GAS System

GAS Documentation

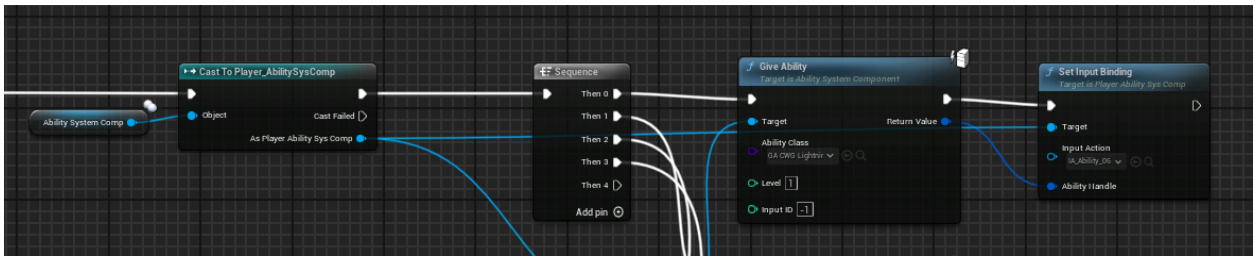
https://github.com/tranek/GASDocumentation

Ability System Component

- Add to any character that (Already implemented by default for character_base)
    - Has Attributes (HP, Mana, etc)
    - Has Abilities
    - Affected by Gameplay Effects

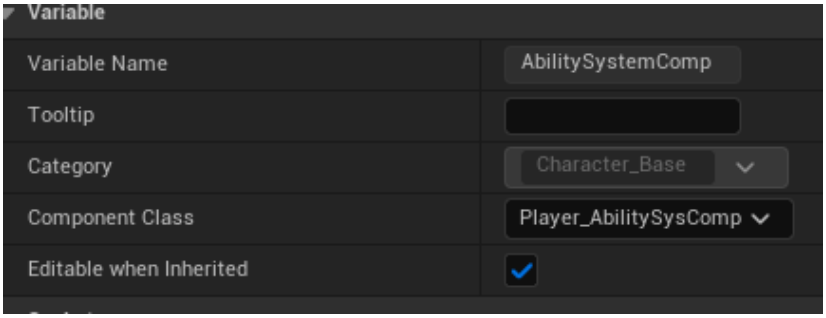
Player Ability System Component

- Derived from the UAbilitySystemComponent
- Input Binding Functionality (Source: Lost)
    *Associates Abilities with Input Actions
        -Trigger Input Action would cast the target ability
    -SetInputBinding
    -GetAbilitiesFromInput
    -ClearInputBinding
    -ClearAbilityBindings



Character_Base

-Derived from ACharacter
-Attributes: Character
    -UAttributeSet_Character
        -Can add additional Attributes universal to all characters
    -UAbilitySystemComponent
        -Component Class can be set in blueprint default



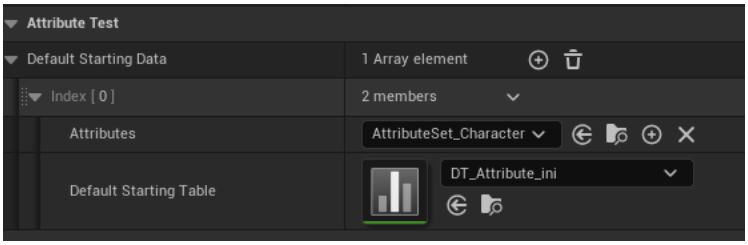
(Set as Ability System Comp for Character_Base)

(Set as Player Ability System Comp for Character_Player)

-AttributeTest_Default Starting Data

-Must Add AttributeSet_Character

-Must have Starting Data from Data Table



-GetAttributeValues (Blueprint Pure)

-OnAttributeValueChangedDelegates (BlueprintAssignable)

-GameEffect Delegates (BlueprintAssignable)

***// Damage is a meta attribute used by the DamageExecution to calculate final damage, which then turns into -Health

Character_Player

-Derived from ACharacter_Base

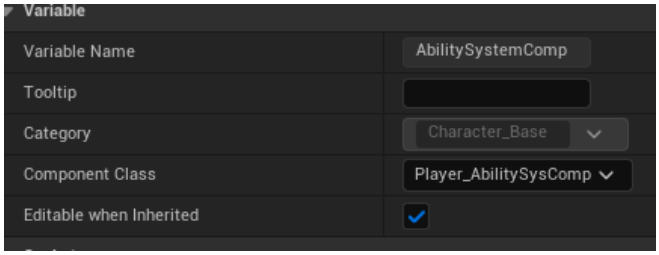
-Has all attributes from Character_Base

-Added additional attributes that only exists on Player Character

-Example: SkillPoint

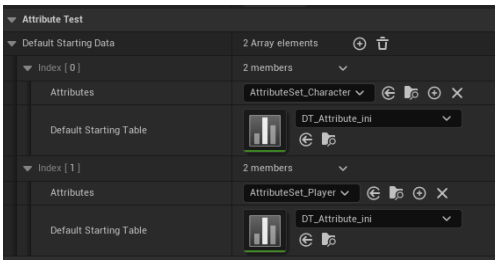
-Other Variables

***Must set the Ability System Comp_Component Class to Player Ability System Comp in Blueprint Default



-AttributeTest_Default Starting Data

-Must Add AttributeSet_Character
-Must have Starting Data from Data Table
-Must Add AttributeSet_Player
-Must have Starting Data from Data Table



