Shortcuts:

- CTRL + Space -> Opens Content Drawer
- F10 -> Fullscreen Mode (Docks all tabs)
- Right Click + Scroll -> Adjust Camera Speed
- ALT + RIght Click -> Zoom in/out
- ALT + Left Click -> Pivot around Obj
- G -> Game Mode
- Shift + Drag -> Lock Camera to Obj
- End (key) -> Snaps obj to one below
- CTRL + E -> Open editor for selected asset
- CTRL + B -> Open content drawer to selected asset
- ALT + P -> Auto play game

Exposure Notes:

- Lit -> Game Settings to turn off auto exposure when building
- Post Processing Volume -> Exposure -> Metering Mode -> Manual to turn off in game auto exposure
- Exposure Compensation controls exposure when set to manual
- Post processing volume ONLY affects the camera
- To create new post processing volume and make it global, scroll down to "Post Process Volume Settings" and turn on "Infinite Extent"

Materials:

- Update materials in real time using a material instance and "Convert to Parameters" elements you want to be edited in real time
- Master Material -> Parent material (one material to rule them all)
- You can combine multiple masks into one texture by using RGB channels to hide the masks within.. then just set the R/G/B channels to the specific mask in the material editor

Static Meshes:

- What unreal calls all 3D objects... p much faces and vertices
- Static Mesh (3d object) + Material (Multiple Textures)

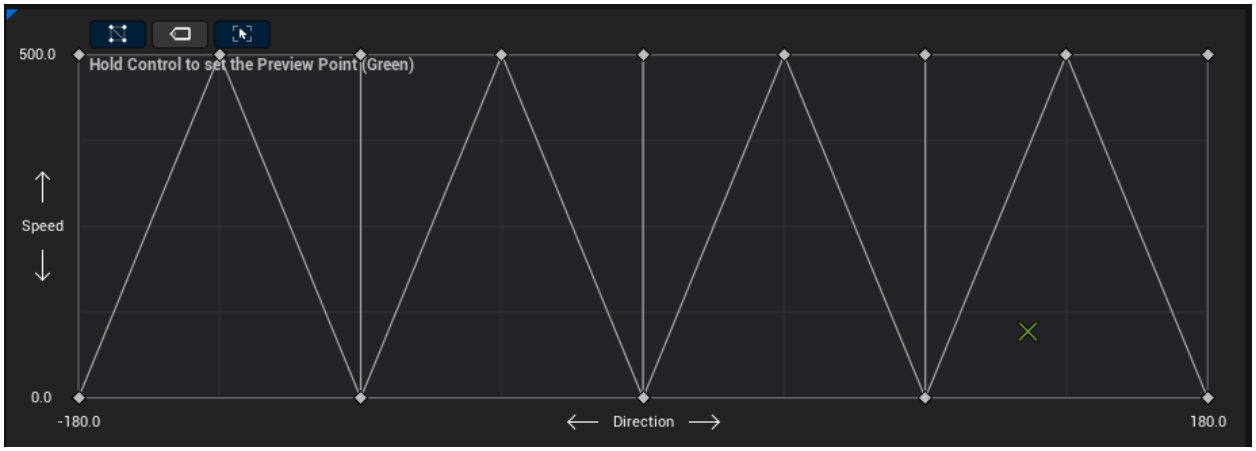
Lighting:

- Lumen vs. Baking
    - Static Lighting Level Scale * Indirect Lighting Quality = 1
    - When Baking lighting, increase the quality of the lighting texture by going to each element's Lighting tab and increasing "Overridden Light Map Res".. Can see light map by going to View Mode -> Optimization View -> Lightmap Density
    - If intensity of shadows between objects is too high, turn off ambient occlusion in the post processing volume
    - When baking, specify where reflections should be by adding a "Sphere Reflection Capture"
    - Sphere Reflection Capture gets 360 degree snapshot of area

- To get better reflections, select Build -> Build Reflection Captures

Blueprint Basics:

