Overview

Ability system is the framework for creating Attributes, Effects and Abilities that are often used to implement combat systems, or any other interaction between actors in game world.
Attributes are Health, Shield, Attack, Critical Chance, Movement Speed etc. They can be added to Player Character, Pawn or any static actor like Tower.
Effects can be infinite or temporary, can be simple or complex. They can modify attributes, apply other effects, abilities or perform more complex logic.
Abilities are actions that can be added to any actor and mainly created to player characters and pawns. Abilities can be passive, for example, Vampirism, Reflection.

Additionally, a Level System is implemented for Player.

System works with **GameplayTags** that helps to identify the target, his current state. For example, if the target can be attackable, interactable, or has control effects.

System works only in **single player** games. All ability system progress is **saved**.

Let's look at **project structure**.
I have one level and two game modes. Third person and top down. Each uses its own Player Character and Player Controller classes.

Logic for player is implemented in BP_PlayerCharacter. BP_Character is parent class.
Logic for AI is implemented in BP_AI. BP_Character is parent class.

Equipment Manager and **Weapon** actor are created specially to test Ability System with weapons. So when player equips weapon, gameplay tags can be added. Switching weapon can be blocked if owner has any control effect.

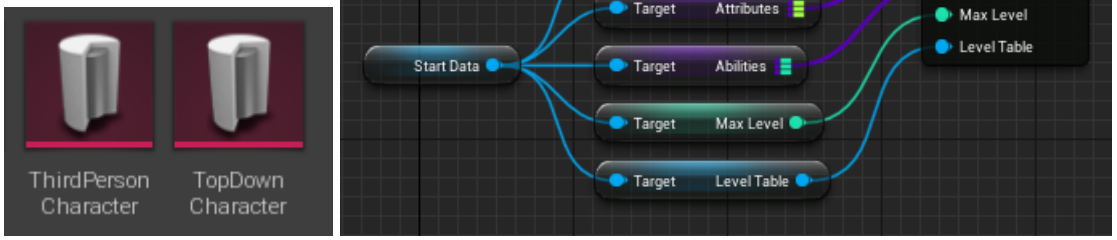
Some project settings have been changed. Gameplay Tags dictionary is created. Some inputs are added. EQS is enabled in Editor Preferences.

Ability Manager

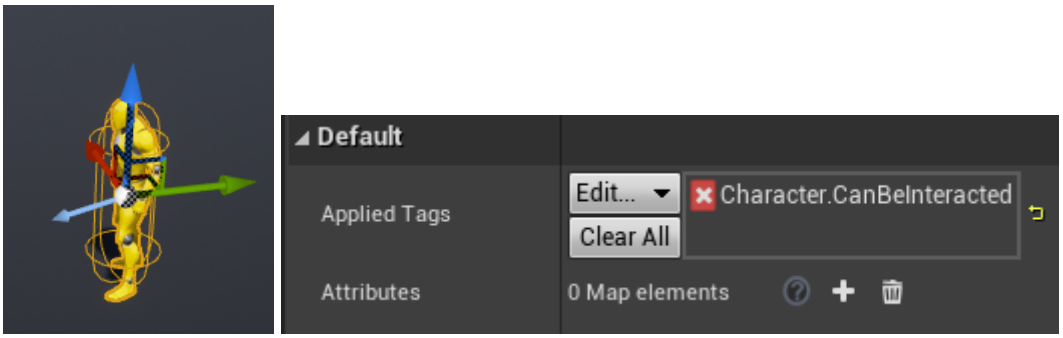
Ability Manager is connected to any actor you want.
Manager stores all ability system data: attributes, effects, abilities, gameplay tags.
It modifies attributes, activates and deactivates abilities, applies and removes effects, handles Level System.

Not in all cases you need to connect the manager to the actor. For example, any area in the world that deals fixed damage may not have a manager.
But if you want the actors to have attributes, abilities, you need to apply effect to actor, or you want the actors to interact with each other according to the ability system rules, the manager should be added to them.

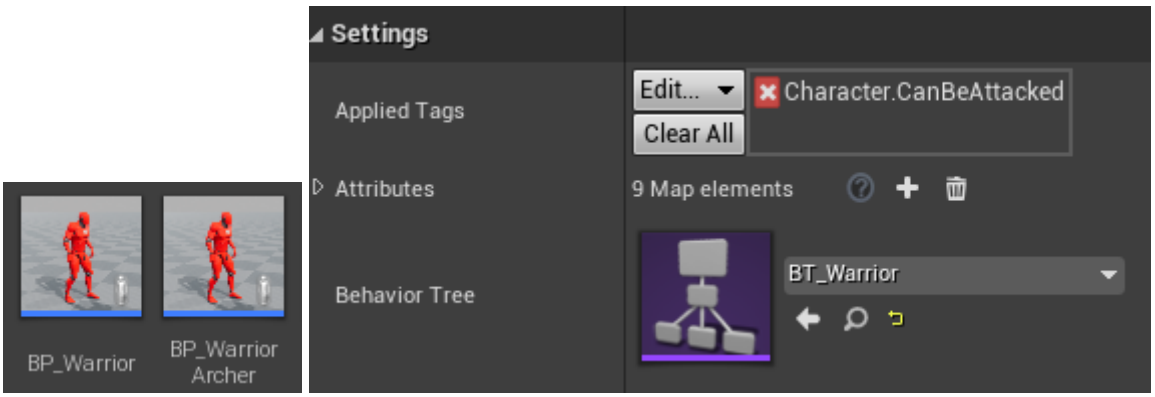
Manager need to understand what attributes and abilities should be created, what tags should be applied to the owner at start game.
Manager gets start data using **GetStartData** interface function.
It's not important how you store this data, but it's necessary to give this data to Manager.
For player I use special data assets with additional info like MaxLevel and LevelTable



You can select yellow AI on the scene and check details panel options. Yellow AI is just interactable.



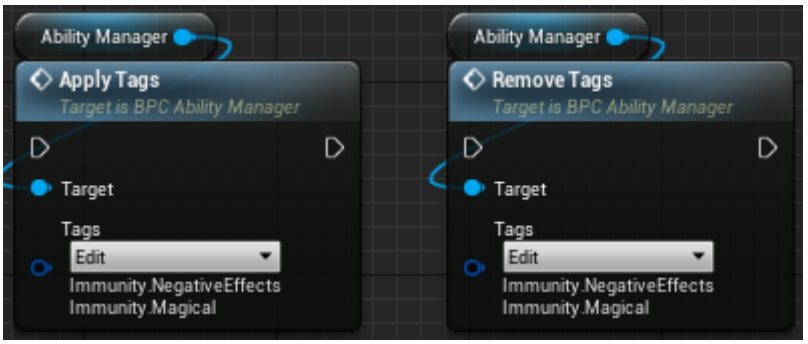
Since monsters are spawned at runtime, you can open **BP_Warrior** or **BP_WarriorArcher** and check settings.



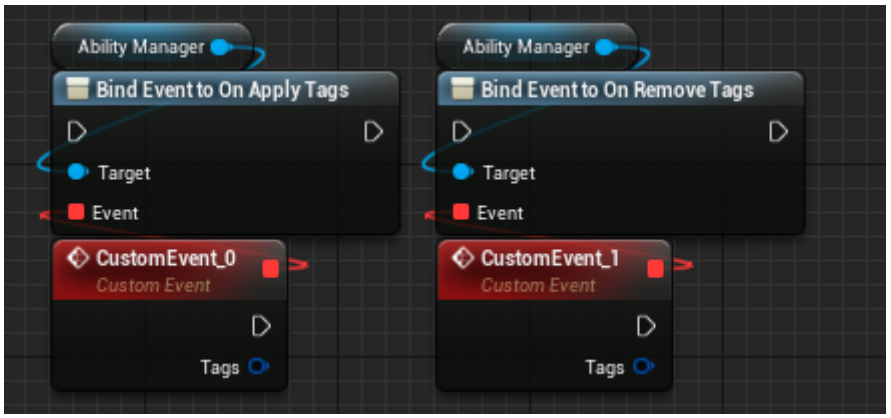
Gameplay Tags

[Epic's documentation]

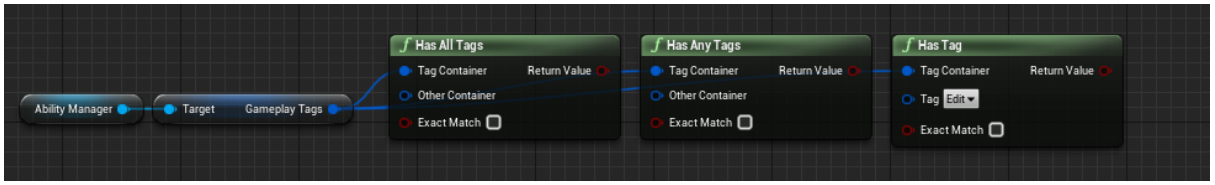
You can apply or remove tags using two functions.



And bind to the appropriate events.

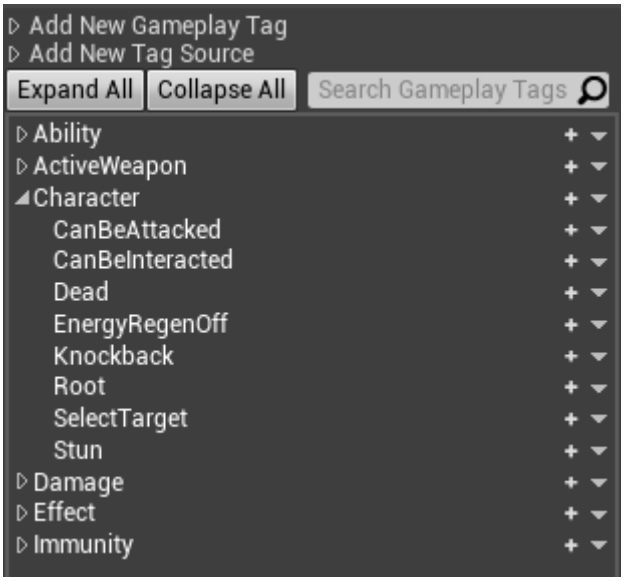


I didn't writing any special functions for getting info about actor tags, so you can use the Epic's library



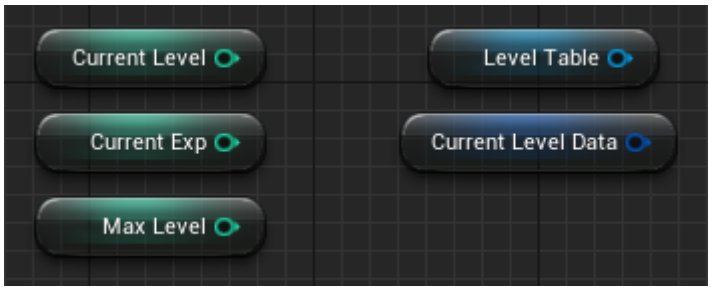
Next tags are already used in project.

Open Edit -> Project Settings -> Gameplay Tags to check them

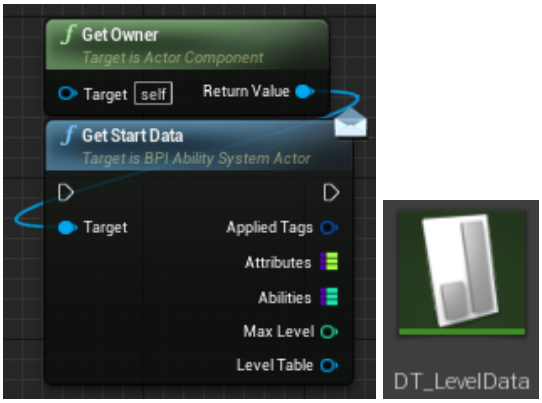


Level System

There are a few variables in Ability Manager.