Adding New Attributes

1.Declare UPROPERTY in AttributeSet_xxxx.h
    -Clamping in AttributeSet_xxxx.cpp PostGameplayEffectExecute if need
2.Declare listener and accessor in Character_xxxx.h
    -Implement in Character_xxxx.cpp
3.Create Gameplay Tags for the new Attributes in Unreal Project Setting
    -subtags: Add, Multiply, Divide, Override
4.Add Modifier to GE_EquipmentAttribute
    -For Attribute: Add. Multiply, Divide
5.Add Multiply and Divide Tags to “All Multi_Divide Tags” in BPC_Inv_Equipment
6.Add Add Tags to “All Add Tags” in BPC_Inv_Equipment
7.Add to Data table “DA Attribute Ini” or any other data table for initial value

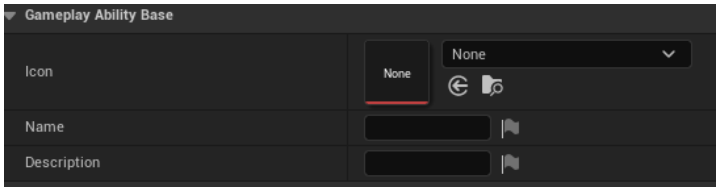
Gameplay Ability_Base

*Use GameplayAbility_Base_BP to create new ability in Blueprint
(default implementation of functions such as TakeUnlockReq, TakeLevelReq)
(default being 1 skill point)
-Derived from UGameplayAbility
-Added Custom Variables (Name, Icon, Description, etc)
-Added Cooldown Duration and Tags (Source: Lost)

Making a New Gameplay Ability

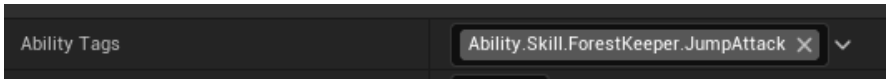
*GameplayAbility_Base_BP instead of GameplayAbility_Base as the former holds default for unlock and level up req functions (Default being -1 skill point). The latter is a c++ class which the base_BP inherits from.

1. Set Icon, Name, Description

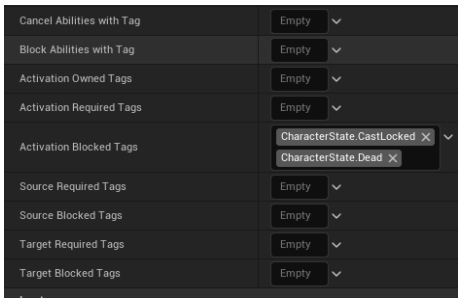


2. Config Class Defaults:

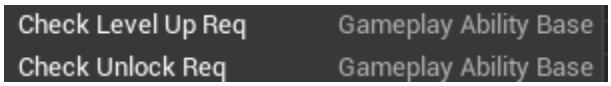
Must Make and Assign Ability Tag

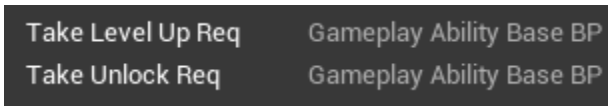


Set Activation/block tags (EX. CastLock)