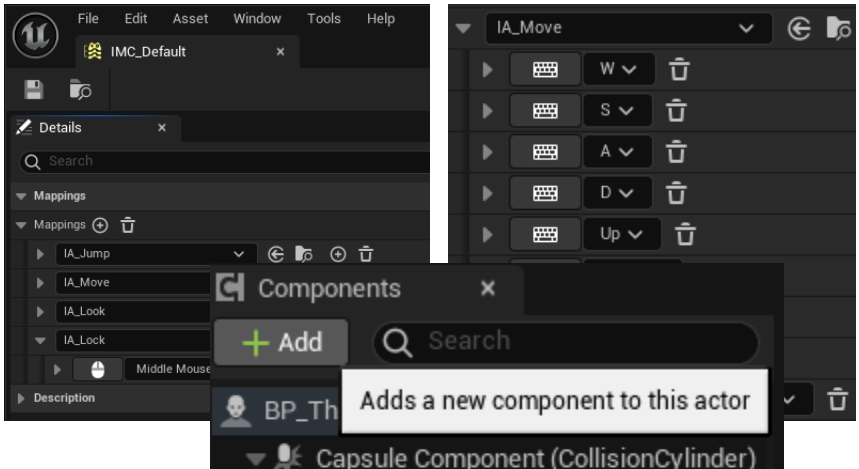
Blend Shapes:
- Used to blend through specific animations (for example, walking in diff directions)
    - Can set parameters like speed/direction/etc and use them to blend between animations
    - Right Click -> Animation -> Blend Space
        - Naming Convention: BS_NameOfThing



Camera:
- Camera Boom -> Connects camera to the player character
    - Camera Lag Param -> Allows for delayed/smooth camera movement

Input:
- Create an “Input Action”
    - Right Click -> Input -> Input Action
    - Naming Convention: IA_NameOfThing
- Inputs can be added to blueprints as “Events” so that every time the input is clicked, the logic connected to the event will be triggered
    - Do this by just searching up the name of the input in the blueprint you wish to add it to… Right Click -> Search IA_NameOfThing

IMC (Input Mapping Manager):
- Stores all the active inputs and the connected keyboard/mouse input
    - Can bind multiple keyboard inputs for



things like “movement”
- Click on the small keyboard and just press button you want to bind the input to

Actor Component:
- A BP class that we can add into other actors to execute logic and make things a bit more organized
    - Naming Convention: BPC_NameOfThing
- Must add to Actor for it to actually work!

AI
- Blackboard
- Behavior Tree
    - Nodes
        - Selector: Executes branches from left-to-right and is typically used to select between subtrees. Selectors stop moving between subtrees when they find a subtree they successfully execute. For example, if the AI is successfully chasing the Player, it will stay in that branch until its execution is finished, then go up to the selector's parent composite to continue the decision flow.
        - Executes branches from left-to-right and is more commonly used to execute a series of children in order. Unlike Selectors, the Sequence continues to execute its children until it reaches a node that fails. For example, if we had a Sequence to move to the Player, check if they are in range, then rotate and attack. If the check if they are in range portion failed, the rotate and attack actions would not be performed
        - Simple Parallel has two "connections". The first one is the Main Task, and it can only be assigned a Task node (meaning no Composites). The second connection (the Background Branch) is the activity that's supposed to be executed while the main Task is still running. Depending on the properties, the Simple Parallel may finish as soon as the Main Task finishes, or wait for the Background Branch to finish as well.
    - Number in top corner of nodes indicates the order of operations
    - Purple nodes are task nodes -> Actions you want the AI to complete
        - Rotate to Face BB entry: node that designates an object you want to rotate and face
- Always select the AI version of Event Receive Execute, Event Receive Abort, and Event Receive Tick if the Agent is an AI Controller. If both generic and AI event versions are implemented, only the more suitable one will be called, meaning the AI version is called for AI, and the generic one otherwise.

