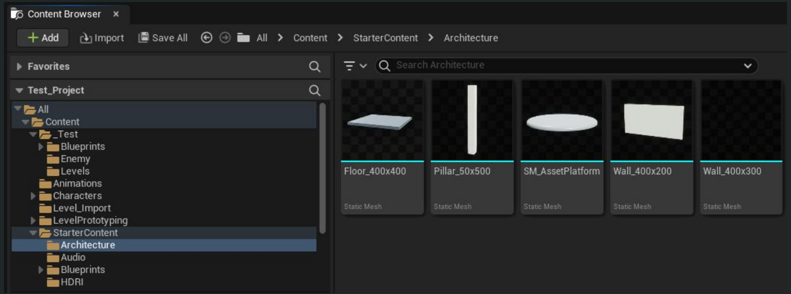
Advanced

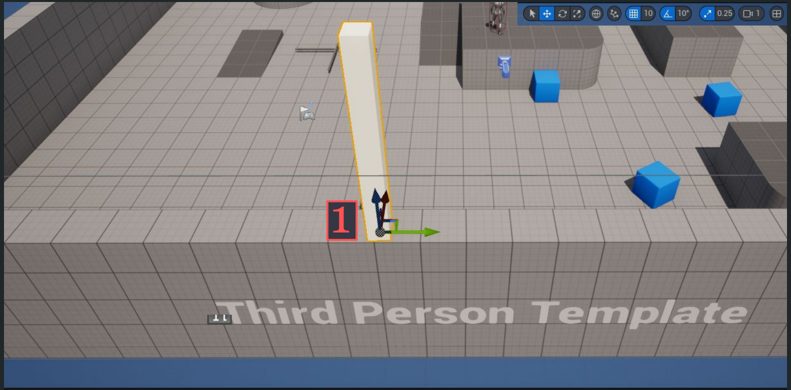
Lightmass

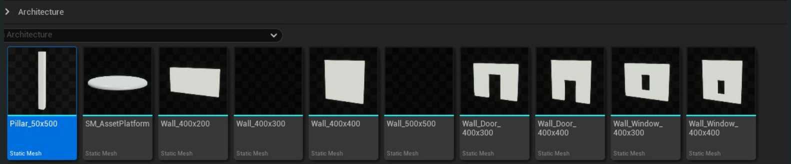
Light Source Angle	1.0

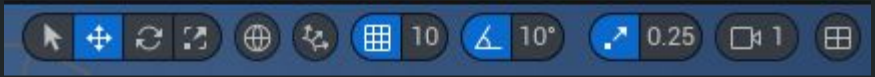
Level Creation

- Once our lights are all set up, we can begin adding assets from the content browser into our Level.
- By clicking and dragging an asset we can add it to our current level.
- The gizmo [1] allows us to manipulate objects in the level.
- Various hotkeys allow us to switch between moving, rotating, or scaling our object:
  - Q: Selection mode
  - W: Movement mode
  - E: Rotation mode
  - R: Scale mode
- Grid / angle / scale snapping can be modified in the upper right of the viewport.







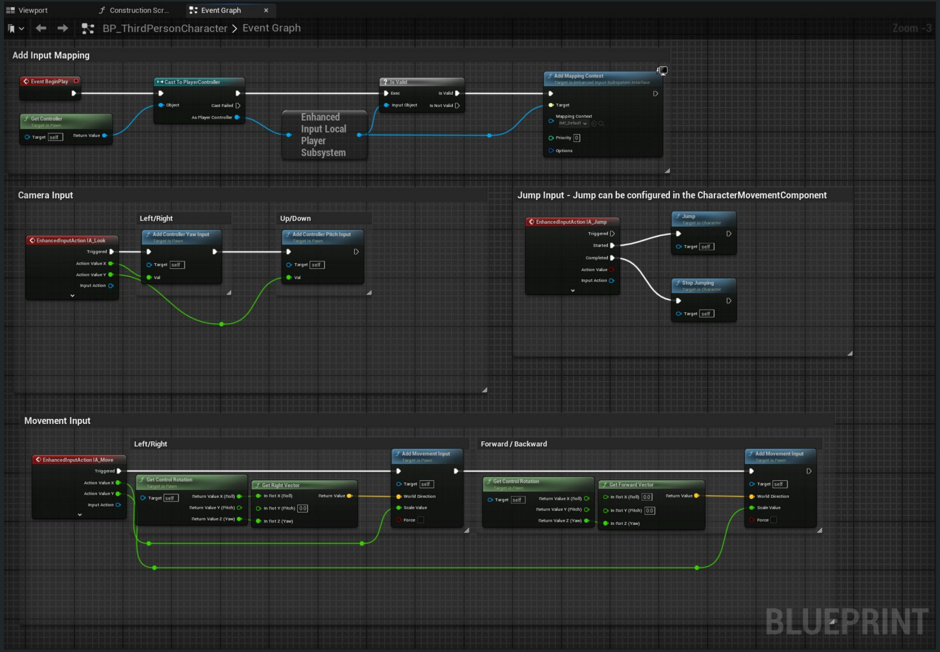


Blueprints

What is a Blueprint?

Blueprints at a glance:

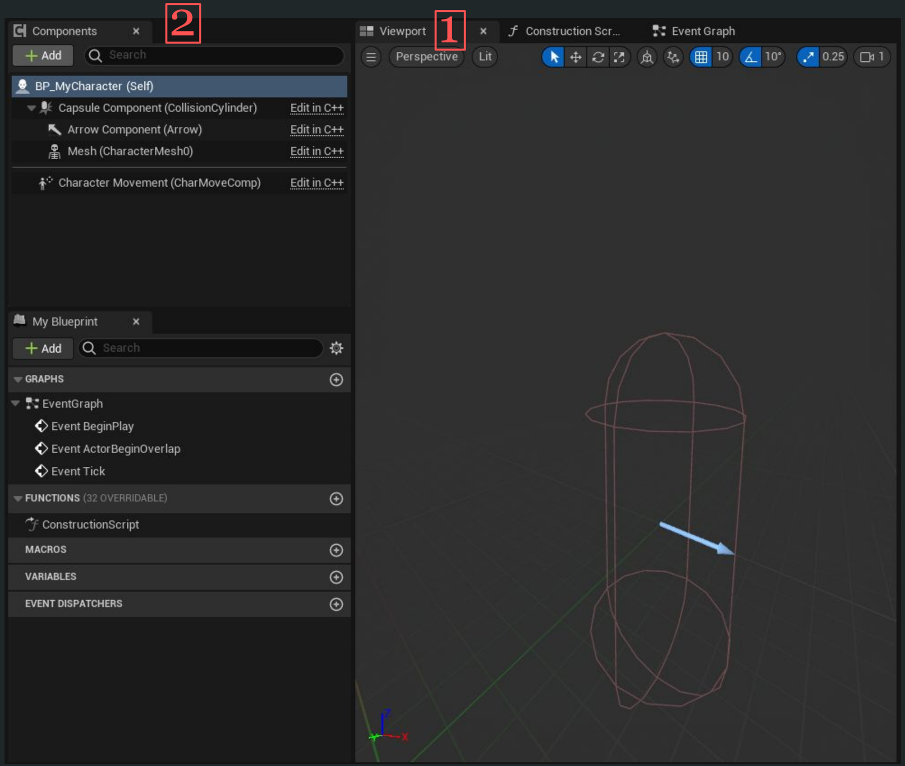
- Blueprints are a type of visual scripting language.
- Blueprint coding is similar to Blender’s geometry nodes and other node-based visual scripting languages.
- Blueprints require more memory than Unreal’s native C++ and sometimes perform worse.
- Many games, even AAA games, have been created purely in Blueprints without issue.
- The best approach is a robust mix of Blueprints and C++.



What is a Blueprint?

Blueprints at a glance:

- Blueprints also allow us to create more advanced assets than a simple static mesh.
- In the 'Viewport' [1] we can see what the Blueprint will look like in a Level.
- In the 'Components' [2] section, we can add or remove pieces of our Blueprint.
- A basic 'Actor' Blueprint is loosely analogous to an empty game object in Unity.
- Note that the component responsible for the root transform of an Actor is variable and can be modified unlike game objects in Unity.



BP_MyCharacter (Self)
Capsule Component (CollisionCylinder) Edit in C++
Arrow Component (Arrow) Edit in C++
Mesh (CharacterMesh0) Edit in C++
Character Movement (CharMoveComp) Edit in C++

EventGraph
Event BeginPlay
Event ActorBeginOverlap
Event Tick
ConstructionScript

Blueprint Pros & Cons

Advantages:

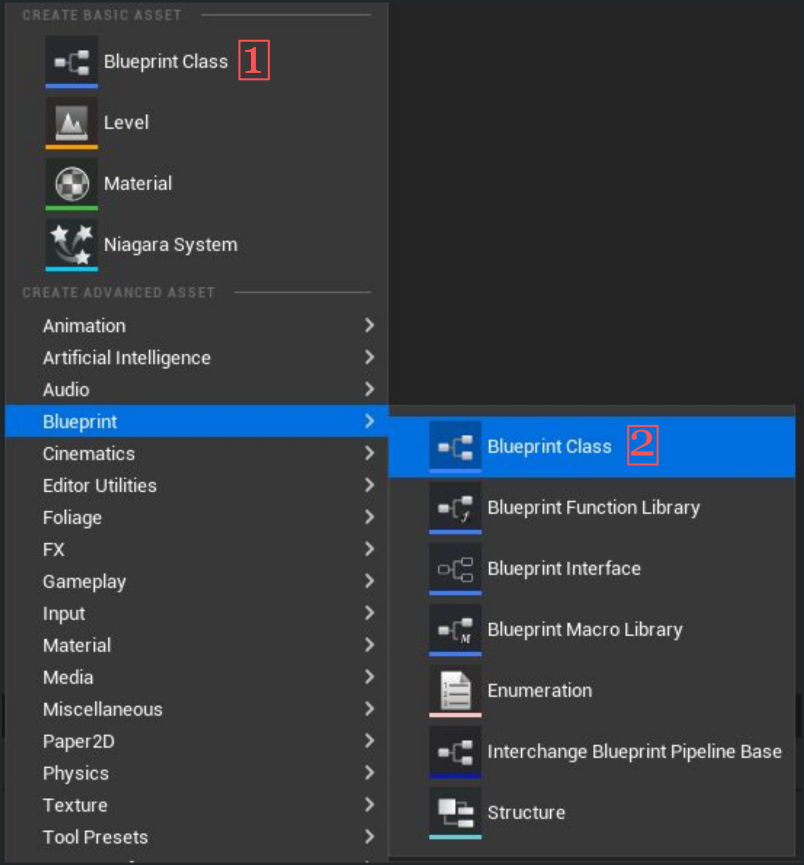
- Blueprints enable rapid prototyping
- They allow for more members of the team to create basic scripted elements by having a lower barrier to entry.
- They allow for existing code to be modified much more easily than native C++ allowing for projects to be more adaptable to changes.
- Blueprints can be extended or have new types created via C++ to allow for easy refinement of the development pipeline without harming existing workflows.
- Blueprints are managed code.