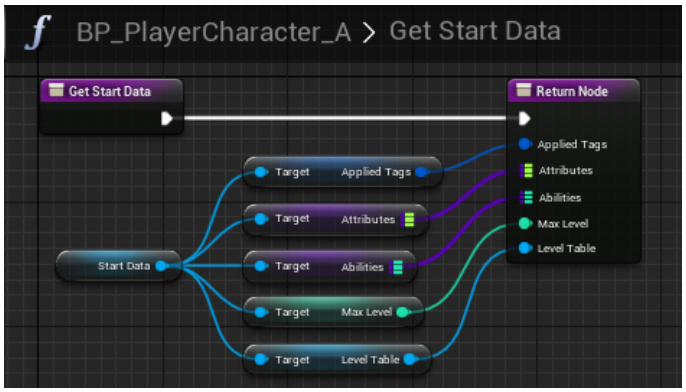


By default, **CurrentLevel** = 1.
MaxLevel, **ExpToLevelUp** and **Reward** can be different for actors, so manager uses interface to get the appropriate data from the owner.



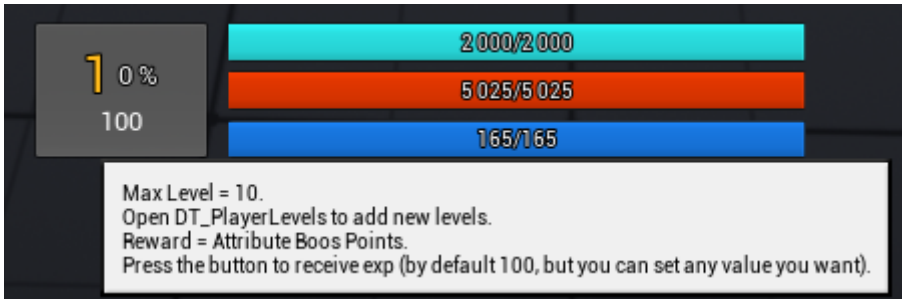
	Ro	BoostPoints	ExperienceToLevelUp
1	1	0.000000	100
2	2	4.000000	200
3	3	4.000000	300
4	4	4.000000	400
5	5	4.000000	500
6	6	4.000000	600
7	7	4.000000	700
8	8	4.000000	800
9	9	4.000000	900
10	10	4.000000	1000

The interface is implemented in Player Character class.
I use data assets to store all data for player.



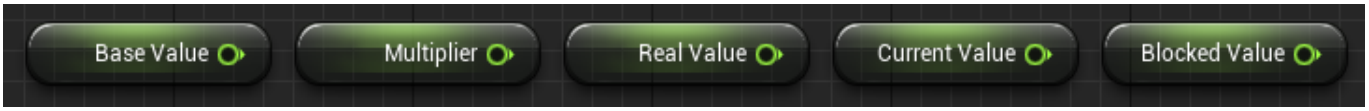
Use **AddExp** function to add exp points.

In the demo, you can use a special WBP_LevelBar to add exp points. Just press button.



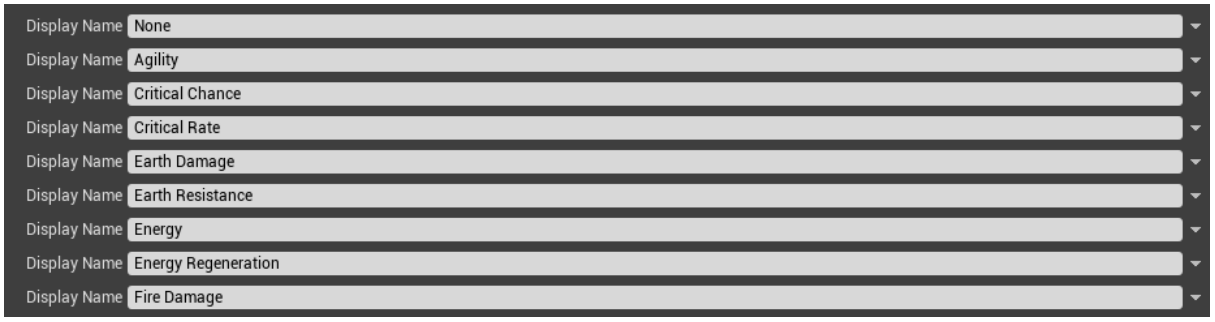
Attributes

An attribute can be Health, Defense, Attack, Critical Chance, Movement Speed, Resistance etc.
In fact, attribute doesn't have one value, but several values, so attribute exists in memory as **BP_Attribute** class.



For example, with **Multiplier** we can increase/decrease attribute by %.
With **CurrentValue** we can implement Health, Mana, Shield.
BlockedValue is created for specific cases. It can be an effect that increases attack, but blocks 75% Health. It makes sense to use BlockedValue only for attributes with CurrentValue. When you modify blocked value, CurrentValue is clamped from **MinValue** to **'RealValue - RealValue*BlockedValue'**. When you modify BlockedValue, use Add or Subtract. Keep in mind you work with percent (from 0 to 1).

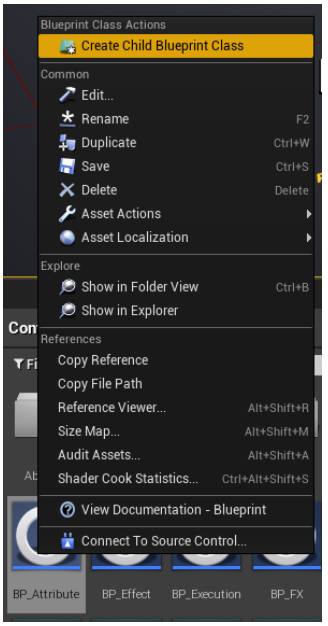
All attribute names are stored in **E_Attribute** enum. The same names are used as keys for searching and working with attributes. At any moment you can add a new attribute name to this enumerator (advice: it is more convenient to work with alphabetical order).



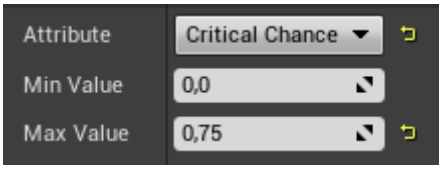
As I said earlier, all logic is in **BP_Attribute**. This is a template, parent class.

Use **Create Child Blueprint Class** option to create new attribute.

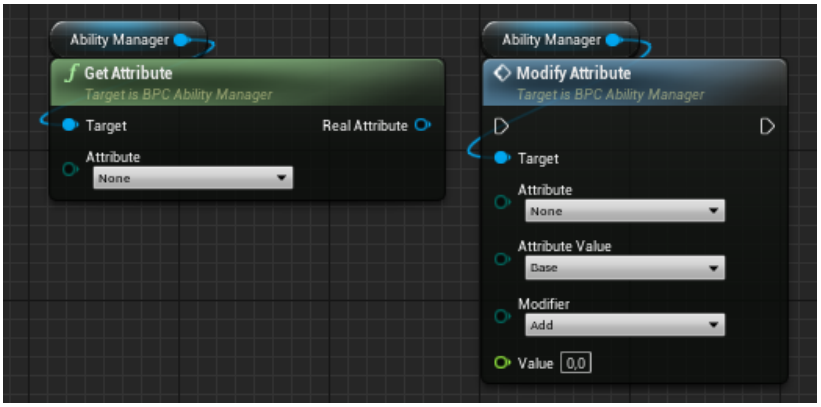
After creating you can open it and set parameters.



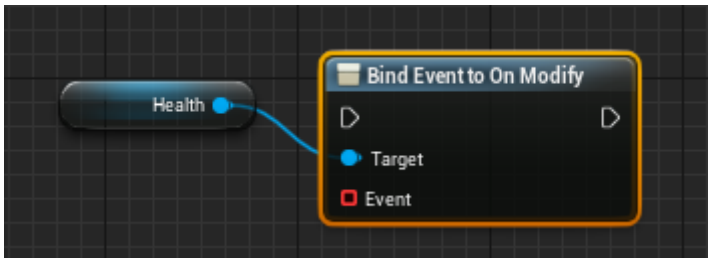
Attribute parameter is used as the key for searching and working with attributes. For example, in your game Player and AI can have Critical Chance. MaxValue for Player - 75%, for AI - 50%. In this case you create two attributes with the same key, but different settings.



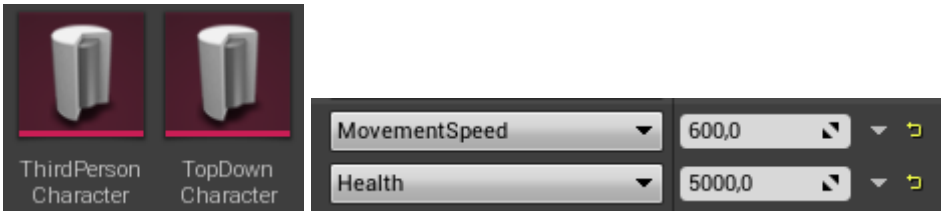
You can use the next function



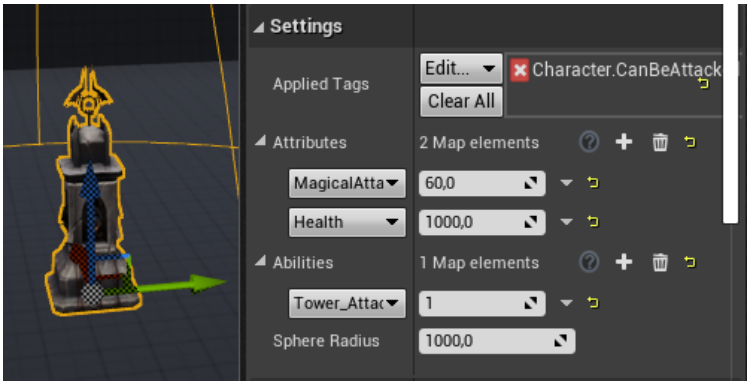
After you get reference to attribute, You can bind to a **modify** event, using dispatcher. It is useful to handle regeneration etc



Once you have created attribute, you need to assign it to an actor.
For **Player**, data is set in data asset. Here you can set attribute and its initial value.



For **Tower**, data is set in Details Panel when you place actor on the scene.



For **AI**, data is set in Details Panel (in Ability System project).

In AITools and RPGTools data is set in data asset.

Effects

Main

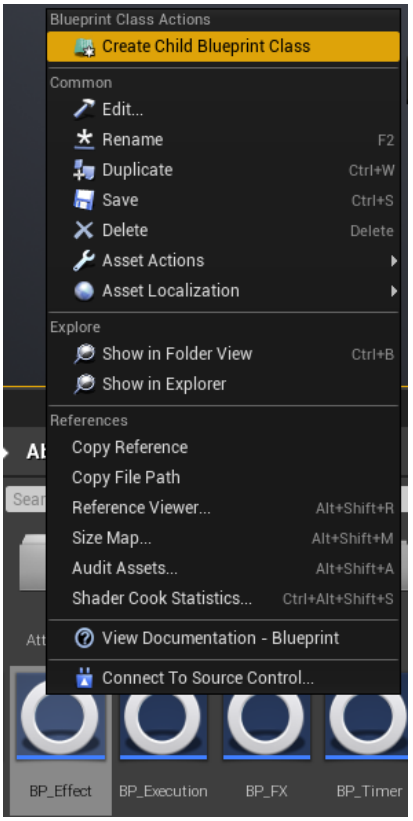
Effects can be infinite or temporary, can be simple or complex. They can modify attributes, apply other effects and abilities, or perform more complex logic.

All logic is implemented in **BP_Effect**. This is a template, parent class.