Standalone Game -> Can run game as Standalone, which closely represents how it would run on its own
Simulate -> Does not spawn player characters but allows you to see how things will move/function (physics and etc..)

Interesting way to manage managers -> have managers in the level that are accessed through singleton, if the manager does not exist already, spawn in level, destroy everything when you load a new level (easy cleanup and lazy loading)

Unreal Engine Fellowship Course

UE5 Basics

Projects and Templates

Concepts

Blueprints: visual scripting language that allows you to create game logic without code.

- Connect nodes to perform tasks, handle events, or manipulate data.
- Used commonly to compliment code.
- In some cases, it does not run as fast as C++ code due to checks/balances, minimal optimization, overhead, etc…

Classes vs. Instances

- A class serves as a template for something you want to create and can be seen in the content browser.
- An instance is what you get when you drag that class into the world.
    - Each instance can have its own separate settings, even though it originates from the same class.

Inheritance

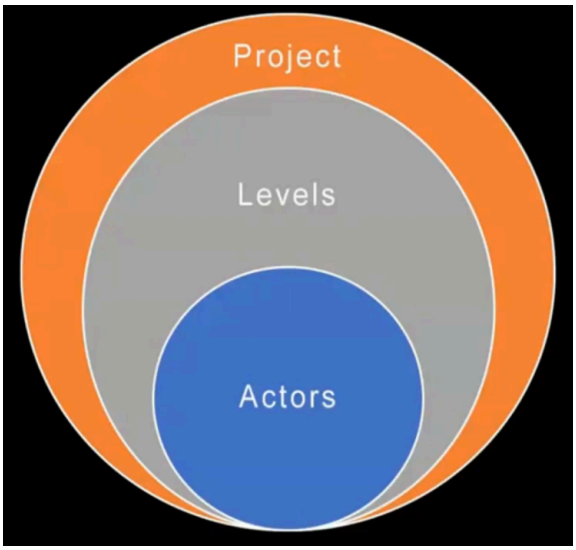
- Inheritance is when a new “child” class takes on properties and behaviors from an existing “parent” class.
    - Allows you to reuse code from the parent class, making it easier to manage and extend functionality.
    - Child classes can have additional features or override the inherited ones.
- Object -> Actor -> Pawn -> Character
    - The Object class in Unreal Engine is the root class that provides core functions like garbage collection.
    - The Actor class has the ability to be spawned in the world, complete with a transform for positioning.
    - The Pawn class can be “possessed” and receive input from a Controller, making it ideal for player or AI-controlled characters.

    - The Character class includes a skeletal mesh and a movement component, allowing for easy setup of animated and controllable characters.

File Saving and Copying

- UE5 File System
    - All imported assets (models, textures, audio, animations, etc) are stored in the Content folder as .uasset files
    - Levels are saved as .umap
    - Projects are saved as .uproject

Project Structure



Viewport

- CTRL + 1 — Creates a bookmark for specific camera position so camera jumps to when you hit (1).. works for all numbers
- End key — Drops object to the ground
- Change grid snap to have more/less control over location of objects (blue nod will turn off grid)

Content Browser

- Collections – Used to keep classes organized
    - Can create custom collections and drag in items to group together so they can all be found in same spot (without changing folder organization)
- Favorites – Can favorite specific folders for easier access
- Filters – Only shows content that has the tags specified

- Right Click Asset -> Migrate if you want a specific asset and all its dependencies
    - Better than manually dragging files from your computer because it takes care of references and imports everything its dependent on (including folder structure)
        - Stuff needs the SAME reference path to not break it… (under the same folders in the project)

Introduction to Blueprints

Concepts

- Blueprints are a visual scripting language that allows you to create game logic without writing code using nodes.