3. Override Unlock/LevelUp Req Functions (Default being 1 skill point)

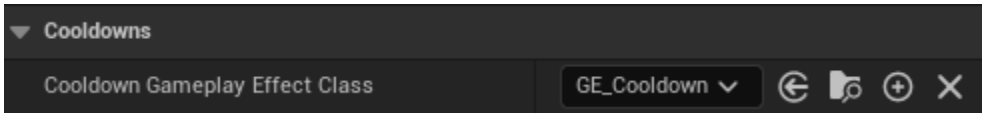




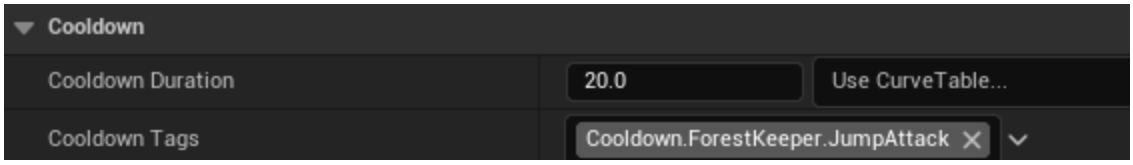
4. Assign Cost Gameplay Effect Class

**This is only 10 SP at the moment, Add Set by Caller later

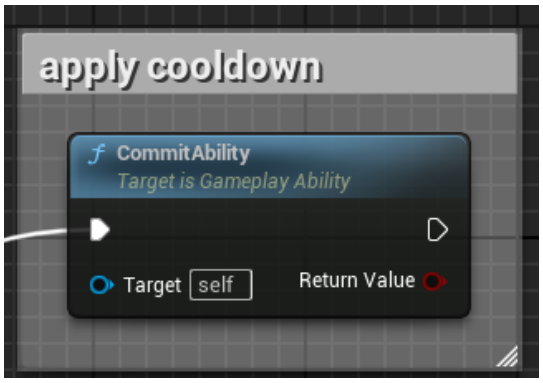
5. Assign Cooldown Gameplay Effect Class



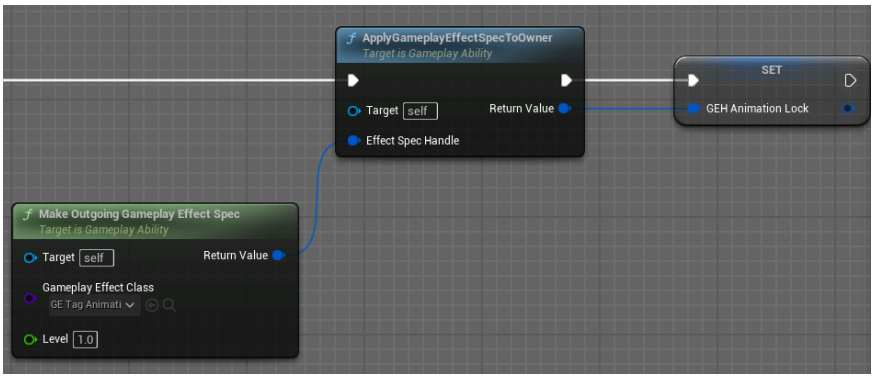
Must Make and Assign Cooldown tag if enabled



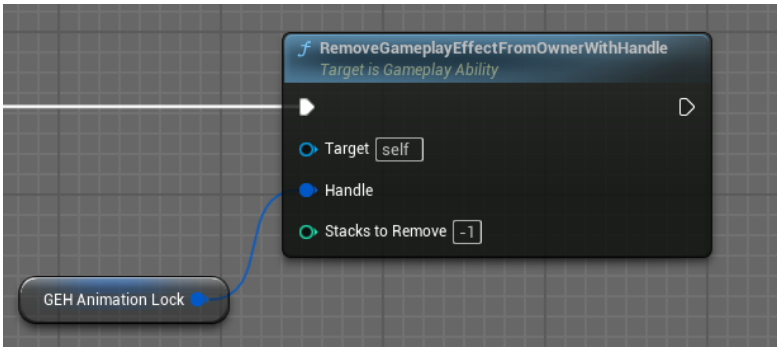
6. Use CommitAbility to Apply Cost and Cooldown



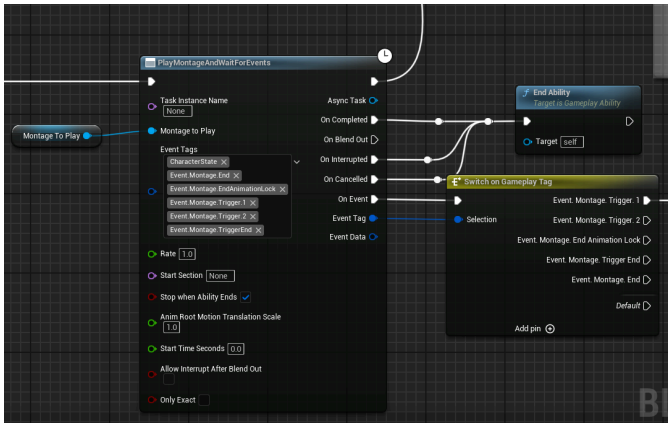
8. Use GameplayEffects that apply Tags to limit movement, double casting, etc.
(EX. Animation Lock, Direction Lock, Movement Lock, CastLock)



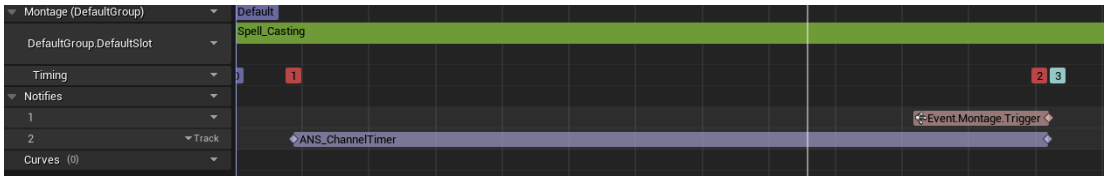
RemoveGameplayEffectFromOwnerWithHandle (remove the tag at ability end)



7. Use PlayMontageAndWaitForEvents to play Animation Montage



-Animation Montage include Notify Event Tag (Trigger)