Disadvantages:

- Blueprints are limited in their capabilities. This is because Unreal uses C++ natively. This means that, 'under the hood' everything is in C++. The only functionality available to Blueprints is what's been explicitly made available to them in C++.
- Blueprints incur more of a memory overhead than native C++.
- In certain situations, Blueprints perform worse than C++.
- Blueprints are managed code.

Blueprints In Practical Terms

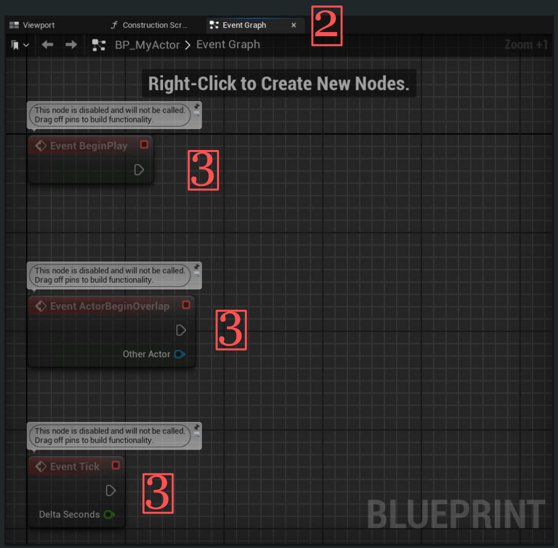
- Blueprints can be created by right-clicking in the content browser.
- A new, basic Blueprint can be created by selecting Blueprint Class. This is available from either the quick access “Create Basic Asset” [1] section or from the “Blueprint” [2] selection menu.
- A quick overview of some of the Blueprint types available in the “Blueprint” [2] menu:
  - Blueprint Class: Opens the submenu of all Blueprint types.
  - Blueprint Interface: Allows Blueprints to communicate with one another without hard references.
  - Enumeration: Used to create a list of named integers.
  - Structure: Used to create a list of named variables.



Blueprints In Practical Terms

- There are several different types of Blueprints. Selecting the correct ‘Parent’ is important.
- Actors are one of the most basic types of Blueprints.
  - They are objects that can appear in a Level.
  - You can script behaviors for an actor.

Many types of Blueprints have the same properties as the ‘Actor’ Blueprint type. These are called ‘children’ and are said to ‘inherit’ these properties from their ‘parent.’



This node is disabled and will not be called.
Drag off pins to build functionality.

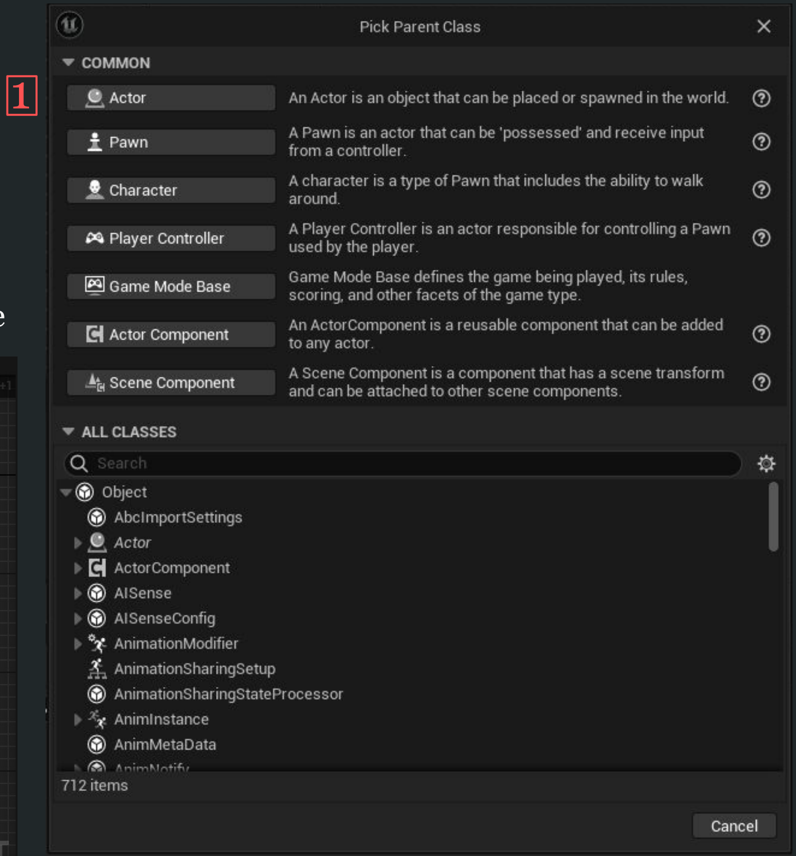
This node is disabled and will not be called.
Drag off pins to build functionality.

This node is disabled and will not be called.
Drag off pins to build functionality.

Here we see the Event Graph [2] of an Actor Blueprint.

The Event Graph is where we create behaviors for an Actor.

The nodes [3] in red begin execution of connected logic.



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.

- Object
  - AbcImportSettings
  - Actor
  - ActorComponent
  - AISense
  - AISenseConfig
  - AnimationModifier
  - AnimationSharingSetup
  - AnimationSharingStateProcessor
  - AnimInstance
  - AnimMetaData
  - AnimNotify

712 items

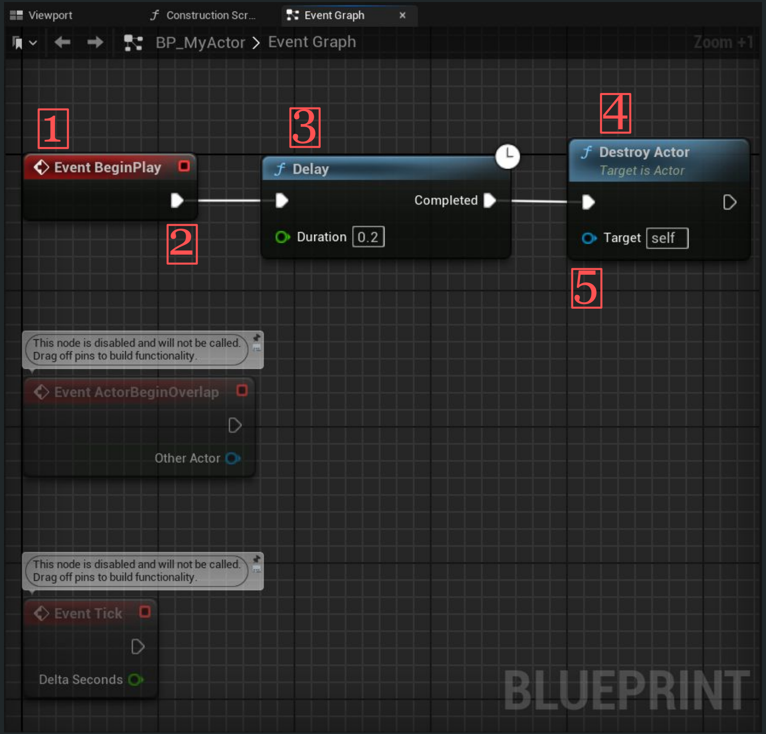
Blueprints In Practical Terms

Example Blueprint:

- The BeginPlay node [1] is an Event node. Events trigger connected nodes when they are triggered. As the name would suggest, BeginPlay triggers as soon as play begins.
- By clicking and dragging from the arrow on the BeginPlay node [2] we are able to set actions that should occur once it has triggered.
- The Delay node [3] allows us to set a configurable delay (in seconds) before the next node can begin.
- The Destroy Actor node [4] destroys whichever Actor we set using the blue input pin [5] on the node.

Blueprint Outcome:

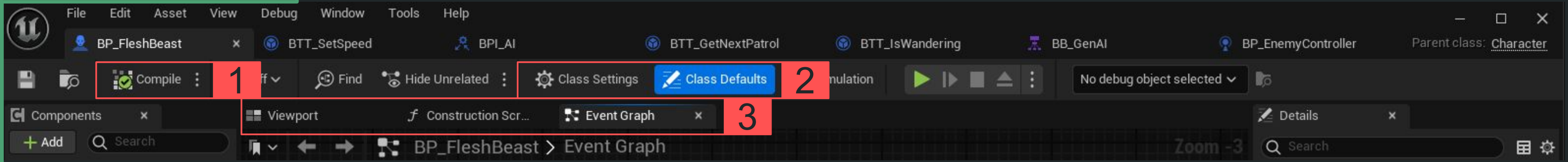
- In this example, the Destroy Actor node is set to its default target, self, which will result in this Blueprint self-destructing after 0.2 seconds.



This node is disabled and will not be called.
Drag off pins to build functionality.

This node is disabled and will not be called.
Drag off pins to build functionality.

Blueprints



1. Compiling

- Blueprints must be compiled before they will function. The compiler will notify you of any errors and sometimes it might even give you a helpful hint about what's wrong.
- Variables cannot have default values set until the Blueprint has been compiled.
- If you're encountering an issue, make sure the current Blueprint is compiled as well as any other Blueprints you're working with.