Event Graph

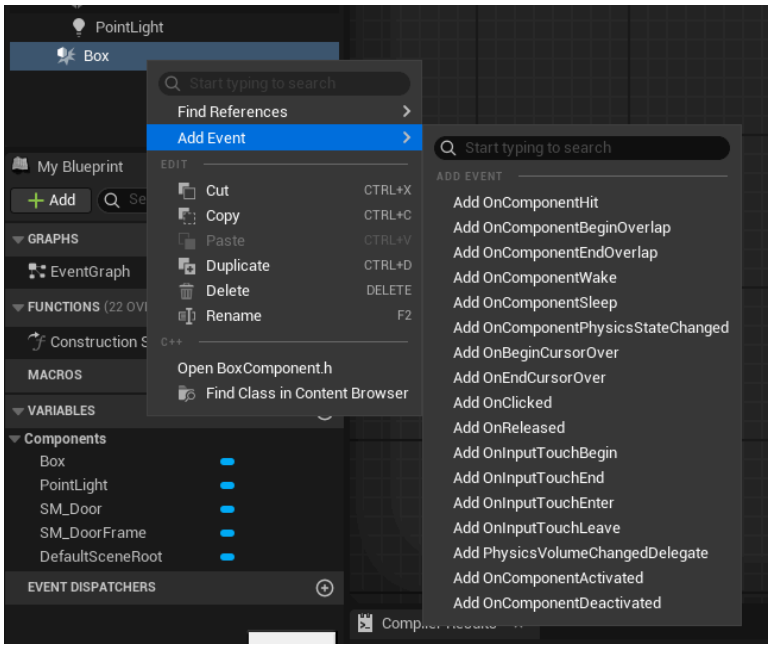
- Branch Node (B + Left Click) – Similar to an if statement
- Sequence Node (S + Left Click)
- Do Once Node (O + Left Click)
- Delay Node (D + Left Click)

Construction Script

- Runs on the point of construction
    - Every time an object is placed or moved around
    - Good when you need to do something when setting up object

Tips

- If you add a component and want an event based on that component (ex, box collision), right click the component and go to “Add Event” for a list of relevant events.



The Unreal Gameplay Framework

Classes Breakdown

- Pawn – the base class for any actor that can be controlled by a player of AI.
    - Provides the capabilities, the controller chooses which of those to use and when.
- Character – a special type of Pawn designed for a vertically-oriented player representation that can walk, run, jump, fly, and swim through the world.
    - Essentially a specialised pawn that can be used as a shortcut to handle a lot of common requirements for pawns.
- Controller – non-physical actors that can possess a Pawn to control its actions. A player Controller is used by human players to control Pawns, while an AI controller implements AI to dictate a Pawn's actions.
    - Actors spawn at runtime
    - Represent "the thing controlling the Pawn"
    - Rotation of the actor is relevant
    - Pawns don't need controllers
- Player Controller – the interface between the Pawn and the human player controlling it. The PlayerController essentially represents the human player's will.
    - Not a specifically highlighted part of the gameplay framework
    - Specialised controller
    - Central place to handle input and player view.
- Game Mode – the primary class that specifies which other classes to use in the gameplay framework and is commonly used to specify game rules for modes, such as capture the flag.
    - Are actors (exist in your level)
    - One per level
    - Controlled via WorldSettings

- Game Instance – persists through the lifetime of the game. Traveling between maps and menus maintain the same instance of this class. Used to manage information and systems that need to exist throughout the lifetime of the game between levels and maps. You can also use the GameInstance class to organize different game instance subsystems.
    - Lives for the duration of the program.
    - Class used for GameInstance is set in project settings.
    - Cannot be deleted/replaced at runtime.
- Game State – contains data and logic relevant to all players in a game, such as team scores, objectives, and a list of all players and their associated player states.
    - Shares data about game state between players in multiple (Game Mode but FOR multiplayer)
    - Mainly relevant for networked multiplayer
    - Acts like a replicated version of GameMode
    - GameMode only present on the server in a networked multiplayer context
- Player State – handles data and logic relevant to its associated player, such as health, ammo count, and inventory.
    - Controller but FOR multiplayer.
    - Primarily exists to be leveraged in networked multiplayer
    - Provides replicated data about the player (distinct from their pawn)
    - Could normally be information on player controller but PlayerControllers only exist locally and on the server.
- HUD – the base object for displaying elements overlaid on the screen. Every human-controlled player in the game has their own instance which draws to their individual viewport.
    - Easy way to make a simple UI
    - Difficult to leverage the tools for anything more complicated or modular
    - Often replaced with UMG widgets