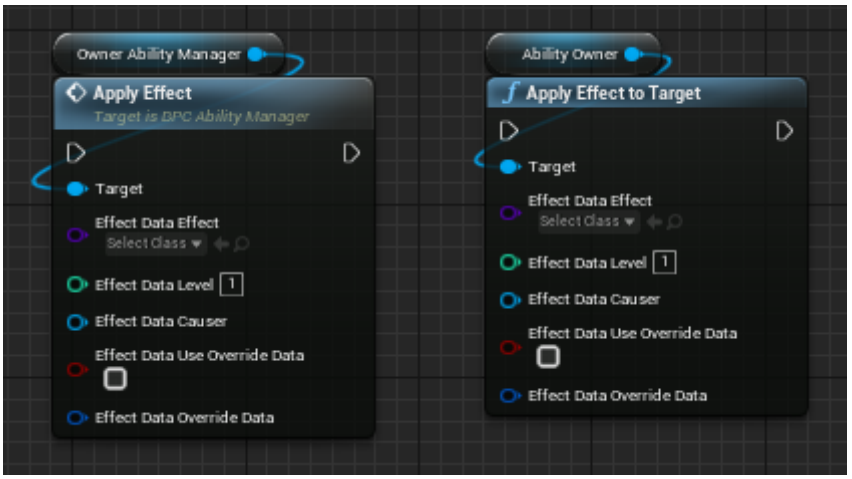
Use **Create Child Blueprint Class** option to create a new effect.

Out of the box effect is very powerful so you don't need to write any logic, just open it and set parameters.

I created videos on Youtube, where I talk about how to use these parameters.

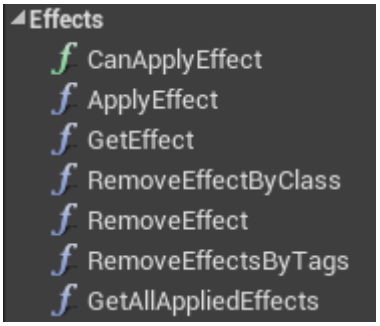


If you want to apply an effect to actor, the actor should have Ability Manager.



*If you haven't reference to manager, use library function **ApplyEffectToTarget***

You can remove the effect in a few ways (by class, by object, by tags), get info about effects.



Override Effect Data

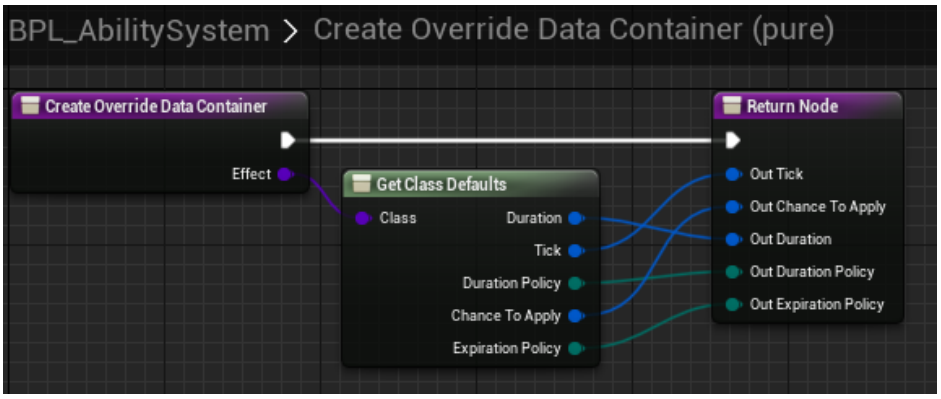
This functionality is useful when you want to use one effect class with different values.
In the demo I use **ImmunityNegativeEffects** as an infinite effect for the player, and temporary effect (15 sec) for monsters.
Also look at the **MovementPower** effect in WBP_TempEffectsWindow. I use one class with different duration settings.

I created a special structure **S_EffectOverrideData**. This structure contains all parameters that can be overridden.

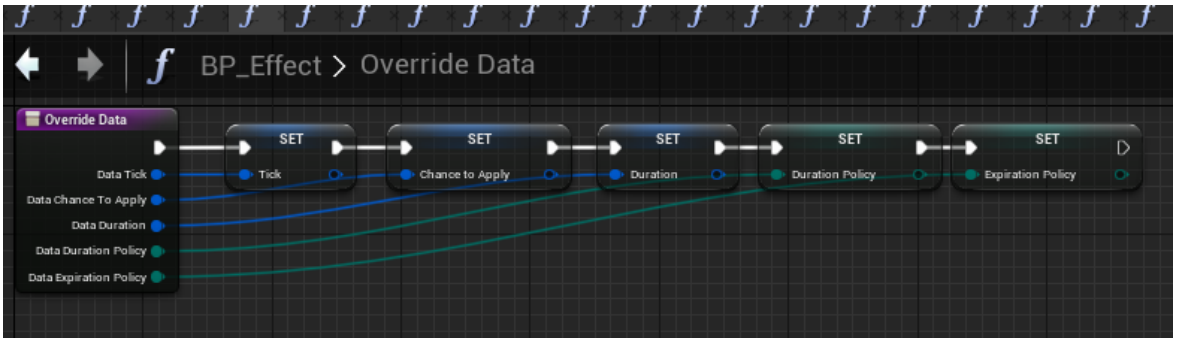
Adding new parameter to structure

1. Open structure and add new parameter

2. Open BPL_AbilitySystem -> CreateOverrideDataContainer and connect new added parameter from class defaults with return node output

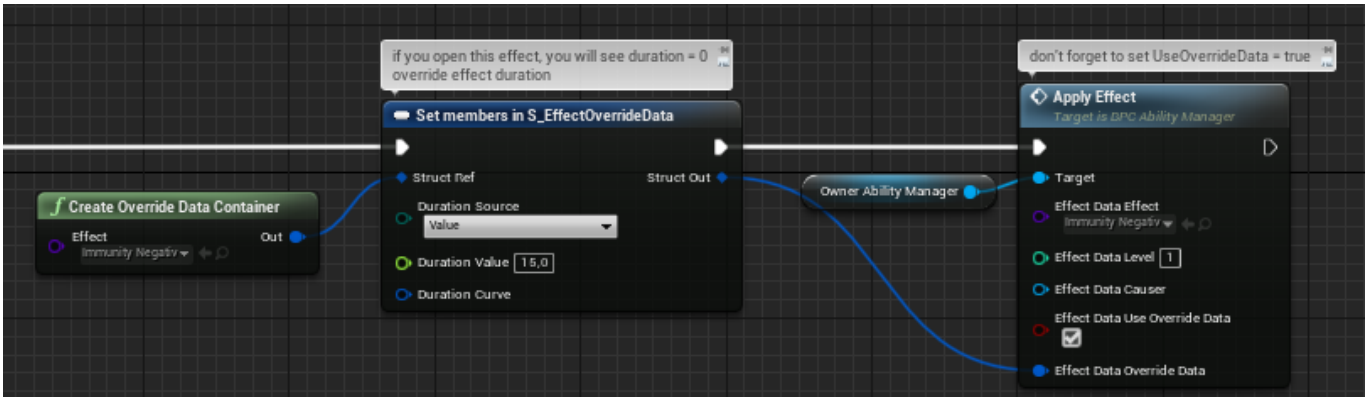


3. Open BP_Effect -> OverrideData and connect new parameter with already existing effect's parameter



Override via blueprint

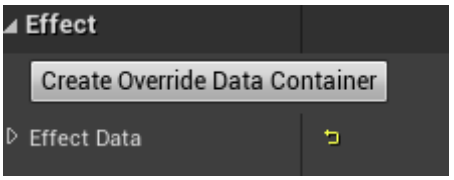
Open **AI_MonsterPower** ability and look at this logic. In all cases you need to use this logic if you want to override effect's data via blueprint.
In this case we override duration. By default, this effect is infinite.



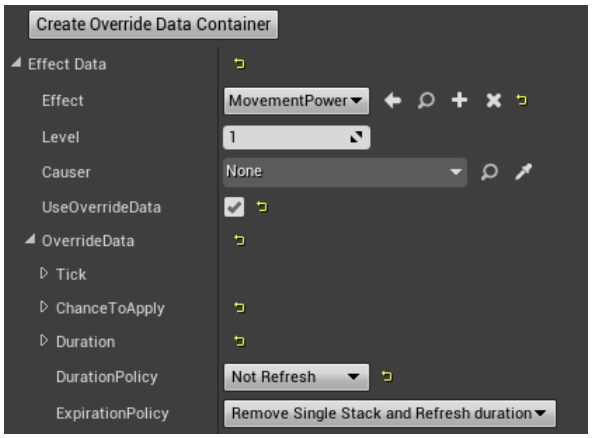
Override via GUI (used in demo)

Open WBP_TempEffectsWindow. Let's take a MovementPower.

1. Select effect in GUI
2. In details panel find 'Effect' section and press **CreateOverrideDataContainer** button



3. After data is created you can change some of these

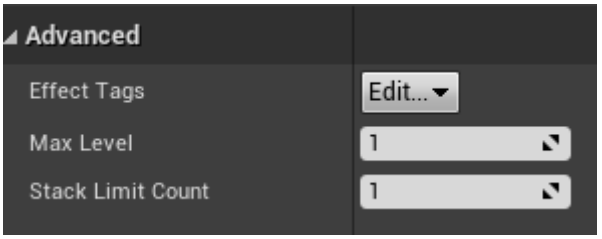


4. Enable **UseOverrideData** option

!!! In the GUI case, think about that when you change default values in effect class, you need to open the GUI and recreate the override data container and override the needed parameter again.
!!! It is preferable to override only those parameters that are automatically generated in tooltip description for effect (like duration, chance), otherwise, old values will be displayed. Modifiers are not generated in tooltip, they are typed manually in description.

Leveling

If you want to add levels to effect, change MaxLevel parameter



Level of applied effect will be clamped from 1 to MaxLevel.

If the same effect is applied, but level is higher, then the old effect is removed, the new one is applied.
If the same effect is applied, but level is lower, then the duration of the old effect is updated, if DurationPolicy = Refresh.

If an effect_1 is applied by effect_2, then levels are the same.
Execution and FX levels are equal to effect's level.

In **Working with float value** section I explain how to work with float values like duration, tick, damage etc and why the curve table is useful for leveling.

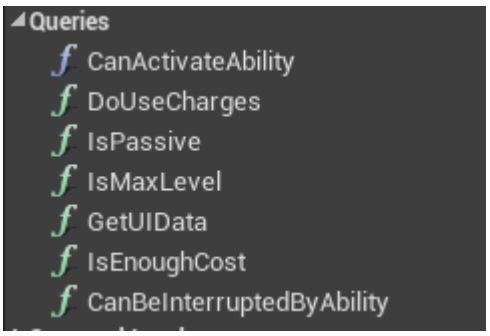
Abilities

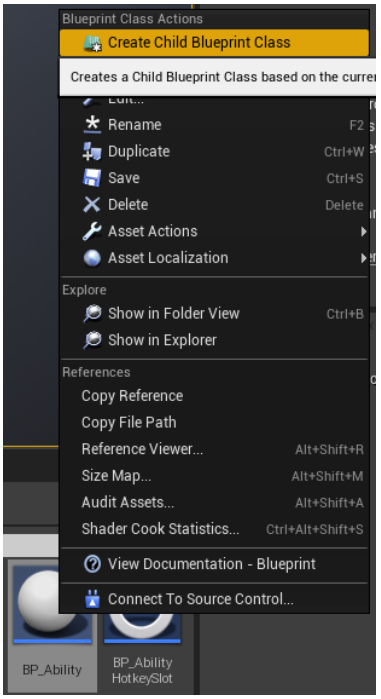
Main

All logic is implemented in **BP_Ability**. This is a template, parent class.