2. Class Settings & Defaults

- Class Settings allow you to adjust various aspects of the Blueprint.
- Much of this is out of the scope of this discussion, however, it is worth mentioning that Blueprint Interfaces are added in the Class Settings menu.
- Class Defaults contains the adjustable settings for the Blueprint as well as all of its attached components.

3. Viewport, Construction Script, & the Event Graph

- Viewport:
  The Viewport allows you to see how your Blueprint will appear if placed into a Level.
- Construction Script:
  The construction script allows for startup logic on the Blueprint.
- Event Graph:
  This is where the majority of the Blueprint's functionality is held.

Blueprint Parents

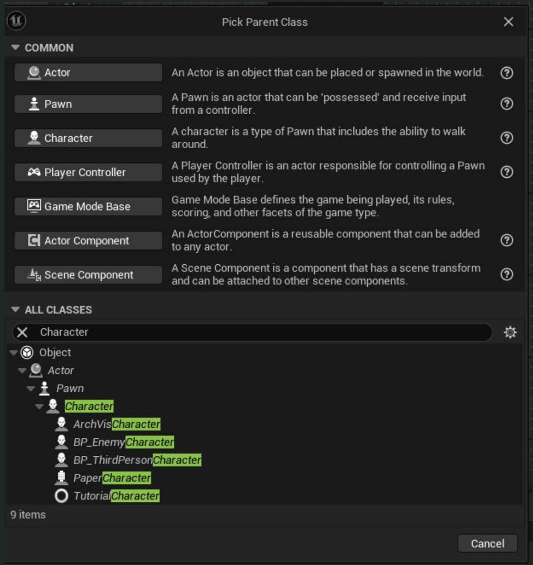
Blueprints can inherit[1] from several different parent classes as we saw earlier. Here are some of the most important ones:

1. Actor: An actor is roughly analogous to Unity's Game Object class. Actors are anything that have a transform[2] and can thus be placed inside of a Level.

2. Pawn: A pawn is a specific type of actor that's designed to be controlled. Examples: player avatars, cars, minigame puzzles, player cursor in an RTS.

3. Player Controller: A player controller is an actor designed to control a pawn. Specific control implementation is often carried inside the pawn, which control set to use is designated by the player controller[3].

4. Character: A type of pawn that has pre-built functionality for moving around in 3D space using the Character Movement component[4].



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 components.

ALL CLASSES

Character

```
Object
  Actor
    Pawn
      Character
        ArchVisCharacter
        BP_EnemyCharacter
        BP_ThirdPersonCharacter
        PaperCharacter
        TutorialCharacter
```

9 items

Cancel

Creating a new Blueprint. This image shows how a Character Blueprint inherits from Pawns which inherit from Actors which inherit from the Object class.

1) *Inheritance*:
When you create a Blueprint you can make it a 'child' of another Blueprint. The new Blueprint has all the functionality of the 'parent' and is said to 'inherit' that functionality.

2) *Transform*:
A set of information used to describe an object in 3D space.
- Location: x | y | z
- Rotation: Pitch | Roll | Yaw
- Scale: x | y | z

Note that z is up in Unreal and that quaternions are hidden from the user.

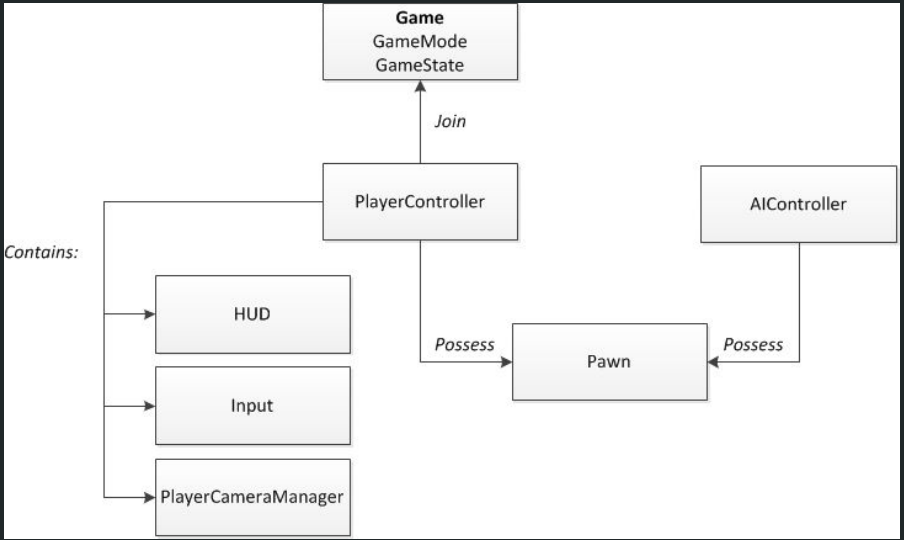
3) *Design*:
Specific implementation can vary, but this is considered best practice.

4) *Components*:
By design, a Blueprint has components that do not necessarily inherit from it. This is a huge design difference as compared with Unity.

The Gameplay Framework

The Gameplay Framework

- Unreal's Gameplay Framework is a set of classes / Blueprints that allow for a modular game creation experience.
- These Blueprints are designed to work with one another but can also function independently.
- Knowing what each of the pieces does and why is important to making informed choices about what a game needs or doesn't.



Gameplay Framework Elements

- Game Instance
- Game Mode
- Game State
- Player State
- Controller
- └ Player Controller
- └ AI Controller
- Pawn
- └ Character

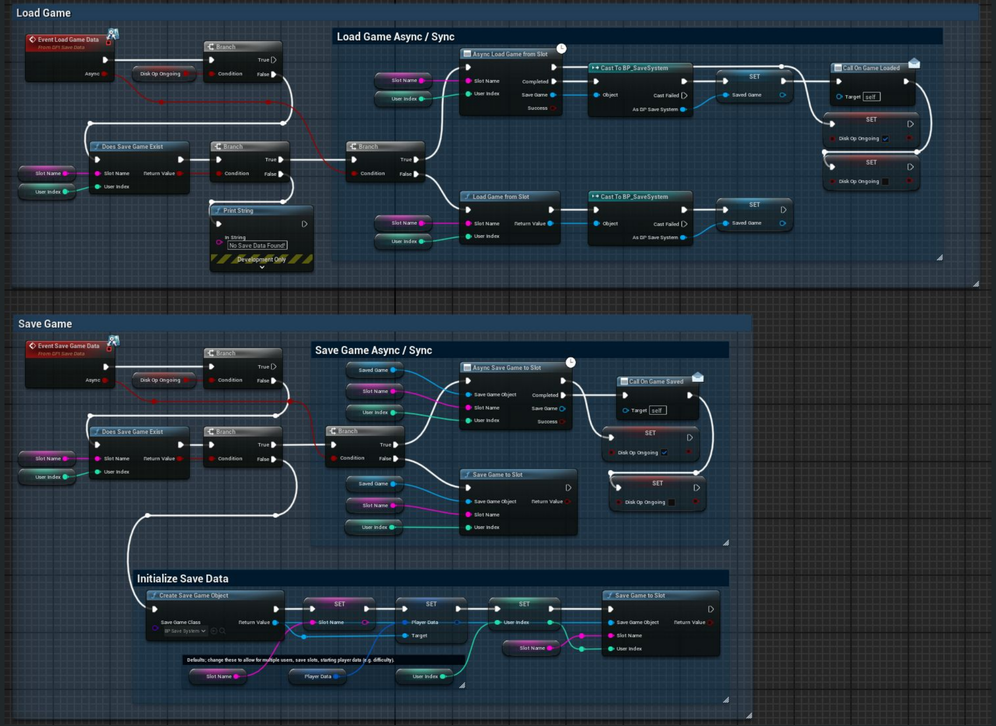
Game Instance

The game instance class is loaded as soon as the game is started.

It is never unloaded until the game is exited.

The game instance class / blueprint is well-suited to holding save system functionality because of its persistence across Level loads.

Key Point: Stuff in the game instance is always available.



Game Mode

In an online game, the game mode is shared with players but can't be modified by them.

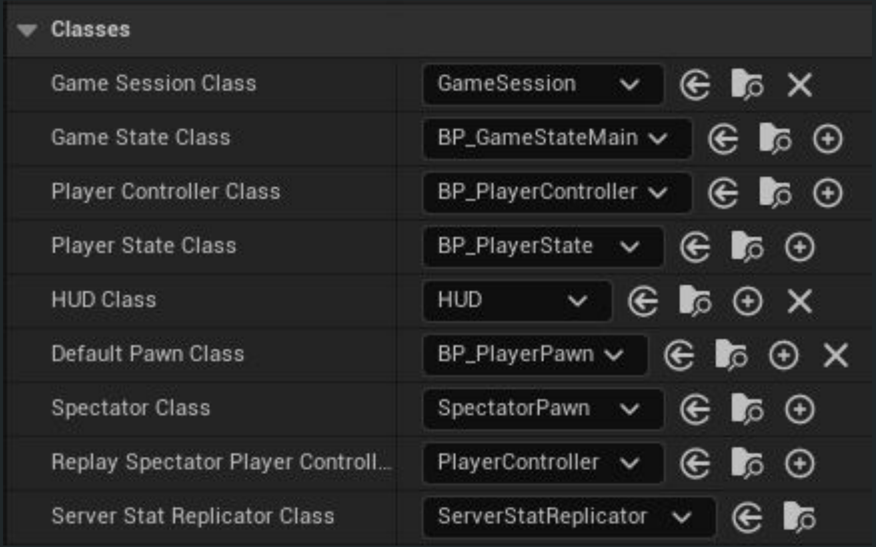
This makes it ideal for setting the rules of the game. The game mode should not track transient data.

The game mode is also critical because it is where we set the required classes Unreal needs to function.

These can also be set in
Main Editor -> Edit -> Project Settings -> Project | Maps & Modes

Note that Game State, Player State, and Player Controller are all set here!

Key Point: Modifying the Game Mode is absolutely critical when creating custom player controllers, etc.