- Camera – represents the player's point of view or how the player sees the world. For this reason, cameras only have relevance to human-controlled players.
    - Can be either an actor or a component
    - Two types, regular and cinematic
    - They don't *have* to be used
    - Actors can have multiple camera components
    - Player view isn't constrained to the pawn they currently have possessed.

<u>Classes but Not Part of Gameplay Framework</u>

- World Settings — where you set and override Level-specific settings. Each level can have unique settings applied to it from the World Settings panel. You can use this panel to do everything from making sure the right Game Mode is activated when you play the level to adjusting global illumination works for that level.
    - Is an actor (although secretly)
    - Base class used for world settings can be set through project settings. But will apply globally across the game.
    - Used to define the game mode used for a level, but only relevant for the persistent level.
- Level Blueprint — specialized type of Blueprint that acts as a level-wide global event graph. Each level in your project has its own Level Blueprint created by default that can be edited within the Unreal Editor, however new Level Blueprints cannot be created through the editor interface.
    - Is an actor (although secretly)
    - Baked into the level
    - Default base class is defined in project settings
    - Often used to implement level-specific logic relating to in-level events such as hitting triggers
    - Can reference any actor in the level
    - "With great power comes great responsibility"
        - Stuff only happening in this specific level is best for level blueprint

Input Mapping Classes (IMC) and Input Actions (IA)

- Input Action – A class you can create in Unreal that is based on taking in player input.
    - Can specify how it is triggered and more
- Input Mapping Class – Contains a collection of specific Input Actions and what keys trigger them
    - Can be added/removed during the game to allow for specific controls at any point
        - For Ex: adding vehicle input mapping to allow player to drive when they enter car and removing it when they leave
            - Helps minimize the need to manage controls that are not needed for the entire game
            - You don't need to worry about what happens when the player hits the "brake" because that control is removed when the player exits car

Introduction to Materials

Real-Time Material Concepts

- What is a Material?

    - An asset applied to a mesh to control its visual look
    - In its simplest form, think of as “Paint” with various properties such as Color and Finish
    - A Material defines how light interacts with the surface it is applied to.
- What is PBR?
    - Unified lighting and shading system
    - Better approximation of light and materials physical interaction
        - Intuitive and consistent
        - Physically accurate
        - Uses real-world physical measurements
    - PBR is a combination of:
        - Materials
        - Lighting
        - Exposure

<u>Primary PBR Inputs</u>

- Base Color
- Metallic
- Specular
- Roughness

<u>Primary Nodes and Textures</u>

- Base Color (Albedo)
    - Flat color without specularity or shading
    - Linear RGB (Vector 3) values between 0~1
    - Avoid pure black (0) and pure white (1) as they don’t really exist in nature
- Metallic
    - Grayscale value that ranges from 0~1
    - Most common usage is on or off (1 or 0)
    - When on (1), it yields a 100% specular reflection
- Roughness
    - Grayscale value that ranges from 0~1
    - Ranges from smooth, mirror-like surface (0) to rougher matte surface (1)