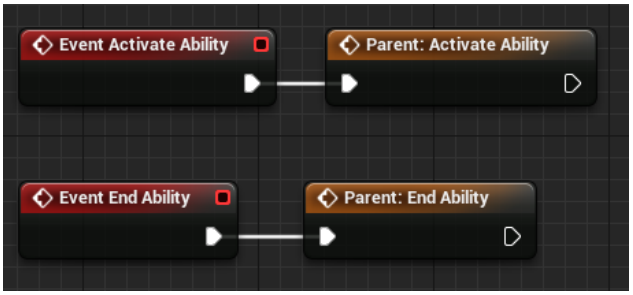
Use **Create Child Blueprint Class** option to create a new ability.

The parent class already contains some logic handling activation, cost, cooldown, charges, leveling etc, and also contains some useful queries

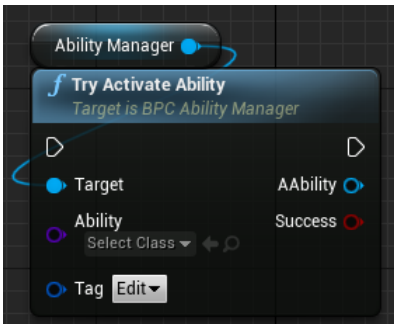
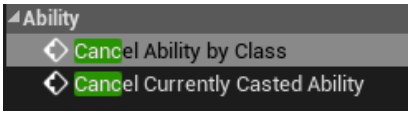
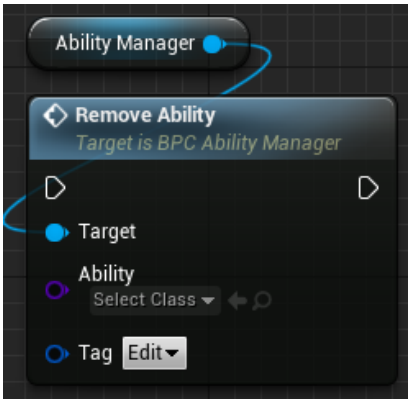




There are 3 core events in ability class: ActivateAbility, CancelAbility, EndAbility.
If you open BP_Ability, you will see that some logic is written for these events.
So when you create a new ability you need to override these events.
Look at abilities in the demo. For all them I override these events.

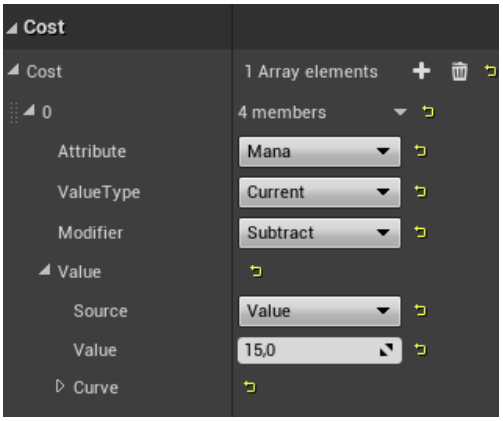


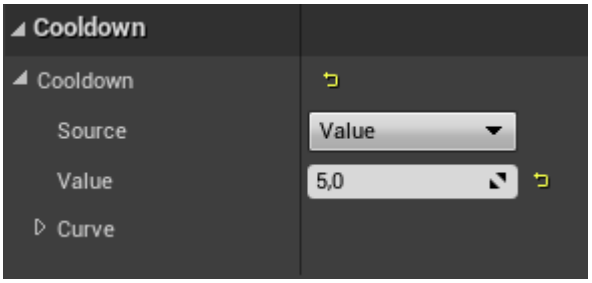
Ability and Manager

Used to give ability to owner	
Used to activate ability If Ability class ISN'T valid, ability tag will be used. Inside the function manager performs some checks, and if the result is valid, ability is activated. If the owner doesn't have the ability, it is not casted.	
You can get info about any owned ability	
Cancel ability	
If ability is casted, it will be canceled automatically and then removed	

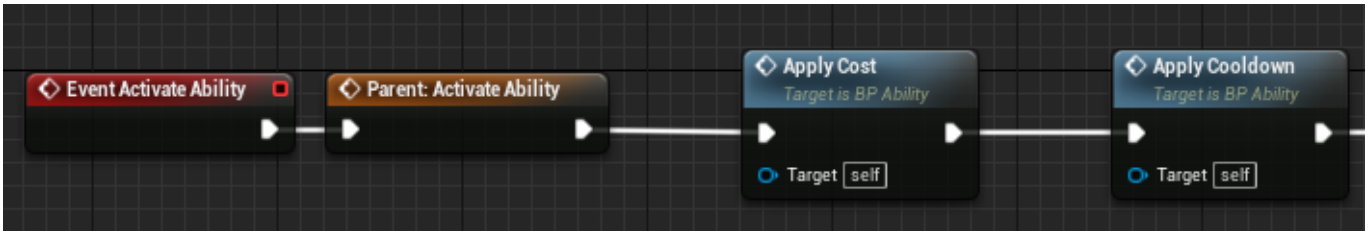
Cost and Cooldown

Set **Cost** (what attributes should be modified) and **Cooldown** (time, after that ability can be casted again) in the details panel.

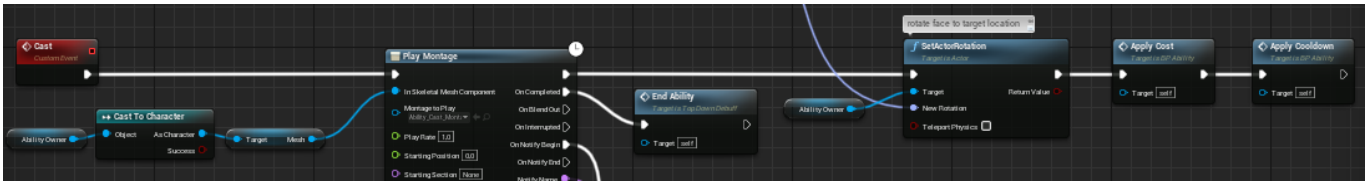




Cost and Cooldown AREN'T applied automatically. You need to call these functions manually.
Look at **Overall_FullRecovery** and **TopDown_Debuff.**
For **FullRecovery** I apply them at once after ability is activated.



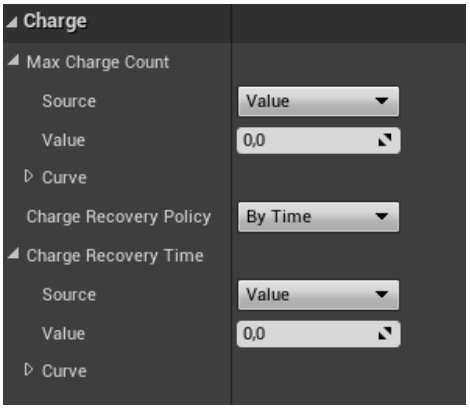
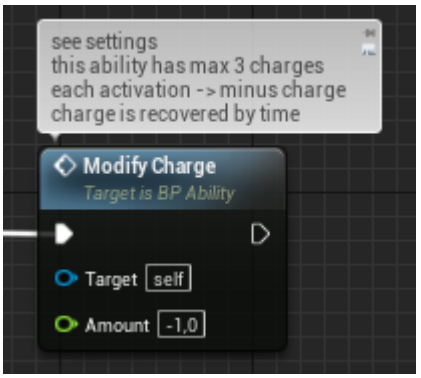
For **Debuff** I apply them only when the player begins cast animation, because before this he should move to location at acceptance distance, and while it moves ability can be canceled.



For **Fast Movement** ability I don't use these functions, because this ability hasn't cost and cooldown.

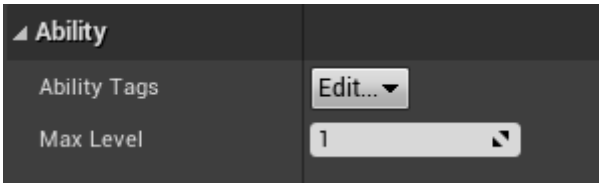
Charges

Ability can have charges, which means that ability can be used a limited number of times.
Charges can be recovered by time or by other external actions.

Set these options id details panel to determine charge policy	
You need to call this function manually when you create an ability with charges. See my **TD_LightingChain** or **TP_Archer_SuperArrow** abilities.	
If charge is recovered by time, the timer is activated automatically. Don't worry about this.	
If charge is recovered by other external actions, use this function from the ability manager. It is useful when you want to recover charge by overlapping any area, or using any item.	

Leveling

If you want to add levels to ability, change MaxLevel parameter



If an ability is applied by effect, then effect ability = effect level.
When level up ability is not canceled. If ability is in cooldown, cooldown time is not updated.
To level-up ability or reset level, you need to call functions from Ability Manager

In **Working with float value** section I am explaining how to work with float values like cooldown, cost, damage etc and why the curve table is useful for leveling.

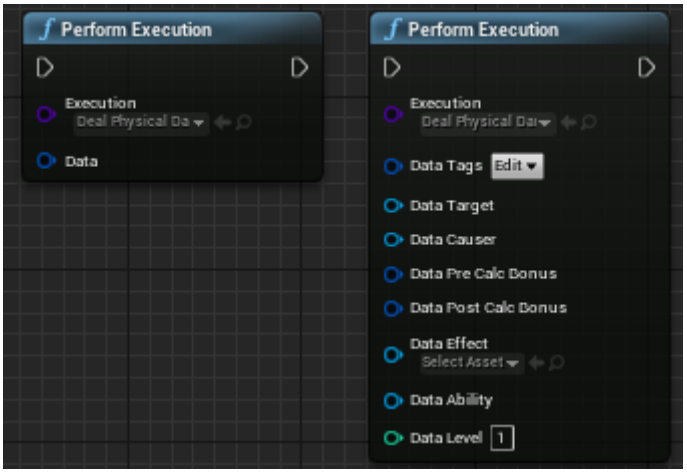
Execution

BP_Execution class is used to perform more complex logic when we apply an effect.
For example, an effect that deals damage. For calculation of damage we need to use the formula.
I store all executions in a separate folder.

When you create an effect, you can set execution in **Executions** parameter

***Heat** effect deals 100 damage each 1 second*

You can call execution from any place in your project you want. Just call library function **PerformExecution**

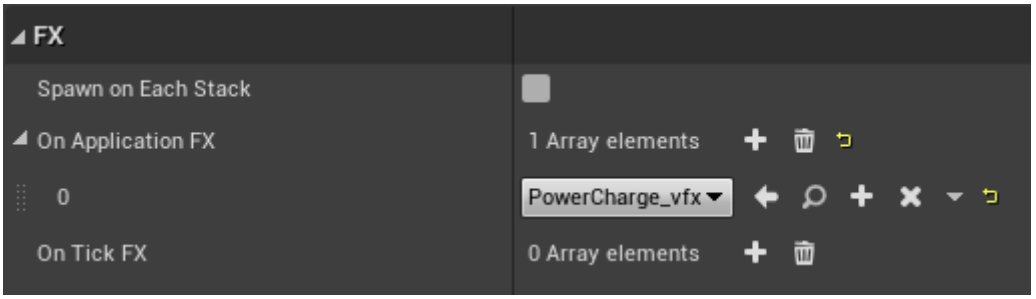


FX

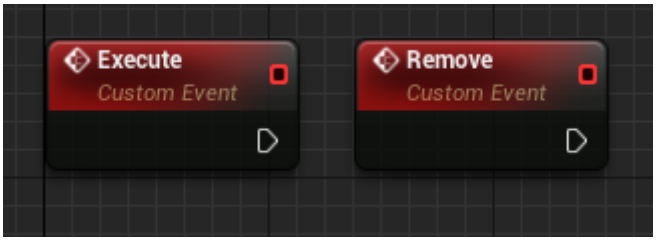
BP_FX class is used only to spawn visual effects for BP_Effect class (vfx, sounds, animations, ui effects etc).
I store all FX in a separate folder.