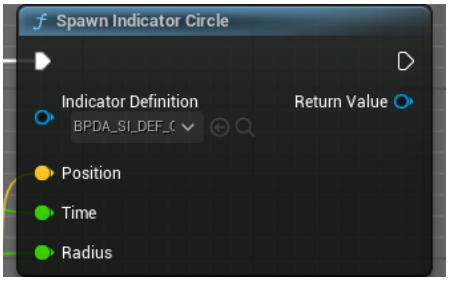


-Notify State_Channel Bar

-calls BP_Hud_Topdown to StartChannel on WB_ChannelBar

8. Ability Indicators (SIZZ Indicator Plugin)

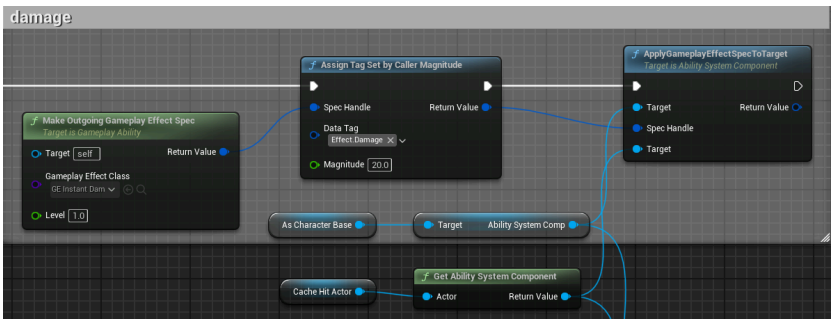
-Spawn Indicator (Circle, Cone, Rect) (Can make custom Indicator Definition).



-WaitforConfirmInput for Player Skills

9. Make Outgoing Gameplay Effect Spec —- Assign Tag Set by Caller Magnitude

Apply game effect and set custom values (Depends on how the target Gameplay Effect is set up)



For more advanced calculation, use custom calculation class

10. End Ability Must be called or the ability will never end or be casted again

Spell Indicator Plugin

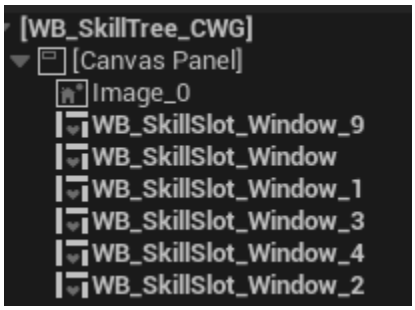
https://sizzonnz.site/spell-indicator

Projectile Abilities

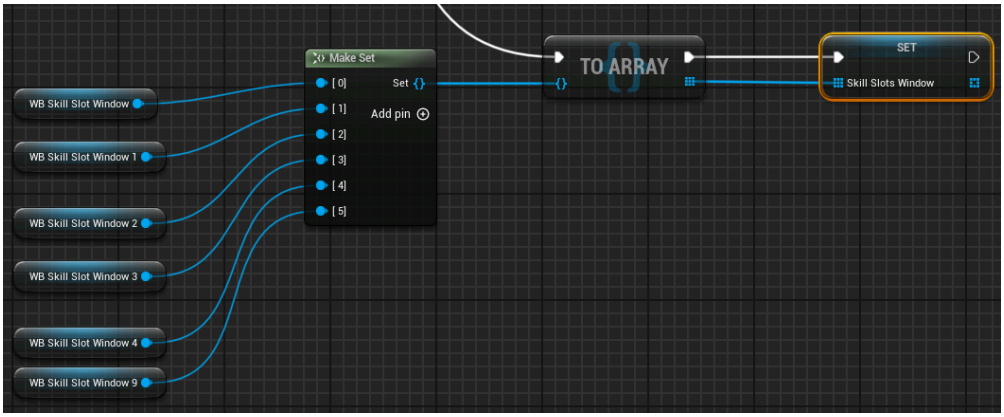
-Pass in the Game Effect Spec to Projectile when Spawned
-Pass in the Caster Object Types (to determine what the projectile is hitting)
-Along with additional Variables

Skill Tree

-Make new Widget Inherit from WB_SkillTree_Base
-Build the skill tree with WB_SkillSlot_Window