Classes

Game Session Class	GameSession
Game State Class	BP_GameStateMain
Player Controller Class	BP_PlayerController
Player State Class	BP_PlayerState
HUD Class	HUD
Default Pawn Class	BP_PlayerPawn
Spectator Class	SpectatorPawn
Replay Spectator Player Controll...	PlayerController
Server Stat Replicator Class	ServerStatReplicator

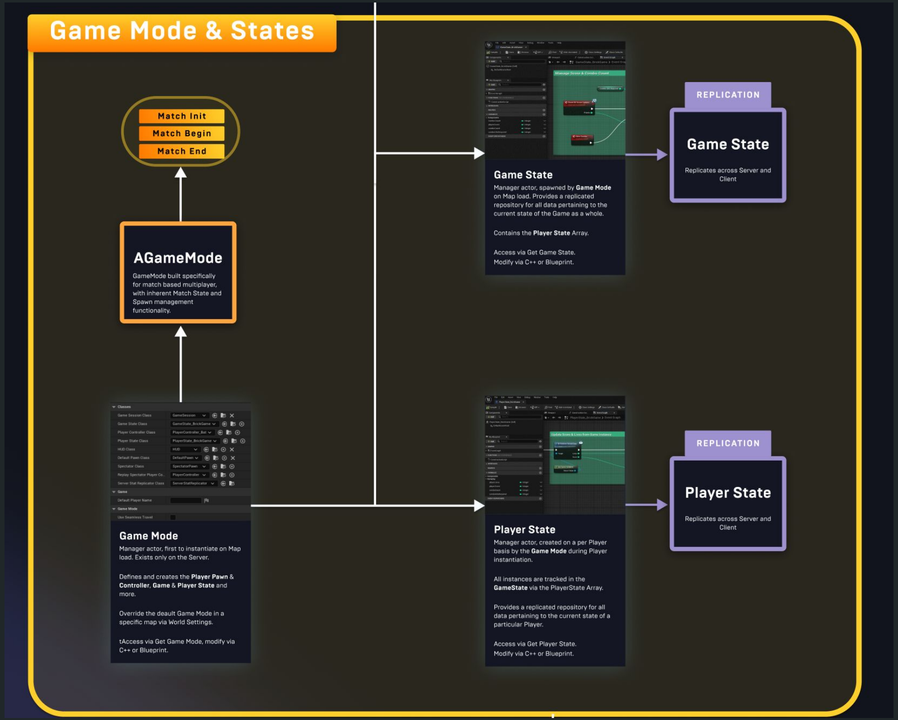
Game State

In an online game, the Game State holds data relevant to all players such as the team's score or the locations of the chess pieces in a chess game.

This necessitates that it be modifiable by any individual player to allow for the score to update.

Game States are **not** persistent when loading a Level.

Key Point: Use this to track and share data about the game world between players in an online game. Also useful offline for tracking the state of the game world.



AGameMode

GameMode built specifically for match based multiplayer, with inherent Match State and Spawn management functionality.

Game Mode

Manager actor, first to instantiate on Map load. Exists only on the Server.

Defines and creates the **Player Pawn** & **Controller**, **Game** & **Player State** and more.

Override the deault Game Mode in a specific map via World Settings.

tAccess via Get Game Mode, modify via C++ or Blueprint.

Game State

Manager actor, spawned by **Game Mode** on Map load. Provides a replicated repository for all data pertaining to the current state of the Game as a whole.

Contains the **Player State** Array.

Access via Get Game State.
Modify via C++ or Blueprint.

Replicates across Server and Client

Player State

Manager actor, created on a per Player basis by the **Game Mode** during Player instantiation.

All instances are tracked in the **GameState** via the PlayerState Array.

Provides a replicated repository for all data pertaining to the current state of a particular Player.

Access via Get Player State.
Modify via C++ or Blueprint.

Replicates across Server and Client

Source: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-framework-quick-reference-in-unreal-engine?application_version=5.3

Player State

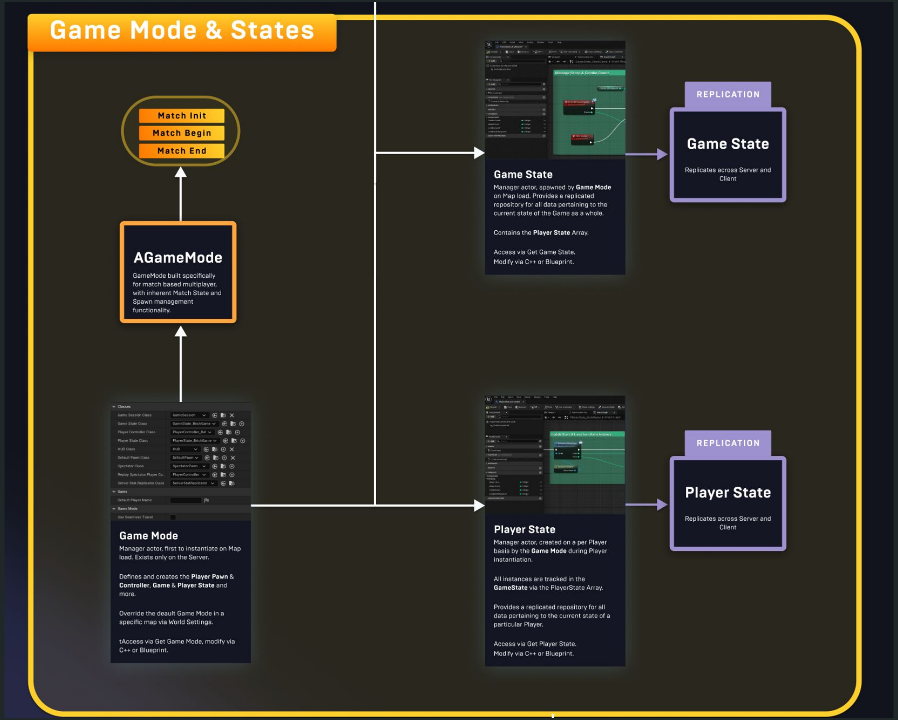
This holds data about the player and can be shared with other players in an online game.

Data such as a player's health or score are good examples of what to store here.

The Player State differs from structs in that it is a full-fledged Blueprint.

Player State is **not** persistent when loading a Level!

Key Point: Use this to track data about the player. Remember that 'hard' loading a Level will cause this to reset to default values!



AGameMode
GameMode built specifically for match based multiplayer, with inherent Match State and Spawn management functionality.

Game Mode
Manager actor, first to instantiate on Map load. Exists only on the Server.

Defines and creates the **Player Pawn** & **Controller**, **Game** & **Player State** and more.

Override the deault Game Mode in a specific map via World Settings.

tAccess via Get Game Mode, modify via C++ or Blueprint.

Game State
Manager actor, spawned by **Game Mode** on Map load. Provides a replicated repository for all data pertaining to the current state of the Game as a whole.

Contains the **Player State** Array.

Access via Get Game State.
Modify via C++ or Blueprint.

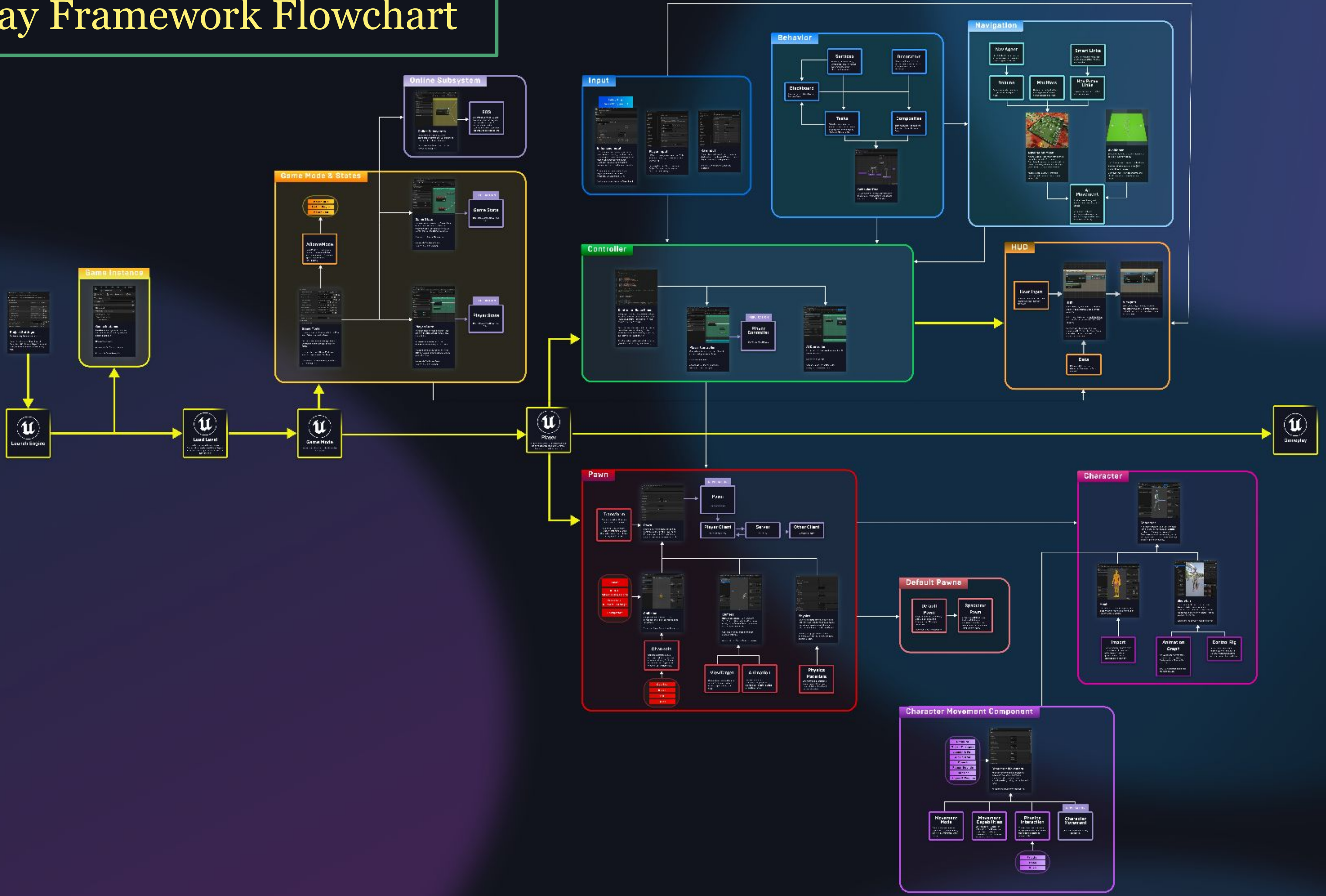
Player State
Manager actor, created on a per Player basis by the **Game Mode** during Player instantiation.