    - Unlike with Metallic, you are encouraged to fine-tune the Roughness value between 0 and 1 or use a Texture to that effect.
    - Roughness is expensive to compute.
- Specular
    - Grayscale value that ranges from 0~1
    - A good rule of thumb is to set the Specular value to 0.5 and fine-tune the Roughness value instead - **On by default set to 0.5 so no node connection needed for it**
    - In some cases, you may decide to edit the roughness value or use a bitmap for that effect.
        - NOTE: used to push stylized looks over the top glossiness BUT not always needed with roughness value
- Guidelines
    - Textures should always be to a power of 2
        - 16x16 all the way to 8192x8192 in size
    - Textures do not have to be square, as long as they are a power of 2
        - 16x128 or 2048x1024 or 2x4096 are all acceptable examples
    - Textures can affect streaming and memory management
    - High-frequency textures can cause anti-aliasing artifacts
- Texture Pyramids (MipMaps) and Custom Settings
    - MipMaps are on by default and help with optimization.
    - Like an LOD for textures.
- Importing Textures
    - Use the Import button or simply drag & drop from Windows Explorer
    - Textures are converted to .uassets
    - You can open a texture in the Texture Editor with a simple double-click
    - Here, you can change the Compression Settings if you need to

    - You can also enable or disable the linear color space (sRGB)

Material Editor and Parent Materials

- Hot Keys for Material Editor
    - Num Key 1, 2, 3, 4 - Variable Constant
    - E Key - Power node
    - R Key - Reflection Vector
    - T Key - Texture Sample
    - U Key - TexCoord
    - B Key - BumpOffset Node
    - N Key - Normalize Node
    - M Key - Multiply Node
    - D Key - Duplicate

Material Notes

- Master Materials
    - The main material that is created
- Material Instance
    - A material that inherits the properties and parameters from master material and can be edited without affecting original main material

Introduction to Lighting

Lighting Definitions and Types

- Direct Lighting – light that falls onto a surface without any interference. The light travels directly to the surface and that surface receives the full color spectrum of the light.
- Indirect (or bounced) Lighting – lighting that has been reflected by other surfaces nearby. As light bounces off these surfaces, light waves are absorbed or reflected based on surfaces properties and passed on to the surface you are looking at. Thus, indirect lighting contributes to the mood and overall light intensity.
- Shadows – when Unreal Engine takes a snapshot of a mesh actor from a light's POV and projects that information onto another mesh actor on the opposite side. In the case of Virtual

shadow Maps, the shadows update in real-time using mesh distance fields for accurate calculations and updates with moveable lights.

Static vs Dynamic Lighting

- Lumen – Unreal Engine 5's fully dynamic global illumination and reflections system that is designed for next-generation consoles, and it is the default global illumination and reflections system. Lumen renders diffuse interreflection with infinite bounces and indirect specular reflections in large, detailed environments at scales ranging from millimeters to kilometers.
    - Note: Lights in scene MUST BE SET TO MOVABLE for Lumen to work on them!!!!!
- Baked (Static) Lighting – pre-computed/rendered lighting. Baked lighting is created using Unreal Lightmass radiosity baker. Similar to rendering lighting with VRay, Corona etc. It is used to create high quality indirect lighting using light and shadow maps. Baked lighting cannot be altered at run-time, but has little impact on real-time performance.
- Real-Time (Dynamic) Lighting – lighting that is updated each frame. It is fully dynamic and can be moved around and modified at run-time. It has no global illumination component (only direct light). It is also expensive to use (affects real-time performance).
    - Unreal can use both static and dynamic lighting. For Arch/Vis projects where assets are typically stationary,baked (static) lighting is mostly used. - (Older way) NOTE: Lumen can be used but there is currently some reflection limits.
- Raytracing – This type of lighting is real-time global illumination updated with all lights set to moveable. If they are animated the GI updates on the fly.
    - NOTE: Some aspects of RT is taking into account with Lumen such as reflections some light calculations

Lumen Settings

- Project Settings -> Rendering

    - Global Illumination -> Set to Lumen
    - Lumen -> Use Hardware Ray Tracing when available (On)
    - Ray Lighting Mode -> Set to Hit Lighting for Reflections
    - Software Ray Tracing Mode -> Set to Detail Tracking
- Modeling Tools -> Enable Ray Tracing While Editing (On)
- SM6 -> Turn on for D3D12 Targeted Shader Formats
    - Helps with virtual shadow maps