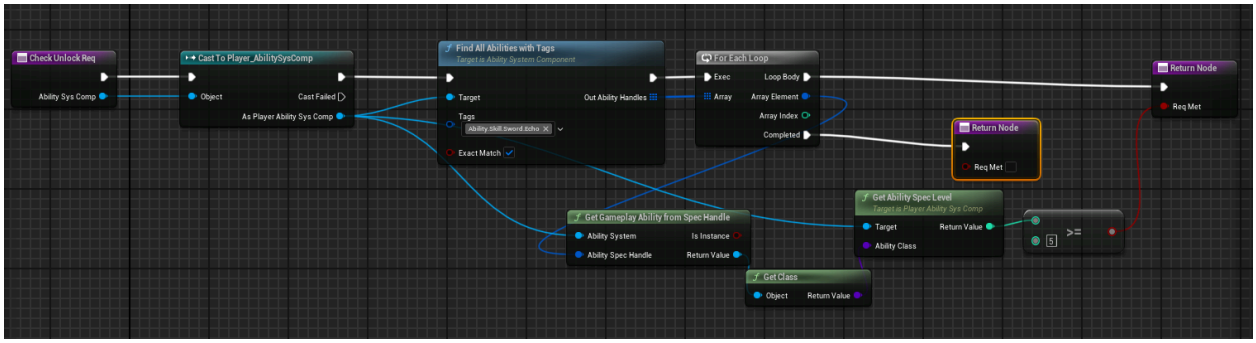
-Add all the skill slot windows to the skill slots window array in EventPreConstruct



-For each WB_SkillSlot_Window assign its GA Skill Class

-For each ability, override CheckUnlockReq CheckLevelUpReq TakeUnlockReq and TakeLevelUpReq to build the skill path

-Use Find All Abilities with Tags and Get Ability Spec Level for Preq skill requirement



-Set the Skill Tree in WB_Menu_Main

*Or latter associate with Individual Characters and make LoadSkillTree function

Gameplay Effect

-Ref Doc https://github.com/tranek/GASDocumentation
-Can also be used to apply Tags to the Ability System Component
    -Use the Tags to enable or disable actions

Gameplay Cue

-Ref Doc https://github.com/tranek/GASDocumentation
-Can be added to Gameplay Effect and trigger while Effect is active

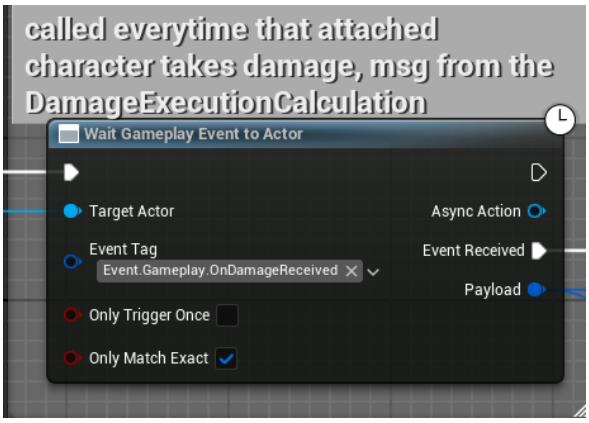
GameplayAbilitySystemUtil (Custom helper function library)

-holds function for blueprint to access variables exist only in c++
-ref GameplayAbilitySystemUtil.cpp

Damage Execution Calculation

Ref Doc
https://dev.epicgames.com/community/snippets/BKP1/unreal-engine-draw-screen-space-damage-numbers-from-world-space-hit-location

-WaitGameplayEventToActor:
    -Event.Gameplay.OnDamageReceived (Called from the DamageExecutionCalculation.cpp)



    -Get the damage data from payload
    -Can be used to spawn Damage Numbers, damage reflect, or anything triggered by taking damage.

-Advanced calculation of Damage such as taking in account of stats can be done in the cpp file.

Articy

Install Articy

-Articy Draft X from https://www.articy.com/en/articydraft/free/

-Articy Unreal Importer from https://github.com/ArticySoftware/ArticyXImporterForUnreal

(Prefer from github as marketplace is often behind in update)

-Make sure project is C++ enabled

-Uncheck "Hot Reload" inside Editor preferences

-Place in project plugin folder and enable

-Generate Project Files

-Add the "ArticyRuntime" dependency (Ref doc)

-Build in IDK (Make sure to checkout from source control)

Dialogue Manager

*Should change it to object class

*Should be created by Gamemode

-Holds the ArticyFlowPlayer

-Creates dialogue box and updates it

-Handles Branching

Ref official doc and youtube tutorial for how its set up

https://github.com/ArticySoftware/ArticyXImporterForUnreal?tab=readme-ov-file#setup

https://www.youtube.com/channel/UCAHixnKHEL9584vWbgyYa0w

*The Menu Text is not displaying correctly at the current version, waiting for fix

Create Dialogue From Articy and Export to Unreal

*Checkout from source control if its in source control

In Articy

-Create Entities

-If player: Add “Main Characters” template

-Would display the player on the left in dialogue box

-Create Dialogue

-Create Dialogue Fragments in Dialogue

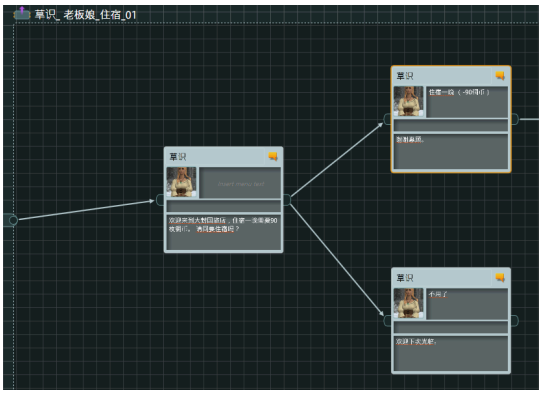
-Set Speaker

-Menu Text (Branching Button Text *bugged)