All instances are tracked in the **GameState** via the PlayerState Array.

Provides a replicated repository for all data pertaining to the current state of a particular Player.

Access via Get Player State.
Modify via C++ or Blueprint.

Gameplay Framework Flowchart

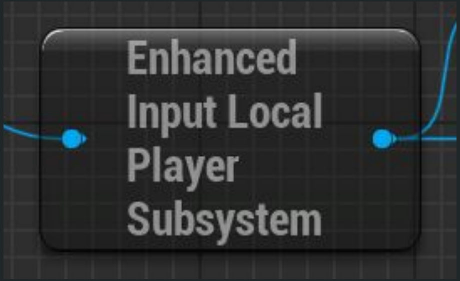


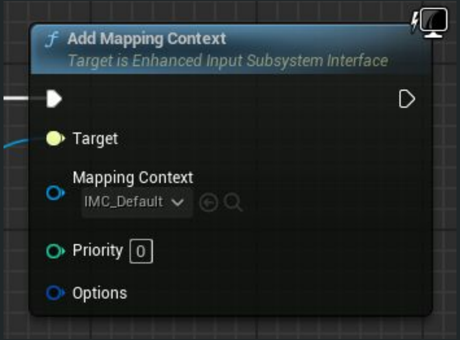
Source: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-framework-in-unreal-engine?application_version=5.3

The Input System

Black Boxes
and How We Learned To Love Them
(aka The Enhanced Input Local Player Subsystem)

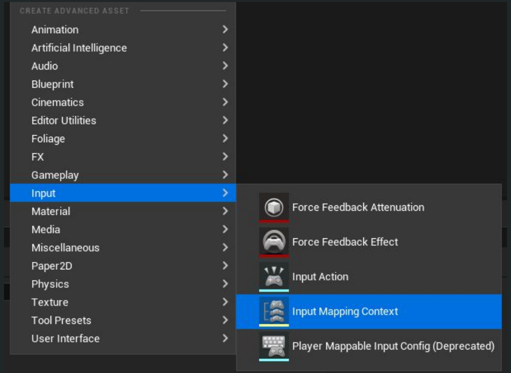
- The Enhanced Input Local Player Subsystem allows for robust, contextually dynamic control layouts.
- Within a Controller Blueprint, you are able to add Mapping Contexts.
- Mapping Contexts determine which set of controls are currently assigned to the Controller.
- Inside a Mapping Context you can bind different keyboard or gamepad inputs to Input Actions.
- Input Actions determine the type of Input they're expecting, such as button presses, joystick movements, mouse movements, etc.
- Multiple Mapping Contexts can be created and switched between from within the Controller based on the situation the player is in such as using different Contexts for controlling an avatar vs. driving a car.

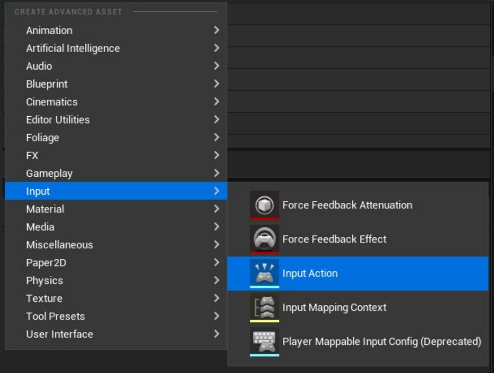


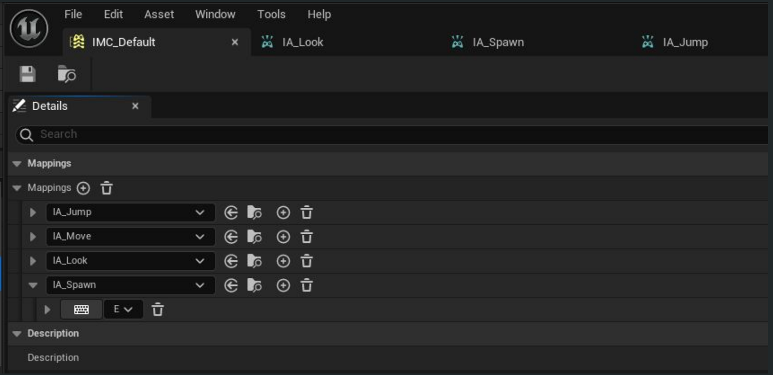

Add Mapping Context
Target is Enhanced Input Subsystem Interface
Target
Mapping Context IMC_Default
Priority 0
Options

Enhanced Input, IMCs, & IAs

In practical terms, here's how the input system works:







Mappings
Mappings
IA_Jump
IA_Move
IA_Look
IA_Spawn
E
Description
Description

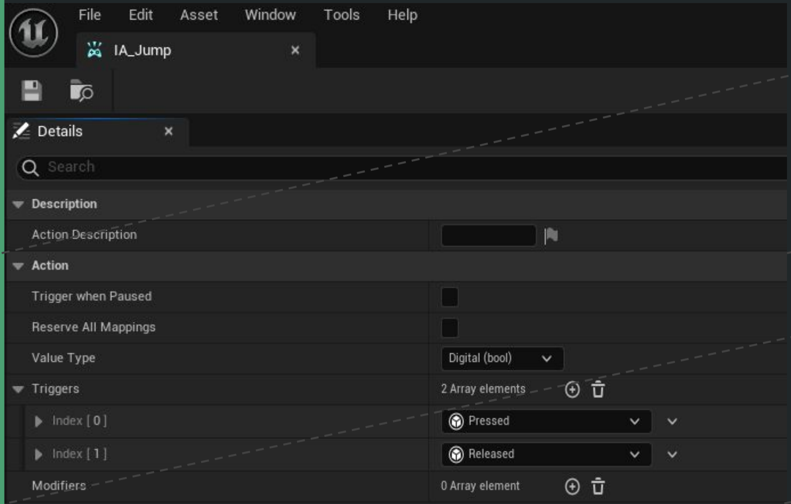
Create an Input Mapping Context
Right-click -> Input -> IMC

Create some Input Actions
Right-click -> Input -> IA

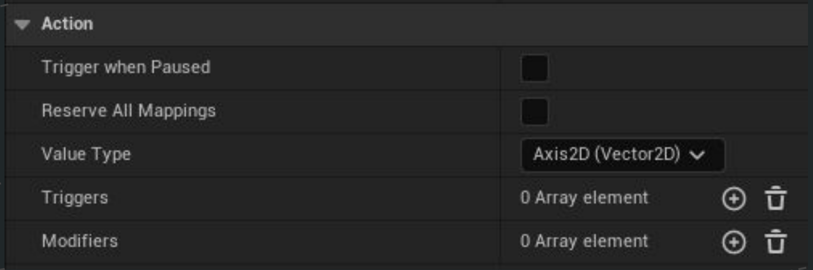
Use the + button in the IMC to add your Input Actions.

Input Actions

When you create an Input Action there are several settings you can adjust to determine how the Input Action triggers.



Description	
Action Description	
Action	
Trigger when Paused	☐
Reserve All Mappings	☐
Value Type	Digital (bool)
Triggers	2 Array elements
Index [0]	Pressed
Index [1]	Released
Modifiers	0 Array element

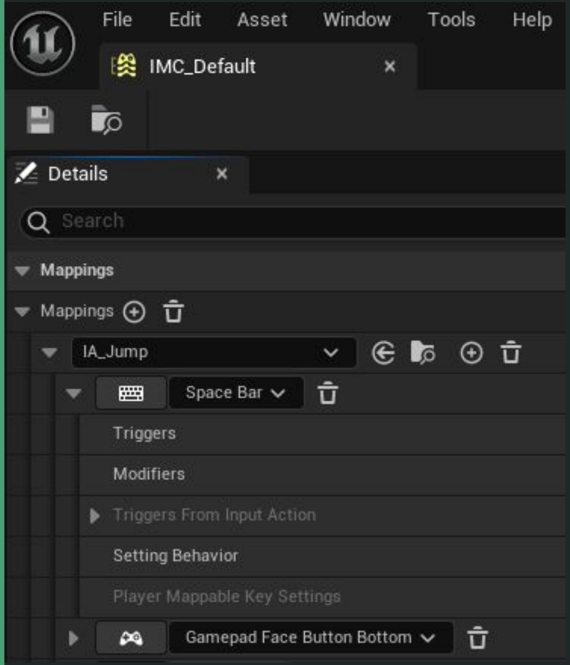


Action	
Trigger when Paused	☐
Reserve All Mappings	☐
Value Type	Axis2D (Vector2D)
Triggers	0 Array element
Modifiers	0 Array element

Here we have a Jump action that is set up to trigger when the key (which is set in the IMC) is either pressed or released. This is useful to allow us to detect the start of a jump while also allowing us to interrupt it if the player wants to begin descending early.

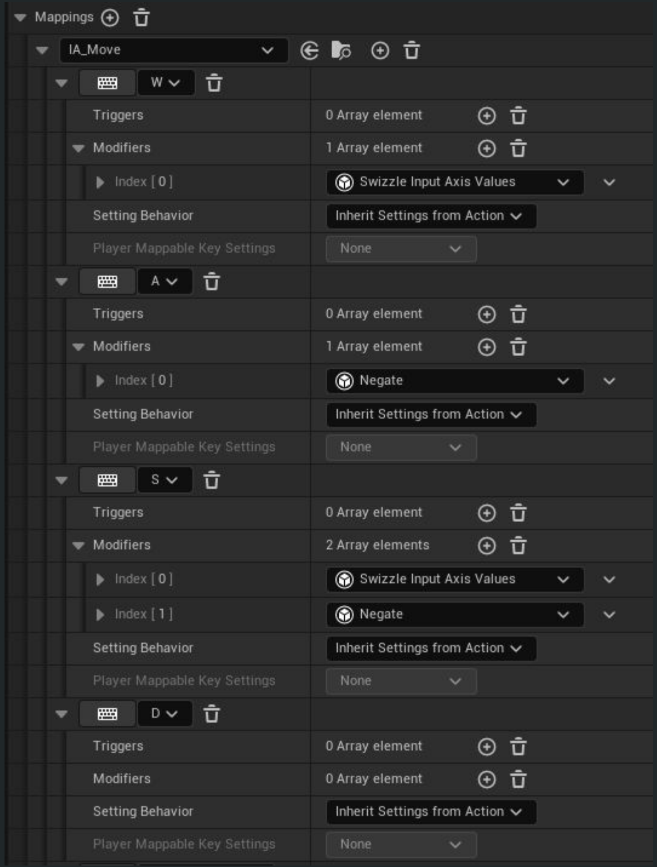
Input Mapping Context

After creating an IMC and adding Input Actions to it, map the keys you'd like to control each action in the IMC.