In fact, it makes sense to use this class only for effects.
If you need to play animation or spawn VFX when you cast ability or projectile hits any objects, you can do it without this class.

If you want to show ‘aura’ or play sound for tickable effect or play animation when the target is stunned, it makes sense to use this class and set the following settings.



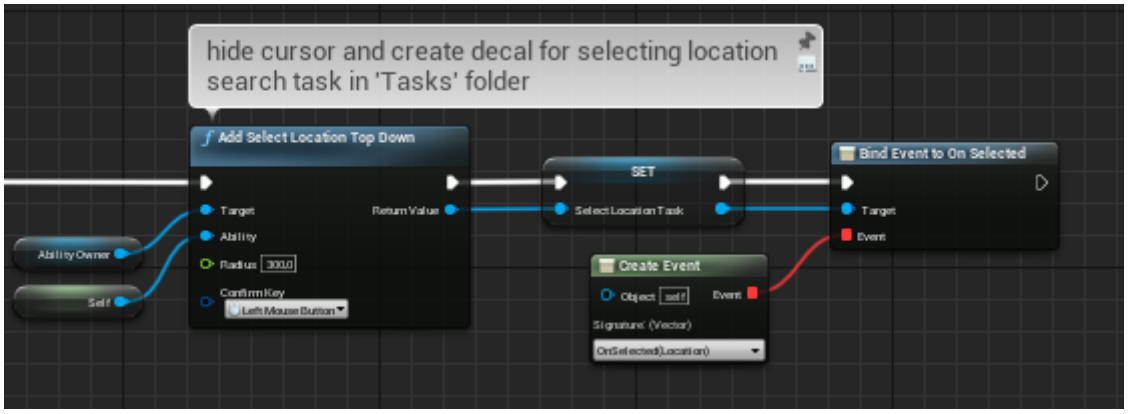
There are two events in class



When you apply FX, you need to call **Execute** event.
Remove event is called, when effect is removed, in order to stop playing FX.
See my custom FX in the demo.

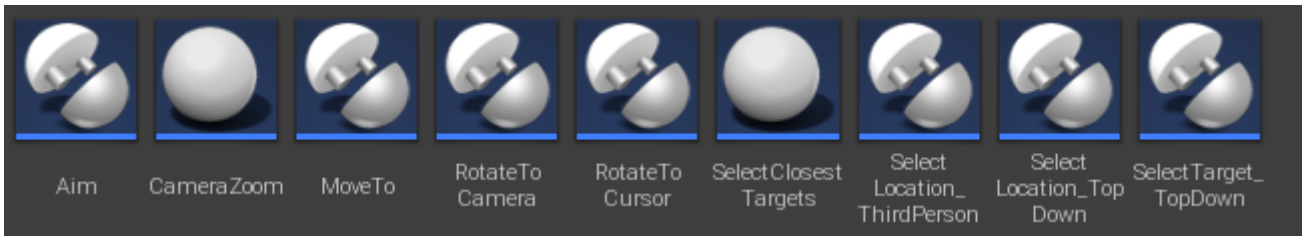
Task

Sometimes, when you create complex abilities, you may need to call logic in a certain time and perform it for a certain time too. After this, the logic shouldn't be performed, but the ability is still activated.
Also it is useful not to write the same logic for different abilities, and reuse it.
I decided that the components or actors are suitable for this. Each Task is a component or actor.
Since we cannot add component to component, in some cases I use Actor class for tasks.
We can add a component or actor in ability actor class and it will be automatically activated (if you implement the Begin Play event).

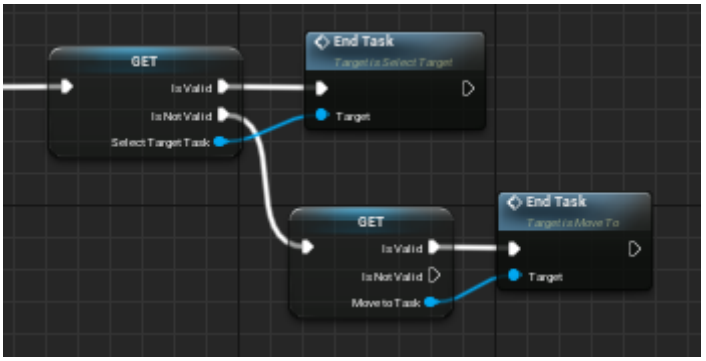


we can subscribe to events that occurs in this task

I store all Tasks in a separate folder. There is no parent class. Each task is a unique class.



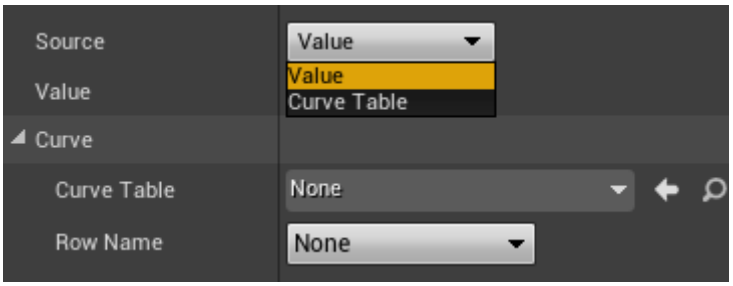
When the ability is canceled or ended, the task can be removed (ofc it can be removed automatically when finished)



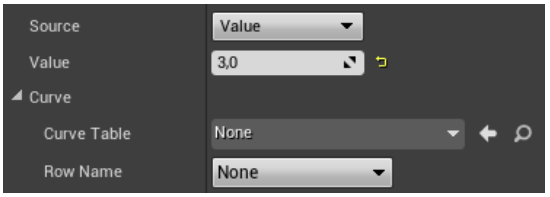
On Practice

Working with Float data

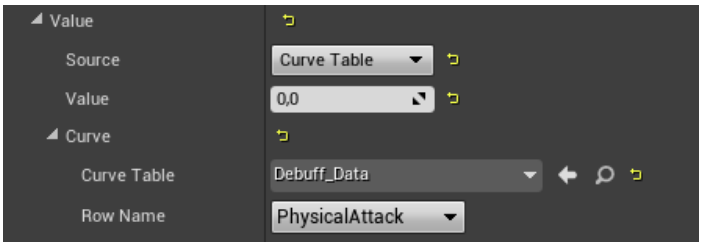
As an example see **TopDown_Debuff** ability, and **Debuff** effect.
When you set any float parameters for Effect or Ability there are two sources: **Value** and **Curve Table**.



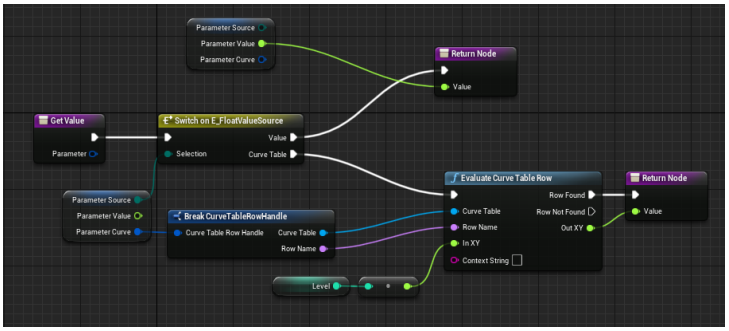
If your Effect or Ability has only 1 level, you can use **Value** source.
For **Value** you just set the value you need.



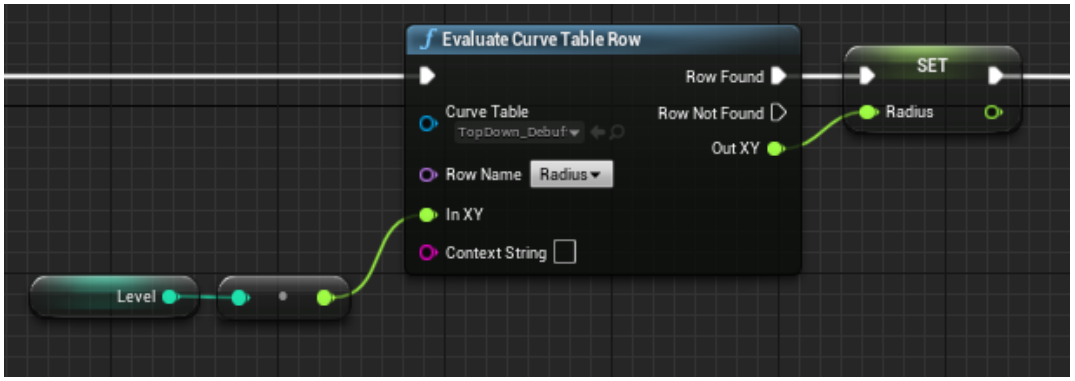
If your Effect or Ability has more levels, it makes sense to take data dynamically in accordance to the current level. For this you need to use a **Curve Table**.
For the **Curve Table** you need to set the curve table and row name.



Using a very simple function, Effect or Ability get needed value.



Also it can be used from BP for those parameters that are not in details panel (see TopDown_Debuff)

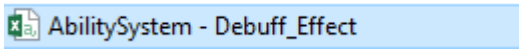


How to create Curve Table

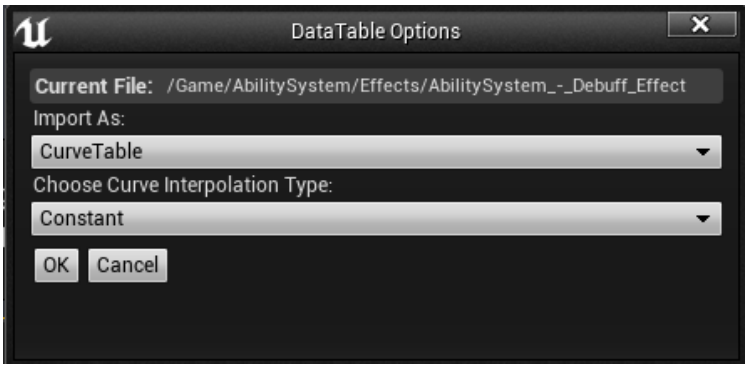
1. Curve table is a CSV or JSON file
2. Let's create new table in Google Sheets

A	B	C	D	E	F
	1	2	3	4	5
Duration	20	30	40	50	60
MaxAbsorbedDamage	500	1000	1500	2000	2500
PhysicalDefense	0.01	0.02	0.03	0.04	0.05
MagicalDefense	0.01	0.02	0.03	0.04	0.05

3. Export a sheet as CSV file
   File -> Download -> CSV (current sheet)
   You will get the file on your disk



4. Drag and drop this file to any folder in content browser
   The special window will appear
5. Select Curve table and Constant options



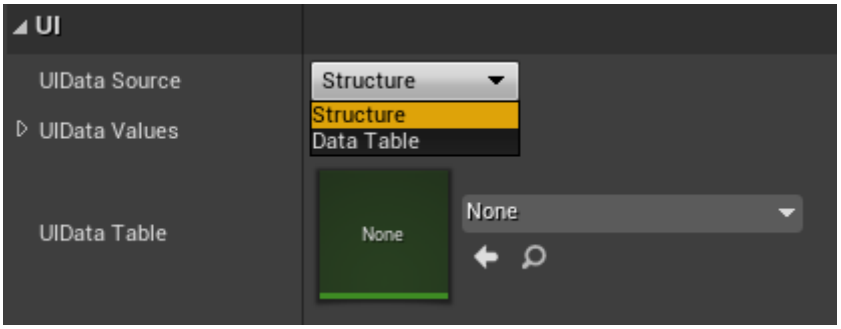
6. Press OK and open file

	1	2	3	4	5
Duration	20	30	40	50	60
MaxAbsorbedDamage	500	1 000	1 500	2 000	2 500
PhysicalDefense	0,01	0,02	0,03	0,04	0,05
MagicalDefense	0,01	0,02	0,03	0,04	0,05

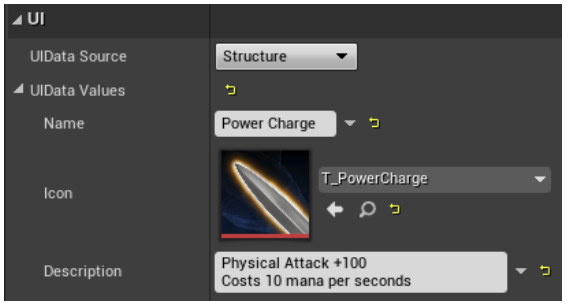
Now you can use this curve table, where column is a Level, and row is a parameter.