-Speech

-etc

-Connect the Dialogue Fragments



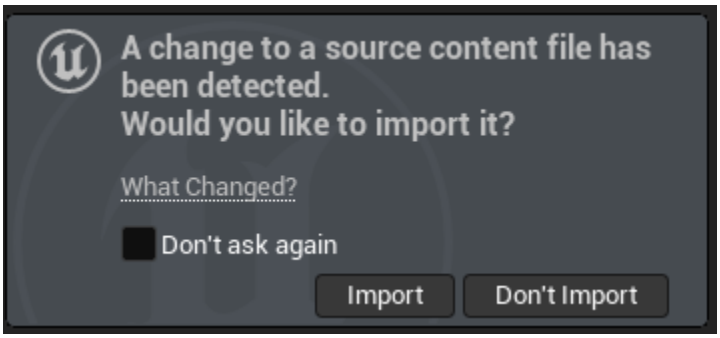
-Export to Unreal Project Content Folder

-Select Package

-Export

In Unreal

-Import changes from text pop up



-Settings → Articy X Importer

-Full Reimport

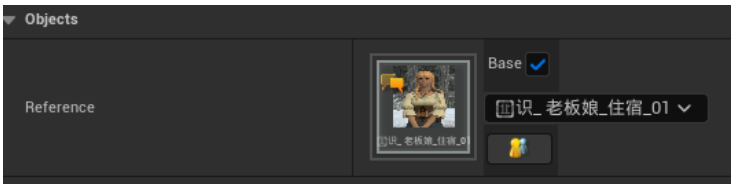
-Import Changes

-Regenerate Assets

-Add ArticyReferenceComponent to Actor

-Or make variable (ArticyRefStructure)

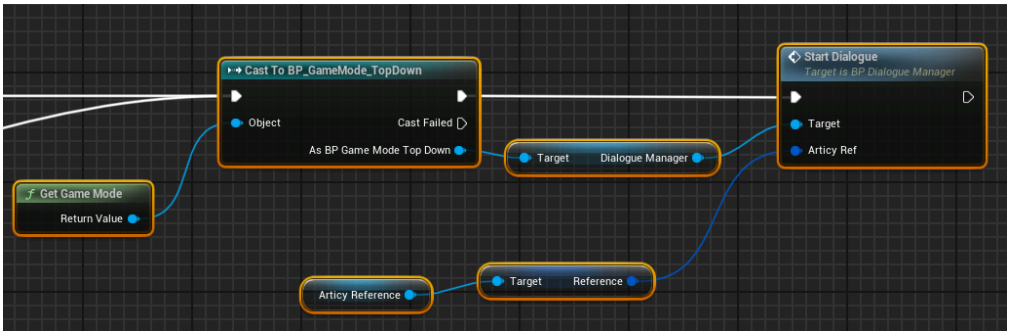
-Set the Reference



-Get Dialogue Manager from GameMode

-Start Dialogue

-pass in the ArticyRef



Fixing Compile Error

*Happens when a new global variable from Articy is added

1. Delete Articy Generated Folder
    a. In xxxProject/Source/xxxProject/ArticyGenerated
2. Generate Visual Studio Project Files
    a. Right Click UEProject file
3. Rebuild project in visual studio
    a. Open xxxproject.sln and rebuild
4. Open Unreal Project
5. Drag in the deleted Articy UE Files
    a. From Content Folder in File Explorer
    b. To Content Folder in UE Content Browser
6. Articy X Importer Window
    a. Full Reimport
    b. Import Changes
    c. Regenerate Assets

Inventory System

BPC_InventorySystem_Base (Component)

-Add to Actor

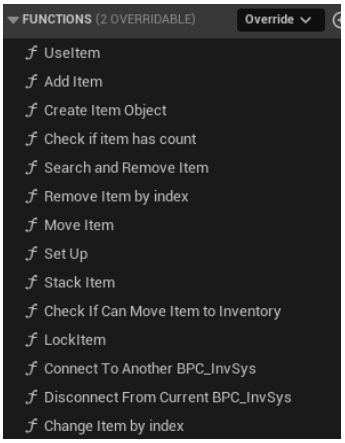
-Call SetUp (Function)

-Sets Inventory Size

***An Actor can hold multiple Inventory Components

-Functions

***Ref Blueprint



Notes

-UseItem *Not Yet Implemented

-Add Item

*Need to add a check (Or function to see enough space before commitment)