Mappings
Mappings
IA_Jump
Space Bar
Triggers
Modifiers
Triggers From Input Action
Setting Behavior
Player Mappable Key Settings
Gamepad Face Button Bottom

The Input Mapping Context allows you to associate multiple types of inputs with a given action.



Mappings
IA_Move

W
- Triggers: 0 Array element
- Modifiers: 1 Array element
  - Index [0]: Swizzle Input Axis Values
- Setting Behavior: Inherit Settings from Action
- Player Mappable Key Settings: None

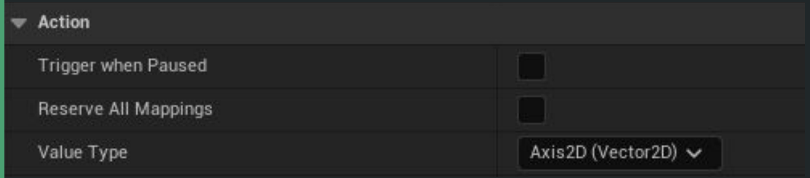
A
- Triggers: 0 Array element
- Modifiers: 1 Array element
  - Index [0]: Negate
- Setting Behavior: Inherit Settings from Action
- Player Mappable Key Settings: None

S
- Triggers: 0 Array element
- Modifiers: 2 Array elements
  - Index [0]: Swizzle Input Axis Values
  - Index [1]: Negate
- Setting Behavior: Inherit Settings from Action
- Player Mappable Key Settings: None

D
- Triggers: 0 Array element
- Modifiers: 0 Array element
- Setting Behavior: Inherit Settings from Action
- Player Mappable Key Settings: None

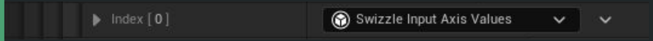
Input Mapping Context

Modifiers may be set in the IMC. These modifiers change the result of an input into something else.



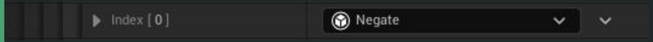
Action	
Trigger when Paused	☐
Reserve All Mappings	☐
Value Type	Axis2D (Vector2D)

If an action is set to 2D Vector, it outputs an x and y value. These values can be anything from -1 to 1, inclusive.



Index [0] Swizzle Input Axis Values

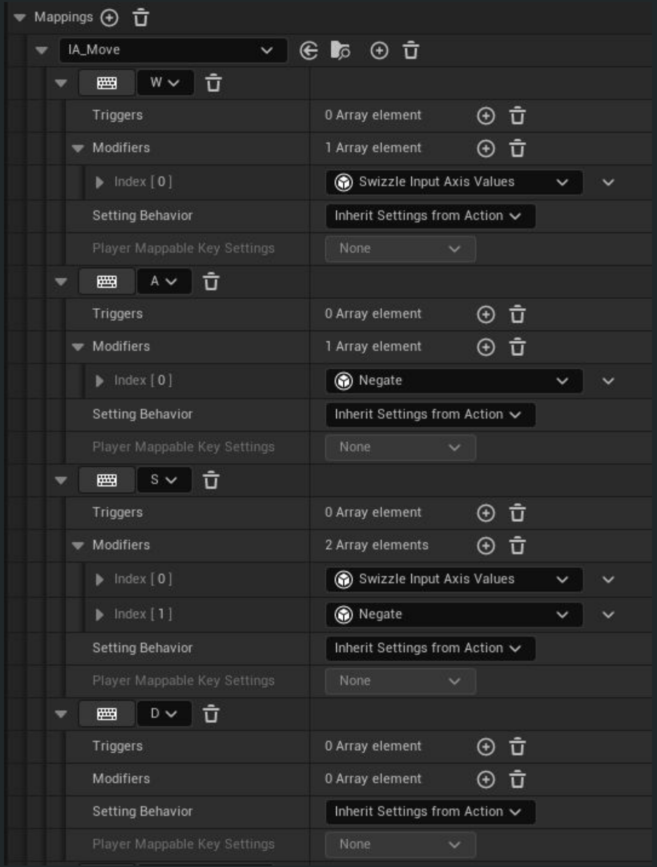
A keypress outputs x = 1, y = 0 when set to be a 2D Vector. If we want x = 0, y = 1 instead we use the 'Swizzle' modifier.



Index [0] Negate

If we want a negative value from a keypress, we use negate.

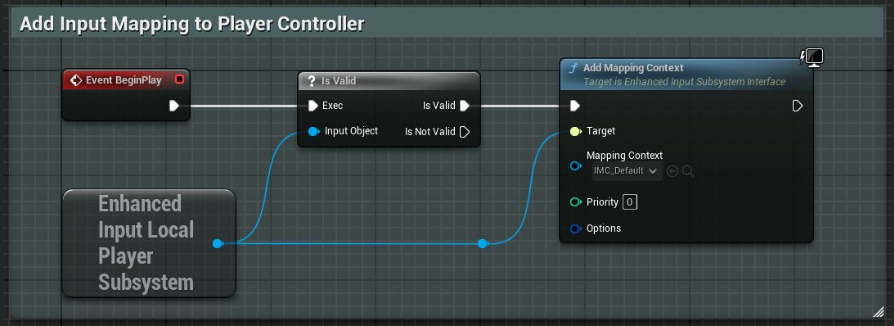
Normal: • Press key -> Output: x = 1, y = 0	Negate: • Press key -> Output: x = -1, y = 0
Swizzle: • Press key -> Output: x = 0, y = 1	Swizzle & Negate: • Press key -> Output: x = 0, y = -1

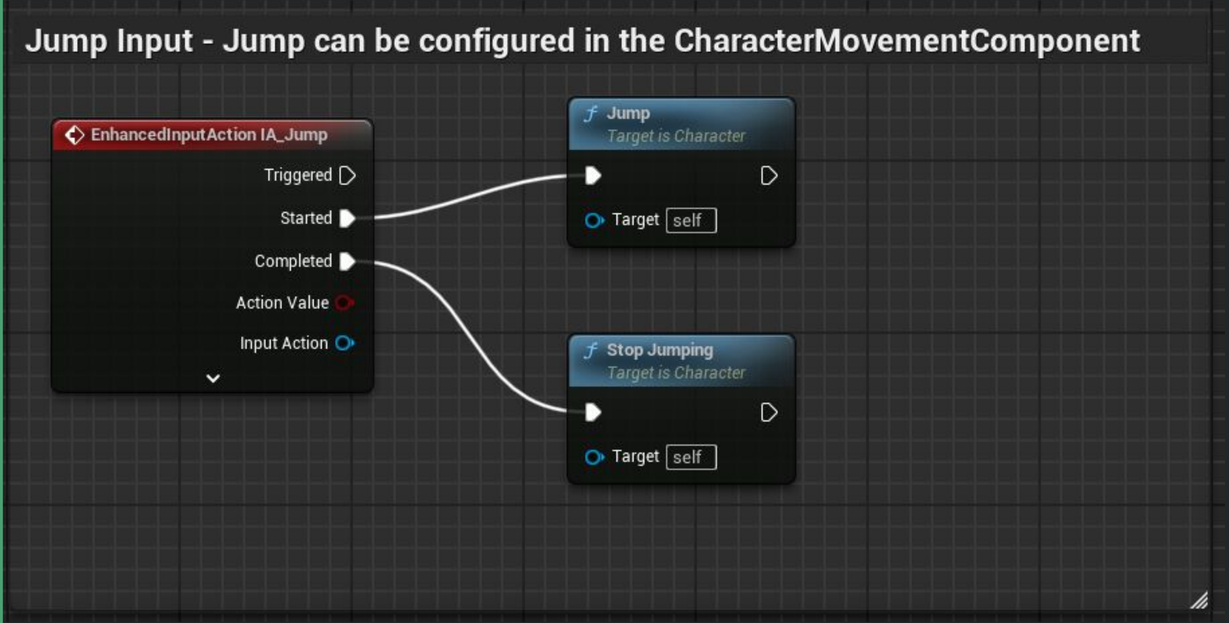


```
Mappings
  IA_Move
    W
      Triggers                      0 Array element
      Modifiers                     1 Array element
        Index [ 0 ]                 Swizzle Input Axis Values
      Setting Behavior              Inherit Settings from Action
      Player Mappable Key Settings  None
    A
      Triggers                      0 Array element
      Modifiers                     1 Array element
        Index [ 0 ]                 Negate
      Setting Behavior              Inherit Settings from Action
      Player Mappable Key Settings  None
    S
      Triggers                      0 Array element
      Modifiers                     2 Array elements
        Index [ 0 ]                 Swizzle Input Axis Values
        Index [ 1 ]                 Negate
      Setting Behavior              Inherit Settings from Action
      Player Mappable Key Settings  None
    D
      Triggers                      0 Array element
      Modifiers                     0 Array element
      Setting Behavior              Inherit Settings from Action
      Player Mappable Key Settings  None
```

Enhanced Input

You're able to access the Input Actions (below) that are associated with the Input Mapping Context(s) that have been set in the relevant Controller (right).