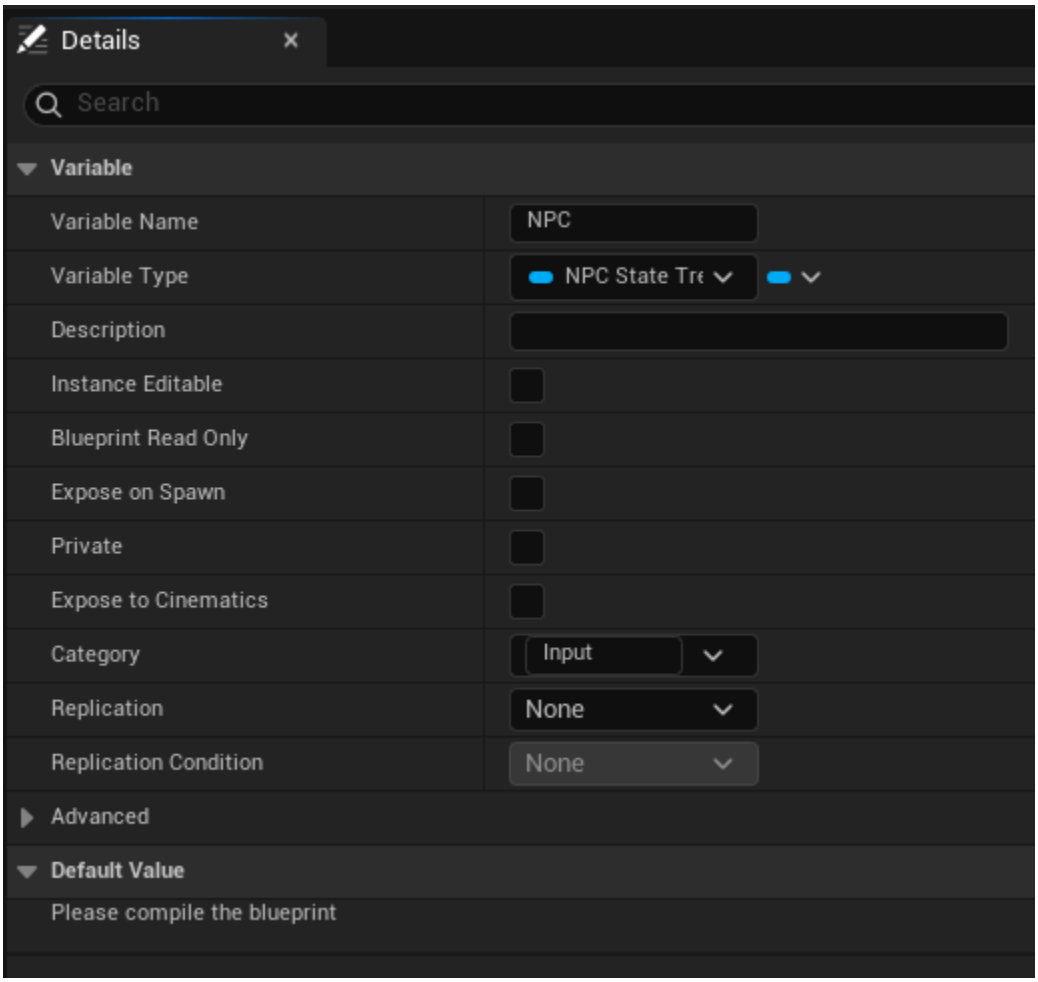
<u>Behavior Trees</u>

- A framework used for creating complex AI behaviours, using a hierarchical structure to manage decisions and actions.
    - Behaviour Trees = The Brain
        - Visually organizes the sequence of tasks & their dependencies.
    - Blackboard = Memory
        - Store variables that can be easily accessed by the behaviour tree.
    - Task = Actions
        - Scripted actions that can be given to the AI controller.
    - Decorator = Filters
        - Decorators can be used to control the flow in the behaviour tree.
    - Service = Checker
        - The service will run periodically to update the blackboard.
- Decorators and Services are added onto existing nodes to change behaviour.
- Tasks are used to create nodes in which the AI does things or stores information or etc..