-Remove Item By Index *Not Yet Implemented

BPC_Inv_Player

-Same as BPC_InventorySystem_Base

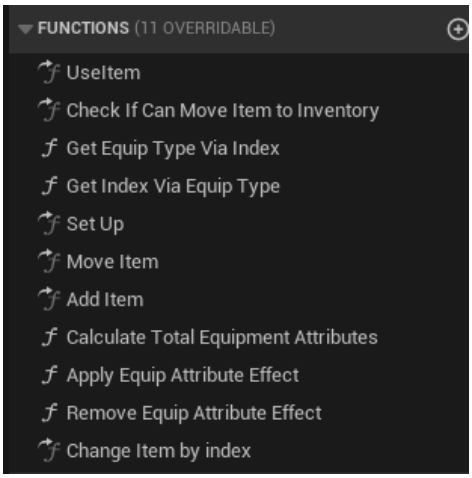
-Used For Get Component by class

-Get BPC_InventorySystem_Base would target both Inv_Equip and Inv_Base

BPC_Inv_Equipment

-Overrides some Functions in BPC_InventorySystem_Base

***Ref Blueprint



-Holds/Equips one equipment for each type

-And one deco equipment for each type

-Each Equipment type has an index reference

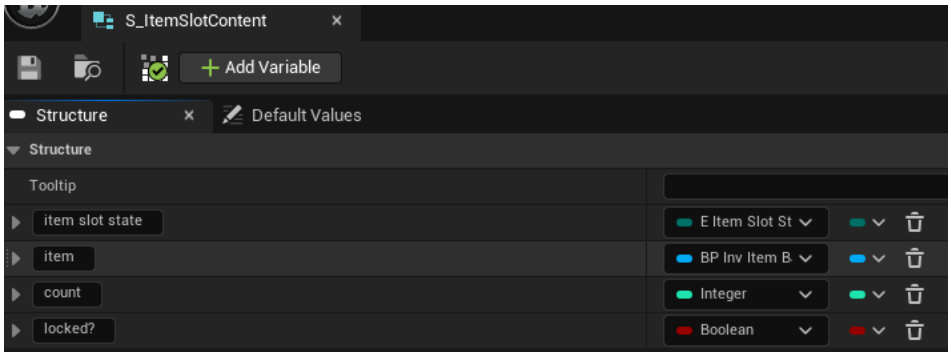
-Has Equipment Attributes

-All_Multi-Divide_Tags and All_Add_Tags

-Used to avoid Set by Caller Error When GE_EquipAttribute is applied

-Caused by Equips not covering all Attributes

ItemInventory



-S Item Slot Content Struct

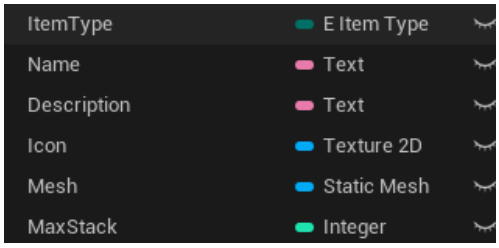
-Where items are stored

-E Item Slot State

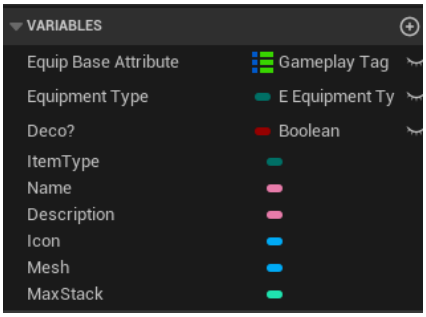
-Item (BP Inv Item)

-count
-Locked?

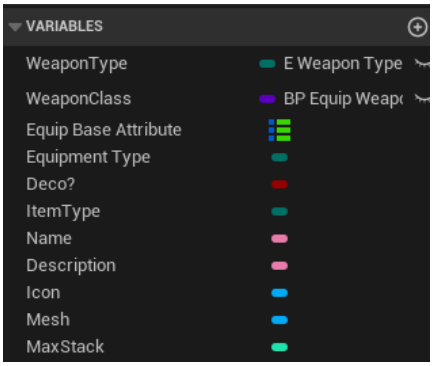
BP_Inv_Item_Base
-Holds PDA_Item
-Basic Item Info



BP_Inv_Item_Consumable
*Not Yet Implemented
BP_Inv_Item_Equipment
-Equipment Attribute
-Holds PDA_Item_Equipment
-Cast PDA_Item to PDA_Item_Equipment
-Equipment Attribute
-Equipment Type
-Deco? (Affect Appearance)



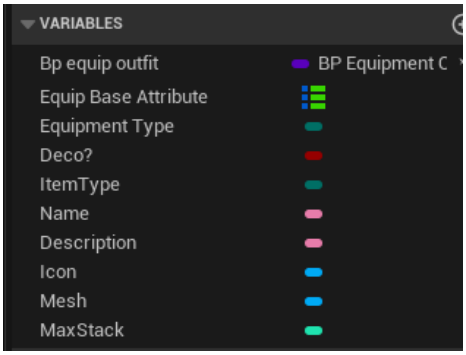
-Cast PDA_Item to PDA_Item_Equipment_Weapon
-Weapon Type
-Weapon Class
-Mesh
-Basic Attack Ability
-Attachment