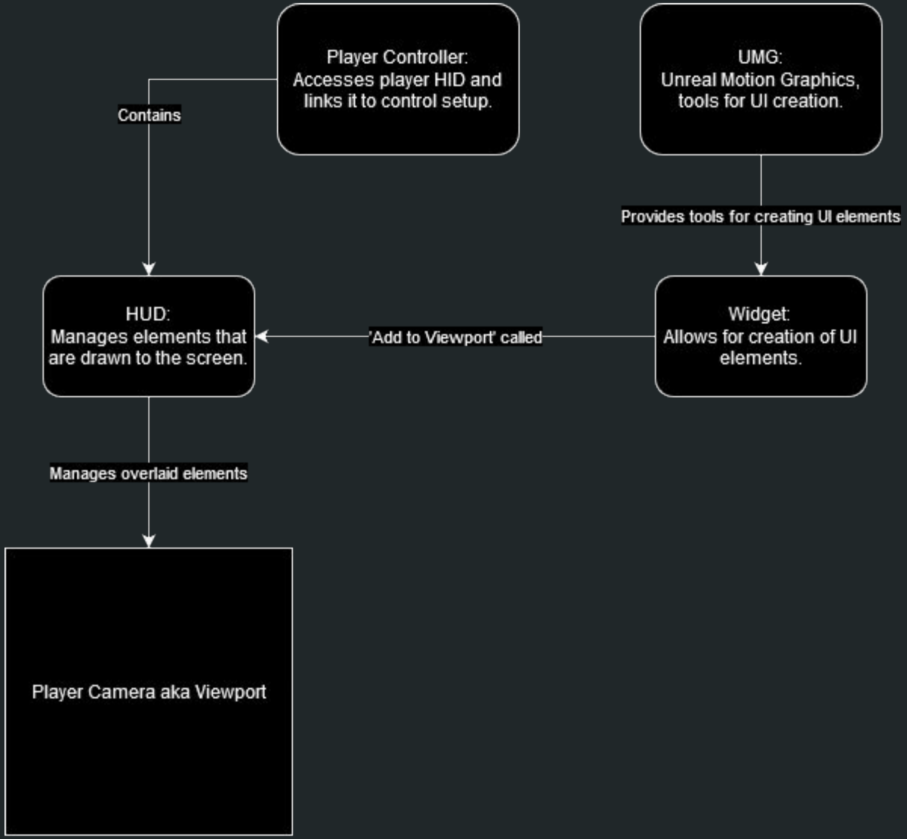


Input Actions are Events that can begin the execution flow of whatever functionality you'd like to implement.

The User Interface

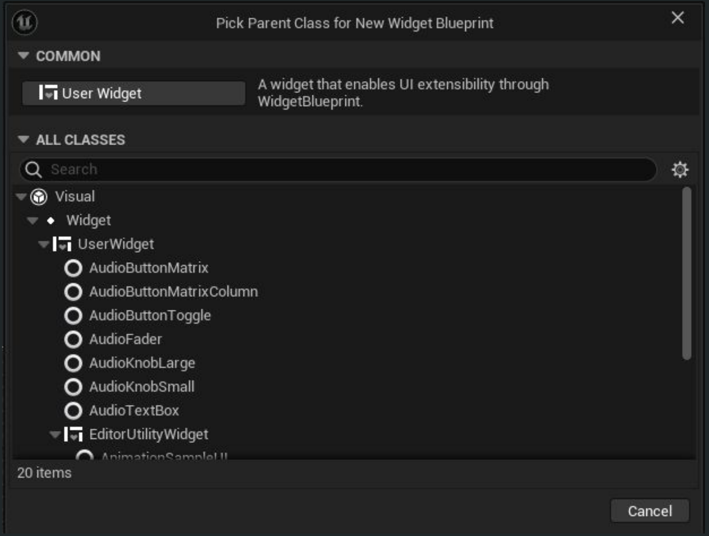
User Interfaces with UMG

- Unreal Motion Graphics aka UMG is Unreal's advanced UI management package.
- As shown in the Gameplay Framework diagram, each Player Controller has a HUD.
- The HUD handles anything that gets drawn to the screen.
- Widgets are a type of object that allow for easy creation of UI elements using the UMG suite of tools.
- When you call "Add to Viewport" for a Widget, the elements that are drawn to the screen are handled by the HUD class.


Player Controller: Accesses player HID and links it to control setup.
UMG: Unreal Motion Graphics, tools for UI creation.
Contains
Provides tools for creating UI elements
HUD: Manages elements that are drawn to the screen.
'Add to Viewport' called
Widget: Allows for creation of UI elements.
Manages overlaid elements
Player Camera aka Viewport

Widgets

- To create a Widget, in the content browser: Right-click -> User Interface -> Widget Blueprint (Shown to the right)
- You'll be prompted with a sub-menu (shown below) for selecting the Widget's parent class. 'User Widget' is most commonly used for UI elements.



Pick Parent Class for New Widget Blueprint

COMMON

User Widget — A widget that enables UI extensibility through WidgetBlueprint.

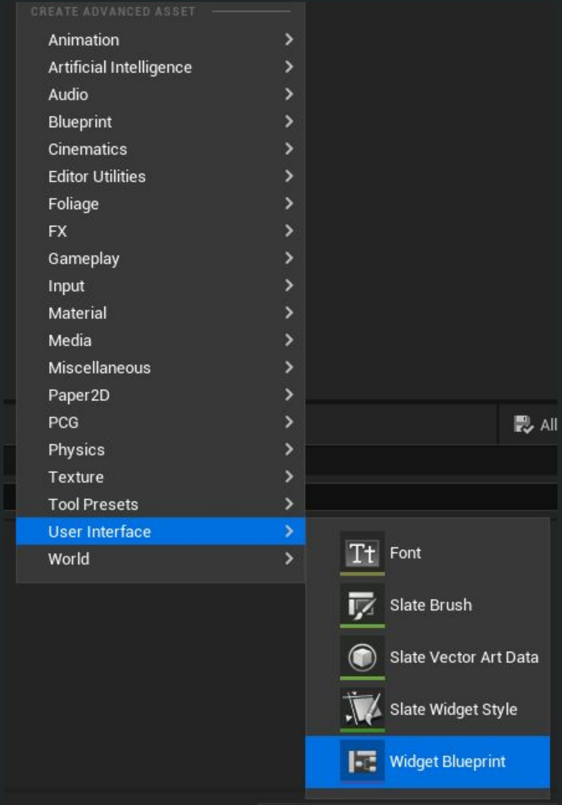
ALL CLASSES

Search

```
Visual
  Widget
    UserWidget
      AudioButtonMatrix
      AudioButtonMatrixColumn
      AudioButtonToggle
      AudioFader
      AudioKnobLarge
      AudioKnobSmall
      AudioTextBox
    EditorUtilityWidget
```

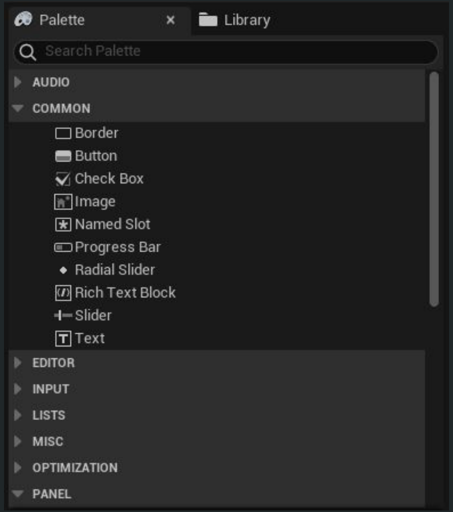
20 items

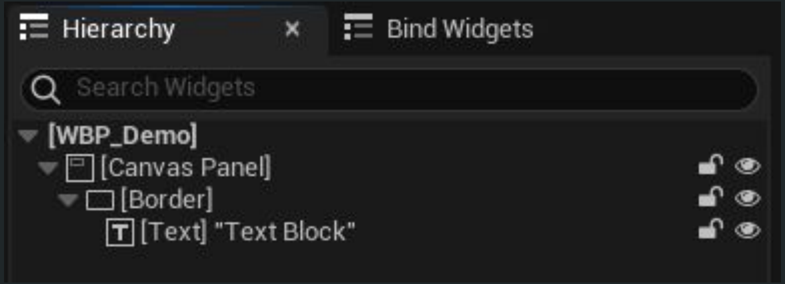
Cancel



Widgets

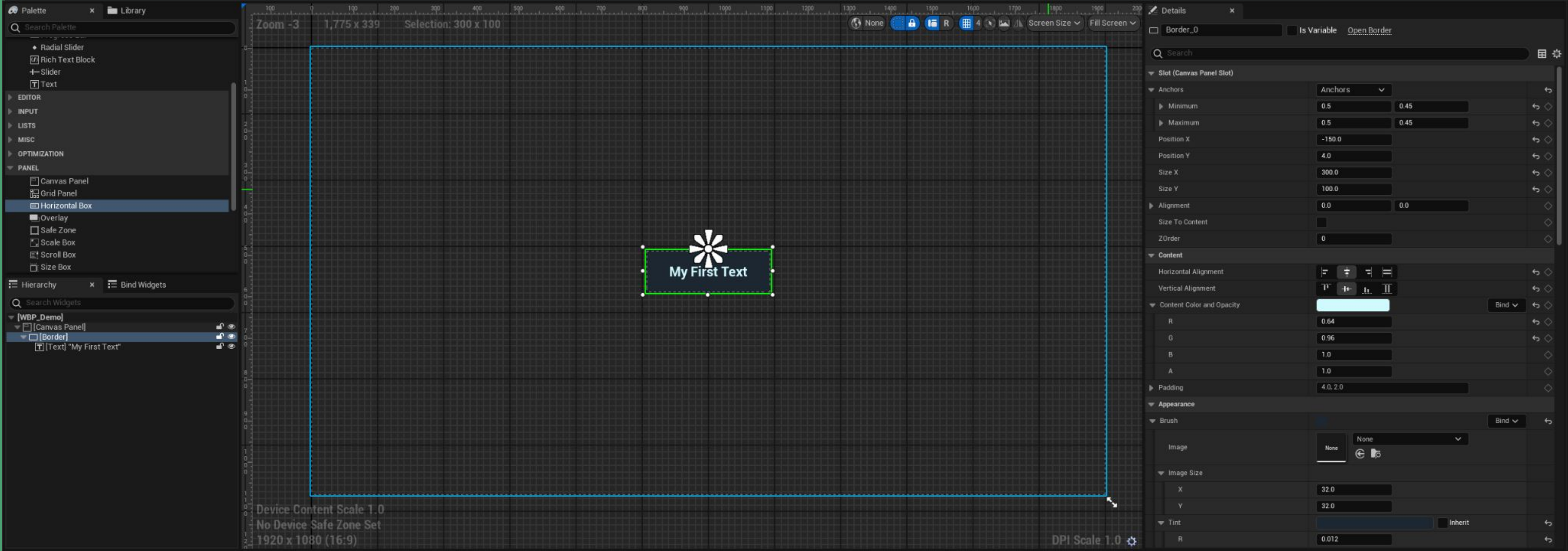
- In the Widget interface, you have access to a Palette.
- The Palette has various options for creating the UI.
- When you add a component to the Widget, such as a button or text, it's shown in the hierarchy.
- Adding a Canvas Panel first is best when creating display elements for the player's view.
- Canvas Panels allow for sub-elements aka children to be anchored wherever you'd like within the Canvas.