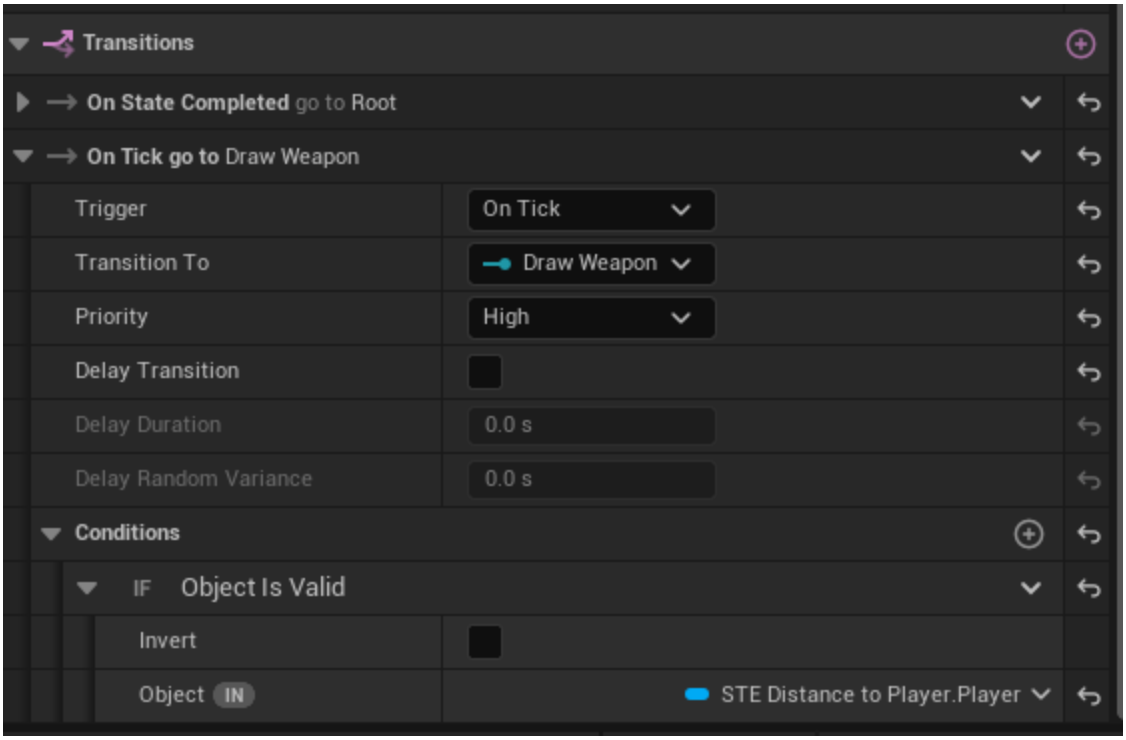
<u>State Tree</u>

- Must be turned on in plug-ins
- Provides better control on transitioning between states compared to BT
    - Honestly, BT is an older legacy.. Phase out from BT to State Machines

- State Tree is added to a pawn through a component! And you set the class of the state tree after.
- When setting variables in task for State Tree, you can set the Category to “Input” or “Output” to require them for the task to run



- You can use evaluators to swap between states in transitions. Variables for evaluators are just stored in the evals so there’s no BB to retrieve it from like in BT.



Material Param Collection:

- Can be used to create globally stored variables for materials so that all the materials can be edited at the same time.
    - For Example, if you have a speed for machine, you can decrease the speed to stop all parts of the machine that are connected via diff materials.