-Cast PDA_Item to PDA_Item_Equipment_OutFit

-BP_Equipment_Outfit Class

-Changes Skeletal Mesh *Not Yet Implemented



Add New Item Type

1. Add to E_ItemType
2. Make new PDA_Item
    a. Add additional info it needs to hold
3. Make new BP_Inv_Item
    a. Add additional info it needs to hold
4. BPC_InventorySystem_Base
    a. CreateItemObject (Function)
        i. How the item should be created (Construct BP_Inv_Item object with PDA_Item)

Make New Item

-Make new DA_Item with corresponding Item Type

Collision

-Hitbox (Used to get Hit by line, sphere, box traces)

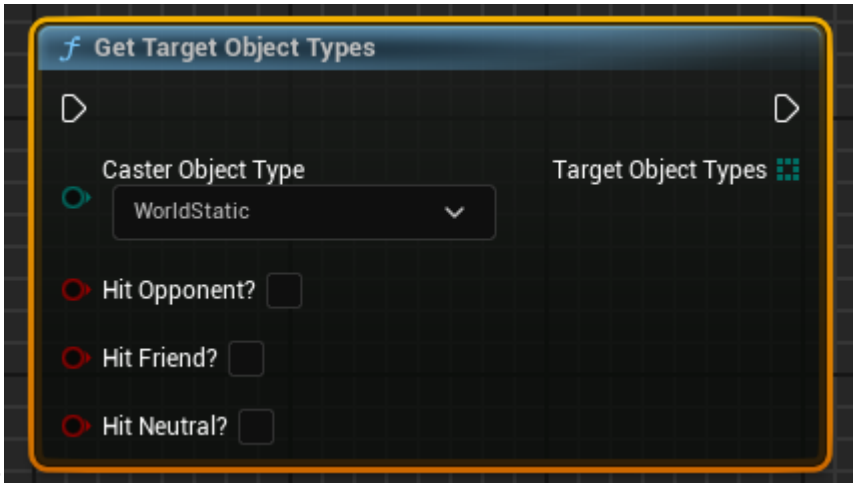
-AllyHitBox

-EnemyHitBox

-NeutralHitBox

-BPFL_Combat → Get Target Object Types

-Returns what the traces should hit based on self object type



-

-GroundedEffect

-For effects that is affected by gravity and collides with static

-Interact

-Works with TraceMouseInteract

-Interact overlap collision

-TraceChannel

-TraceMousePad

-An invisible plane is created that follows player character

-used to control spell cast direction/location

-TraceMouseInteract

-Trace to hit objects that can interact with

-TraceMouseLocation

-Used to work with ClickMove

-trace and return location player can move to via navigation

Animation Retargeting: Mixamo to Blender to Unreal

***requires unreal 5.4+, but can export and import to 5.3 etc.

Set up

https://www.mixamo.com/#/

1.Download a Rig from Maximo

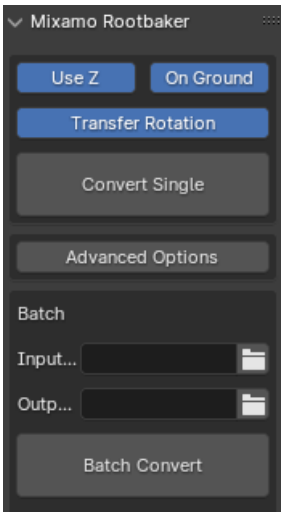
2.Download mixamo rootbaker plugin and install on blender

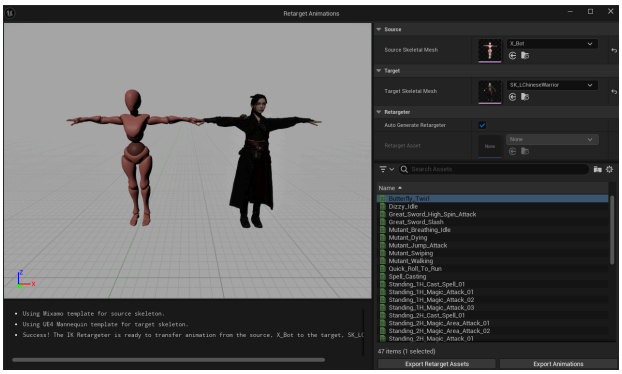
https://github.com/enziop/mixamo_converter

3. Convert/Add rootbone to the Rig from Maximo

4. Import the Rig to Unreal 5.4+

Animation

1. Download Animation from Mixamo (No Skin)
    a. Some Animation has stay in place checkbox
2. Convert using the Mixamo Rootbaker in Blender
    a. 

3. Import the Animation to Unreal 5.4+
    a. Select the Mixamo Rig as Skeletal Mesh
4. Import in your custom Rig
5. Right click on the Mixamo Rig Animation and select animation targeting
    a. Set your custom Rig as target rig
    b. 
6. Export Animations that have been retargeted to Unreal 5.3 etc.