Widgets

- As shown below, we have a Canvas Panel, a Border, and a Text object.
- The Canvas Panel defines the area you'd like to arrange elements in.
- Pay attention to the DPI Scale in the lower right.
- The DPI Scale shows the ratio between the Canvas' size and the targeted display size. Here it's 1.0 because the Canvas is 1920x1080 (shown lower left) which is the same as the target monitor.



[WBP_Demo]
[Canvas Panel]
[Border]
[Text] "My First Text"

Zoom -3 1,775 x 339 Selection: 300 x 100

Device Content Scale 1.0
No Device Safe Zone Set
1920 x 1080 (16:9)
DPI Scale 1.0

Border_0 Is Variable Open Border

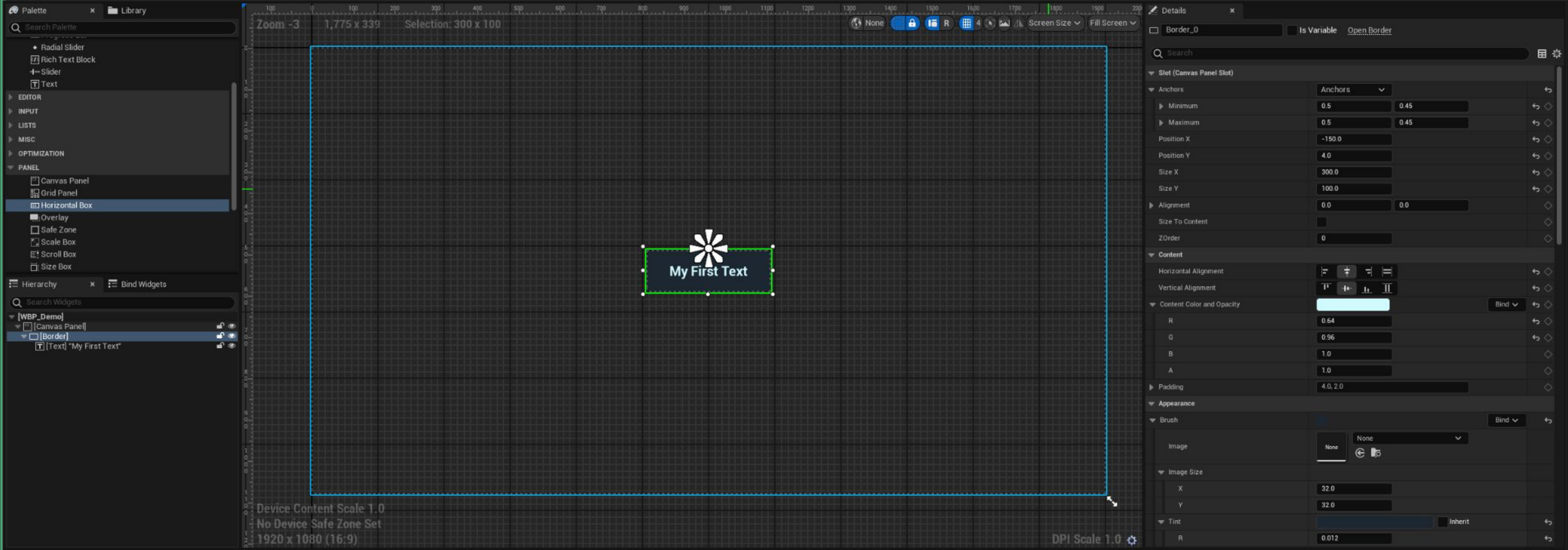
Slot (Canvas Panel Slot)
Anchors: Anchors
Minimum: 0.5 0.45
Maximum: 0.5 0.45
Position X: -150.0
Position Y: 4.0
Size X: 300.0
Size Y: 100.0
Alignment: 0.0 0.0
Size To Content
ZOrder: 0

Content
Horizontal Alignment
Vertical Alignment
Content Color and Opacity
R: 0.64
G: 0.96
B: 1.0
A: 1.0
Padding: 4.0, 2.0

Appearance
Brush
Image: None
Image Size
X: 32.0
Y: 32.0
Tint: Inherit
R: 0.012

Widgets

- The Border object is selected.On the right, we can see we can choose where it should be Anchored.
- When the monitor displaying the Widget is not 1920x1080, the Anchor determines the point from which the element is scaled.
- We can click and drag the diagonal arrow on the Canvas (shown lower right, just above DPI Scale) to test how our UI elements will changed based on the monitor used to display them.

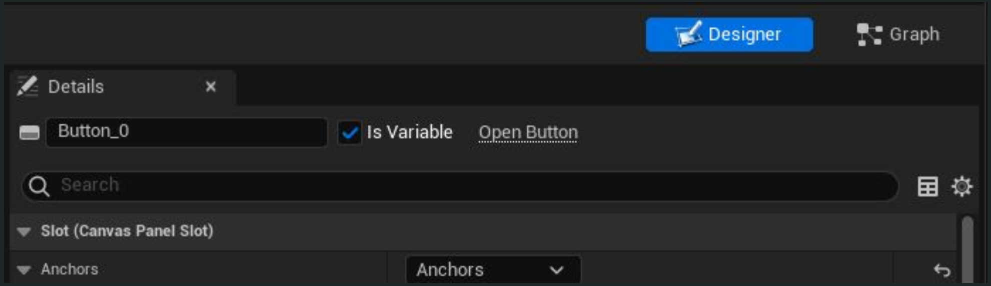


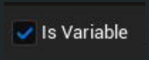
Border_0 · Is Variable · Open Border

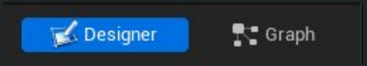
Slot (Canvas Panel Slot)		
Anchors	Anchors	
Minimum	0.5	0.45
Maximum	0.5	0.45
Position X	-150.0	
Position Y	4.0	
Size X	300.0	
Size Y	100.0	
Alignment	0.0	0.0
Size To Content		
ZOrder	0	
Content		
Horizontal Alignment		
Vertical Alignment		
Content Color and Opacity		Bind
R	0.64	
G	0.96	
B	1.0	
A	1.0	
Padding	4.0, 2.0	
Appearance		
Brush		Bind
Image	None	None
Image Size		
X	32.0	
Y	32.0	
Tint		Inherit
R	0.012	

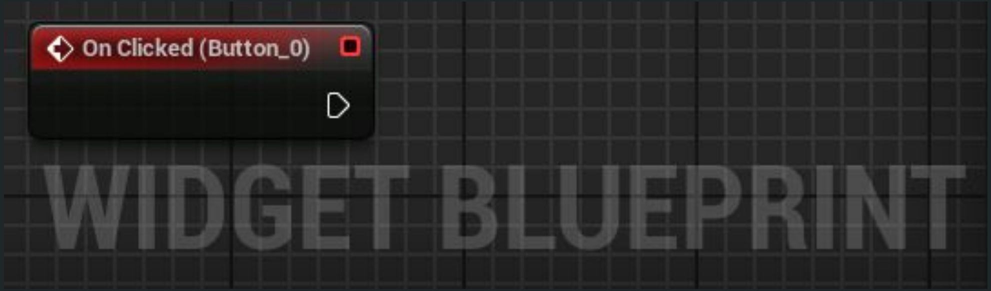
Buttons

- Widgets (aka Widget Blueprints) have access to a full Blueprint event editor, like any other Blueprint.
- ‘Designer’ is where the Widget’s visual components are set up. Selecting ‘Graph’ takes you to the Event Graph.
- Creating a button in a Widget does not expose it to the Blueprint side of the Widget by default.
- Selecting ‘Is Variable’ will allow you to execute logic inside of the Blueprint when the button is clicked.









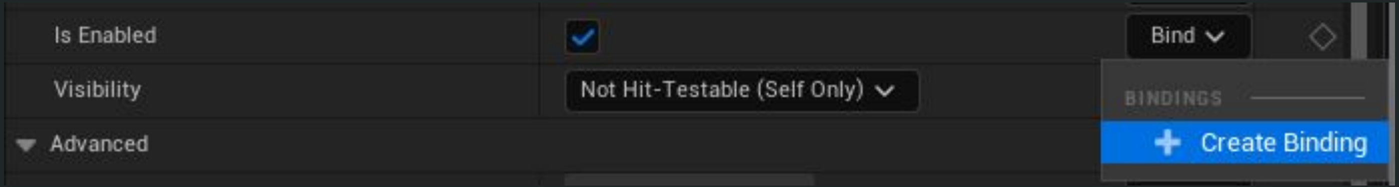
Important Notes

For the efficiency-minded, you should be aware that when you create a binding in a Widget in order to update the associated value, that binding is polled every frame.

This can cause negative performance impacts in situations where you need many different Widgets operating at once.

If a Widget is not in view or otherwise tick-disabled, it will not poll the bound value.

Unreal has this behavior by default in order to allow for smooth updates to UI elements when bound.



A Very Real Thank You

Created & Presented by Els Fouché