Working with UI data

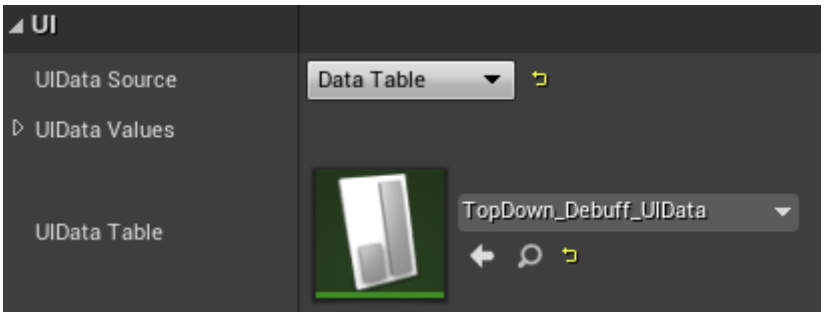
As an example see **TopDown_Debuff** ability, and **Debuff** effect.
When you set any UI parameters for Effect or Ability there are two sources: **Structure** and **Data Table**.



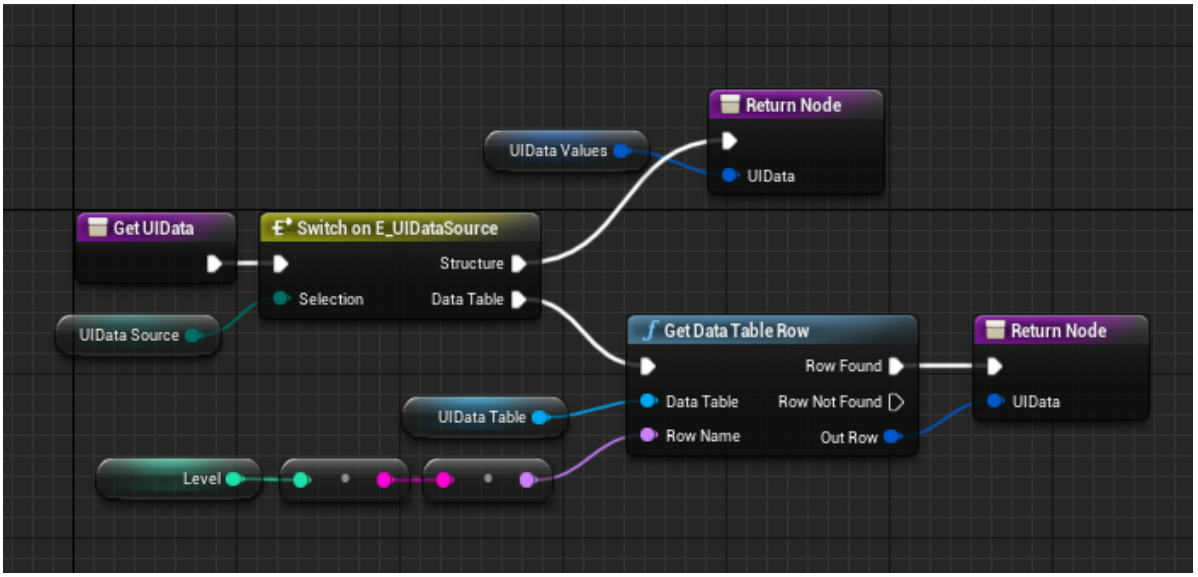
If UI Data are not dependent on level, or can be generated (like duration, stack count etc) you can use **Structure** source.
For **Structure** you just set the parameters you need.



If UI data are dependent on level and cannot be generated, you need to use a **Data Table**.
For the **Data Table** you need to set the data table.



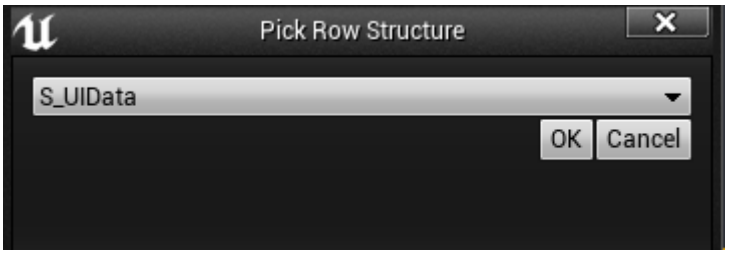
Using a very simple function, Effect or Ability get needed parameters.



How to create Data Table

I created a special **S_UIData** structure. You can add more variables to it if you need.

1. Miscellaneous -> Data Table -> Select structure



2. Set data in data table

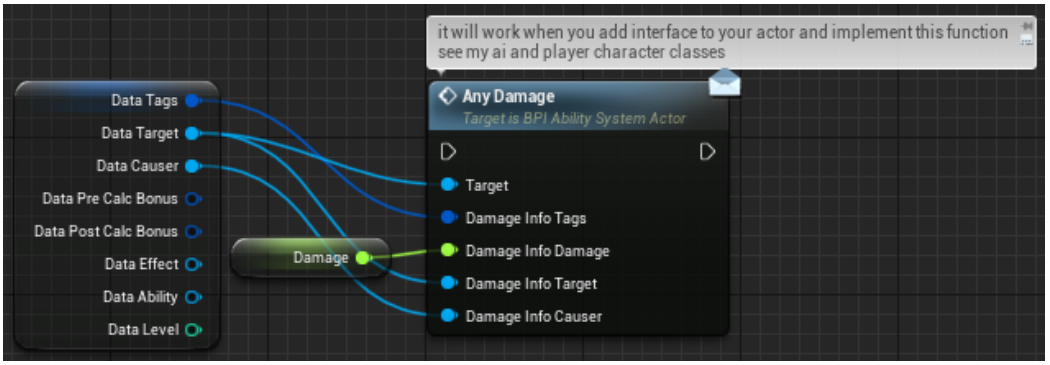
	Name	Icon	Description
1	Debuff	Texture2D'/Game/AbilitySystem/GUI/Remove/T_Debuff.T_Debuff'	Applies a level 1 "Debuff" effect to all enemies in selected area Area radius: 200 Physical and Magical Defense -20% Physical and Magical Attack -20%
2	Debuff	Texture2D'/Game/AbilitySystem/GUI/Remove/T_Debuff.T_Debuff'	Applies a level 2 "Debuff" effect to all enemies in selected area Area radius: 250 Physical and Magical Defense -25% Physical and Magical Attack -25%
3	Debuff	Texture2D'/Game/AbilitySystem/GUI/Remove/T_Debuff.T_Debuff'	Applies a level 3 "Debuff" effect to all enemies in selected area Area radius: 300 Physical and Magical Defense -30% Physical and Magical Attack -30%
4	Debuff	Texture2D'/Game/AbilitySystem/GUI/Remove/T_Debuff.T_Debuff'	Applies a level 4 "Debuff" effect to all enemies in selected area Area radius: 350 Physical and Magical Defense -35% Physical and Magical Attack -35%
5	Debuff	Texture2D'/Game/AbilitySystem/GUI/Remove/T_Debuff.T_Debuff'	Applies a level 5 "Debuff" effect to all enemies in selected area Area radius: 400 Physical and Magical Defense -40% Physical and Magical Attack -40%

Now you can use this data table, where row name is a level.

Deal Damage

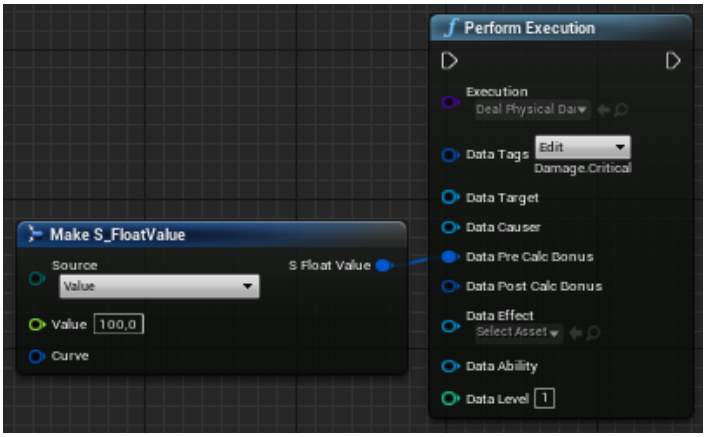
There are two ways to deal damage:

- call **AnyDamage** interface function directly

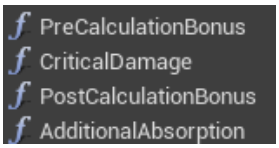


- call execution class that calculates and deals damage to target=> BP_Execution **AnyDamage** is called in this formula. Also execution class takes into account Target and Causer attributes.

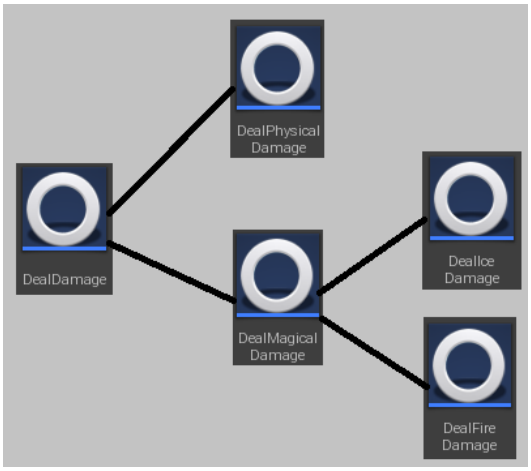
You can call execution from any place you want: when you use effect or cast ability etc.
Also when you call an execution class, you can influence the result by adding bonuses and tags.
For example, ability can always deal critical damage (for this we need just add tag Damage.Critical).
Or ability deals +100 bonus damage etc.



In the demo, I have only one big formula - this is the **DealDamage** class.
I calculate damage in four steps. One function for each step.

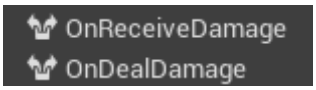


In **DealDamage** I check full immunity, calc base damage, crit, base absorption and apply bonuses.
I can create child classes, and override the functions.
So, for example, in **DealPhysicalDamage** and **DealMagicalDamage** I override **AdditionalAbsorption** function and add checking for 'Shield' effects and if the target has this effect, damage is absorbed.
After damage is calculated, I call **DealDamage** function

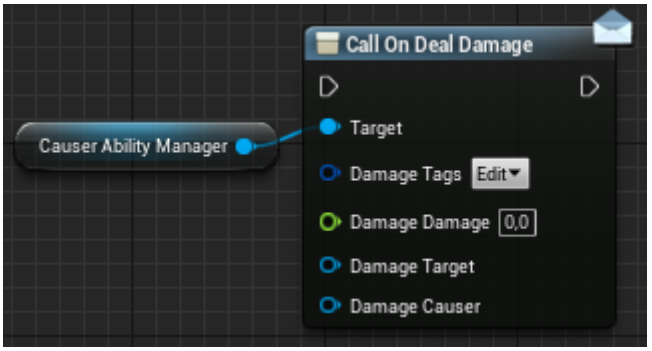


So that the formula will be universal for all damage types, I moved all parameters in the details panel. When you create a child class from DealDamage, you can set the following parameters.

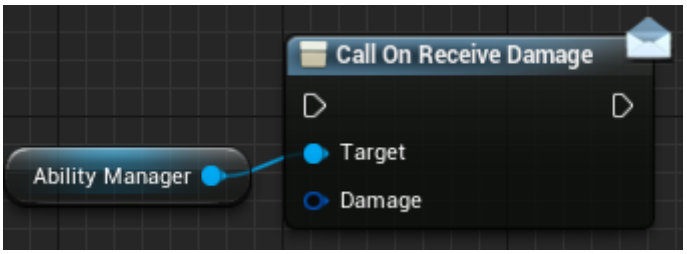
Also note that the Ability Manager has two Event Dispatchers.
These events are very useful if we want to implement such effects like Vampirism, Reflection etc.



OnDealDamage is called in the execution class (**DealDamage** function).

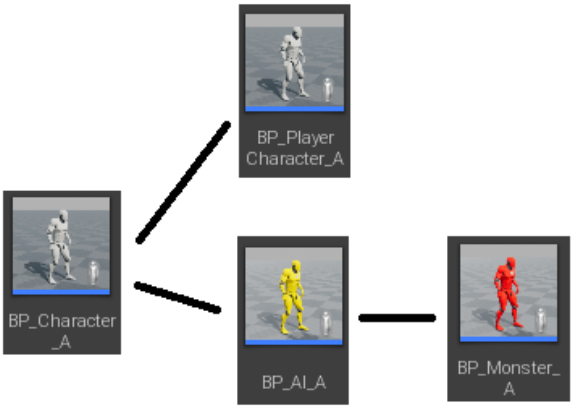


OnReceiveDamage is called in character class (**AnyDamage** event).

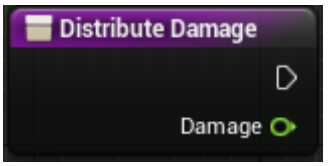


Damage Distribution

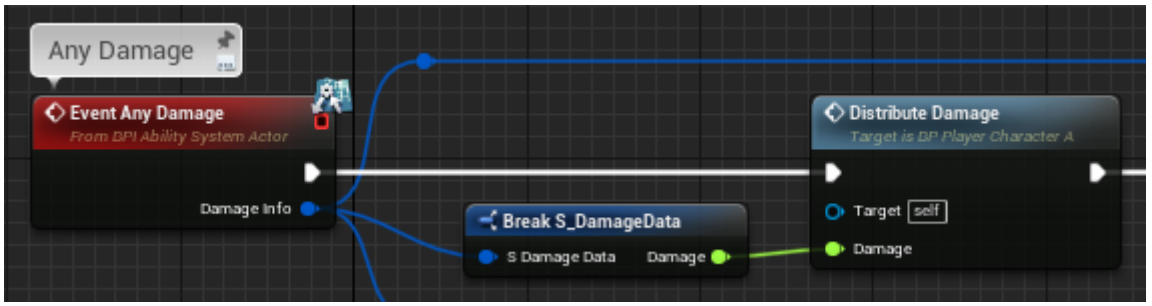
In demo I have the follow hierarchy of classes



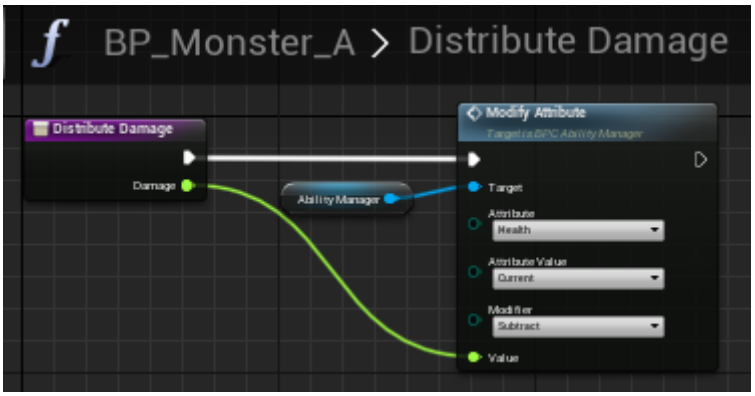
BP_Character has **DistributeDamage** function.



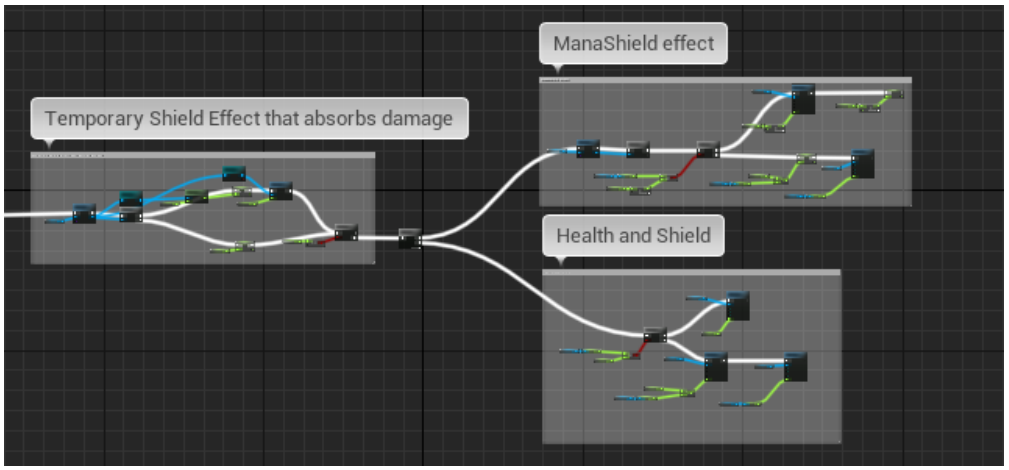
This function is overridden in **BP_PlayerCharacter** and **BP_Monster** (**BP_AI** in RPG Tools).
This function is called when character receives damage



For Monster I just modify Health

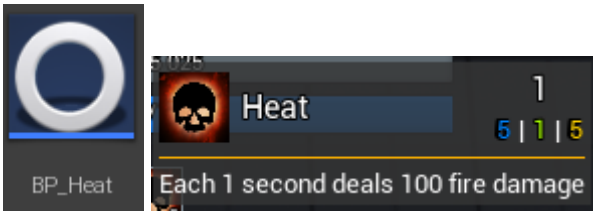


For Player I have Health, Shield and Mana attributes, and received damage can be distributed between these attributes.
Also here we can check if actor has special effects and abilities. In the demo, if the player has **Overall_Shield** ability, the damage is absorbed by this shield.


ManaShield effect
Temporary Shield Effect that absorbs damage
Health and Shield

Damage over time

Let's look at **Heat** effect.

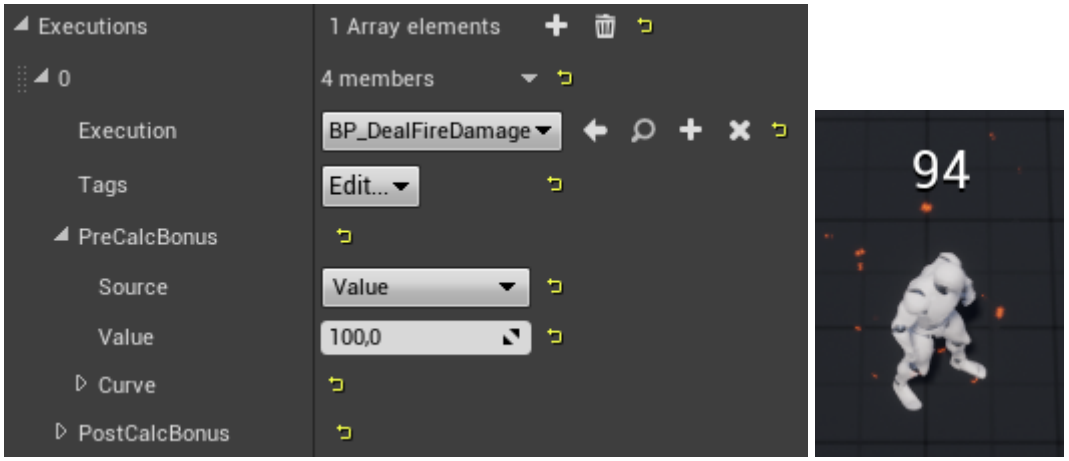


Effect is periodic and stackable

I use a formula with PreCalcBonus.

This formula takes into account Causer attributes, but since causer is Area that has no any attributes, calculation is based on PreCalcBonus (100).

Final damage is less, since formula takes into account defense attributes to calculate final damage.

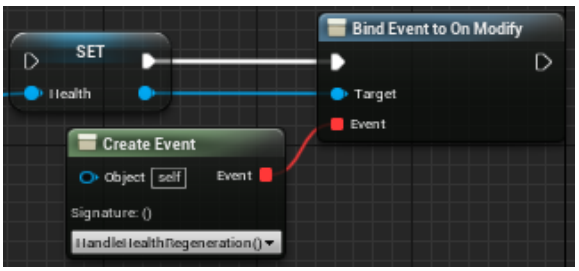


Regeneration

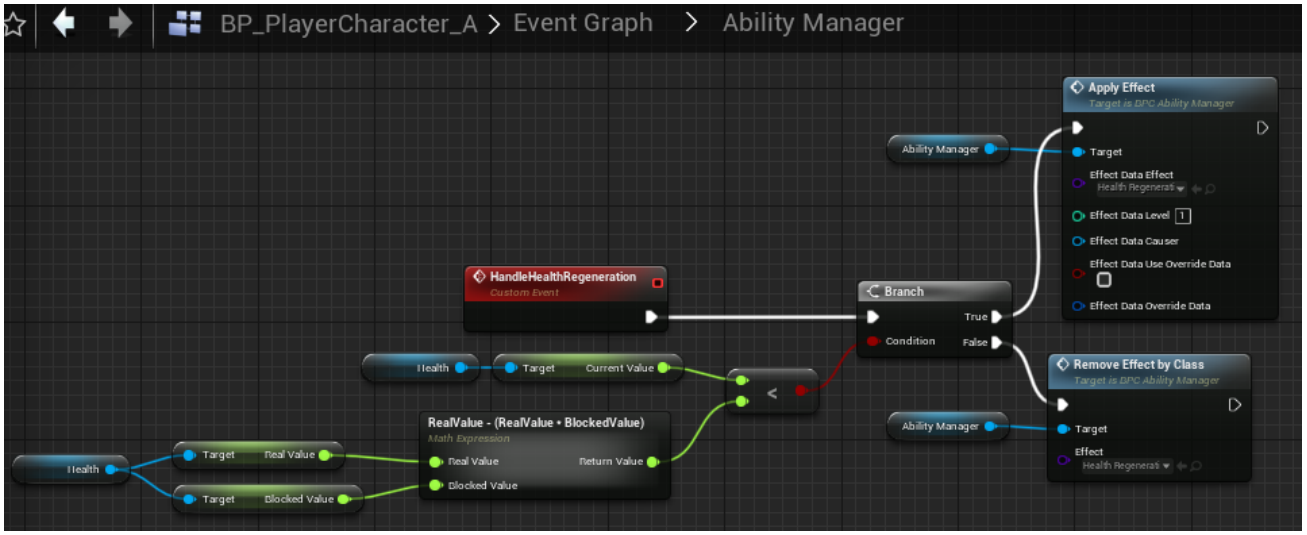
In each character class, at **BeginPlay**, after Ability Manager is initialized, I get reference to a certain attributes:

- for AI: health
- for player: health, shield, mana, energy

I save this references and bind to **OnModify** event

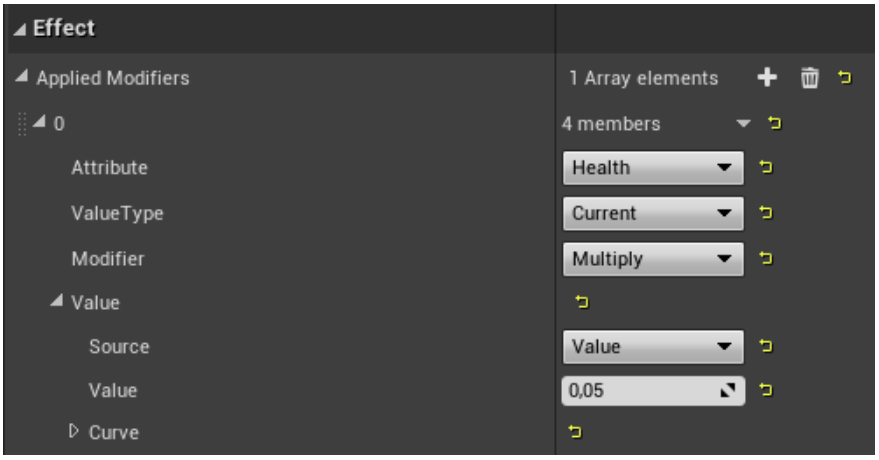


In EventGraph I created the special functions **Handle<Attribute>Regeneration**.
The function is called when the attribute is modified. If CurrentValue != RealValue, regeneration effect is applied. If CurrentValue is max, the regeneration effect is removed.



Regeneration effect is periodic.

For **Monster**, I use simple regeneration (see **MonsterHealthRegeneration**).



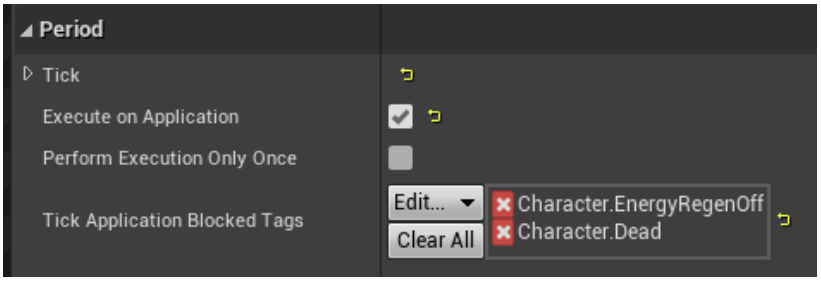
5% health is restored each 1 second

For **Player**, I use regeneration with a special execution class.
Player has HealthRegeneration and ManaRegeneration attributes.

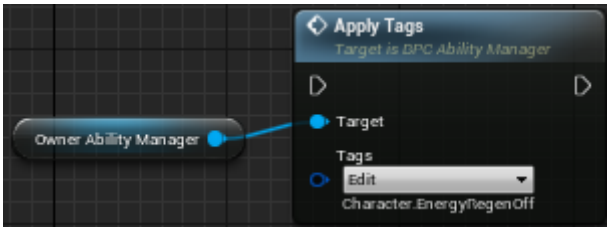
Using this class, we can restore current value based on these attributes.
You can use the Execution class with any logic (formula) you need.

For periodic effect, the execution class is created only one time. Then, each tick effect calls logic from it.

You don't need to remove effects if you want to **block regeneration**.
You can block regeneration by adding the special tags to the following parameter.
Let's look at the **EnergyRegeneration** effect.



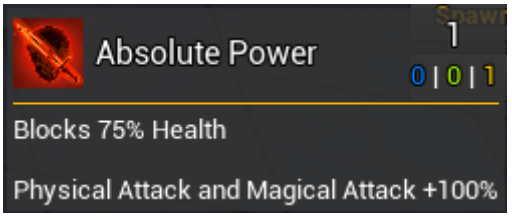
In the demo, when the player uses **Overall_FastMovement** ability, **Character.EnergyRegenOff** tag is applied to owner.



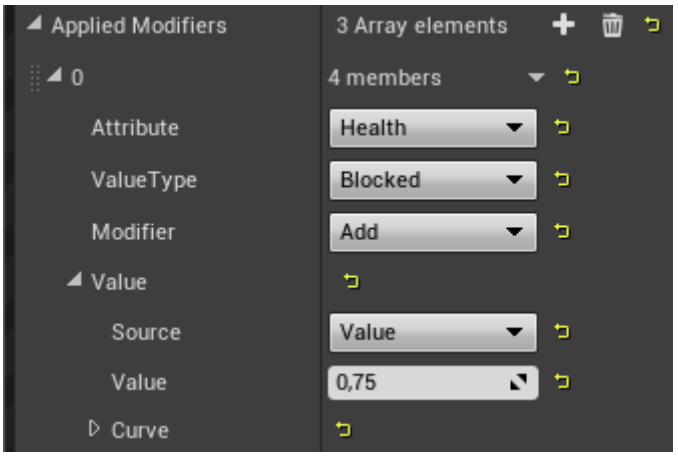
Also when the player is dead, I apply the tag **Character.Dead**, that blocks regeneration for all attributes.
When the tag is removed, regeneration automatically continues to work.

Attribute Value Blocking

Let's look at **AbsolutePower** effect.



To Implement this it's enough to change Blocked Value.



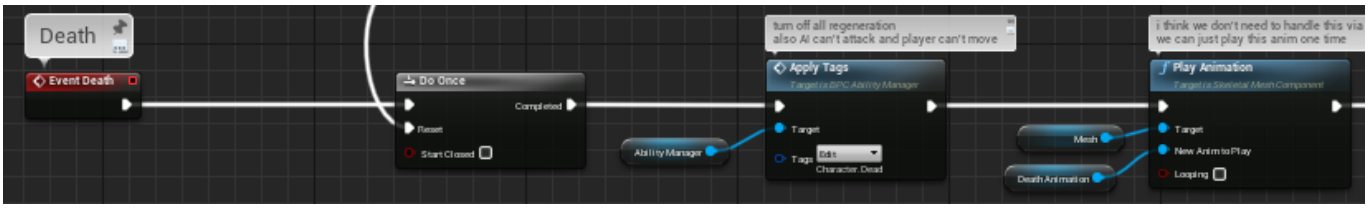
You work with percent from 0 to 100, where 0,75 is 75%.

Result

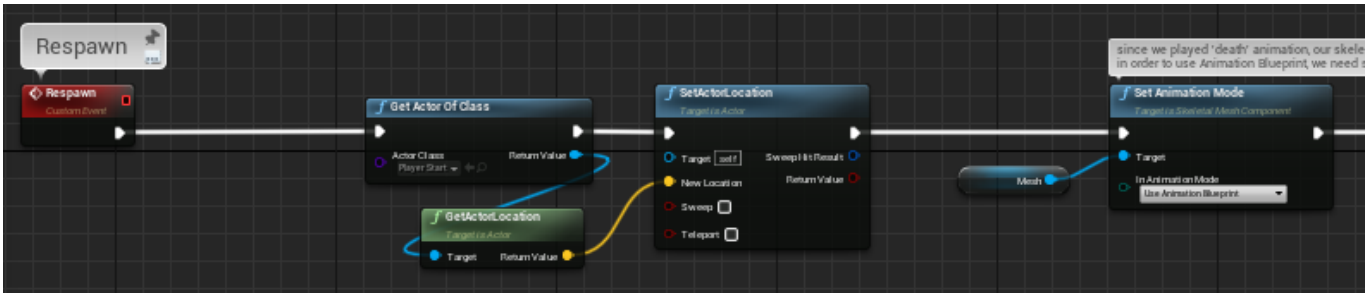
It will work correctly only with attributes with current value like Health, Mana, Shield, Energy etc

Death and Resurrection

I don’t handle playing death animation in Animation Blueprint class. I think it doesn’t make sense, because we need to play animation only one time, and we can do it in character class.
So, in character class (for player and ai) I call **Death** event, when **Health’s CurrentValue** = 0.
I apply **Character.Dead** tag and play animation



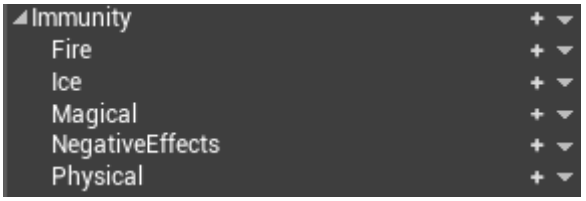
When I respawn the player, I remove the tag and return Animation Mode back to ‘UseAnimationBlueprint’.



When you create abilities, effects etc, you specify this tag in some parameters and the system knows that if owner has this tag, he cannot move, cast ability or cannot be attacked etc

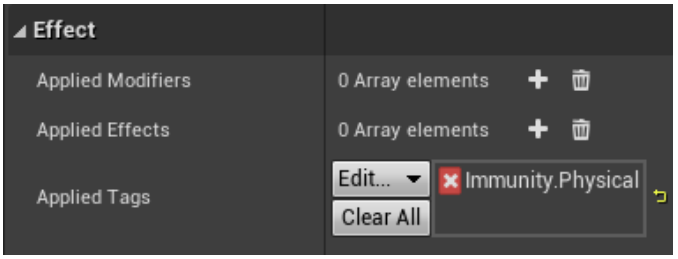
Immunity

In the demo I have 5 immunities: Physical, Magical, Fire, Ice, NegativeEffects.
To give immunity to target you need to apply Gameplay Tags to target

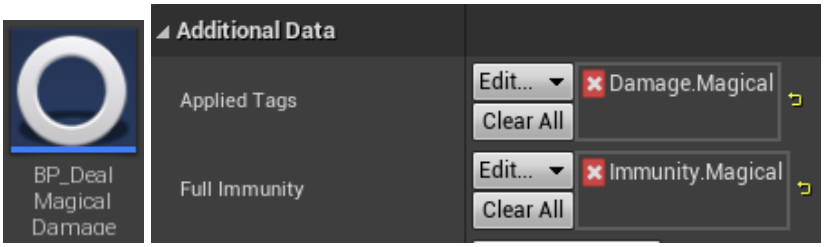


You can do this via blueprint

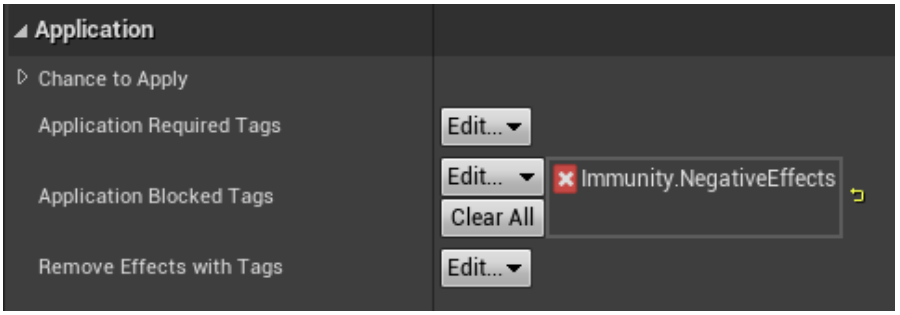
Or via Effect where you need set parameter **AppliedTags**



Physical, Magical, Fire and Ice immunity fully absorb damage.
I check for immunities in **DealDamage** execution class and its children



NegativeEffects immunity blocks negative effects application.
When you create effect, you need just add this tag to parameter **ApplicationBlockedTags**

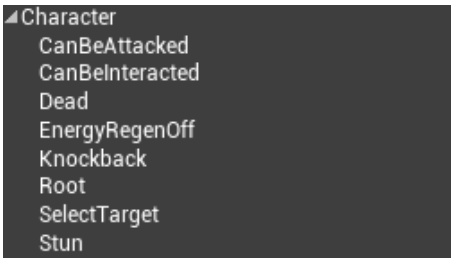


Controls

In the demo I have the following controls:

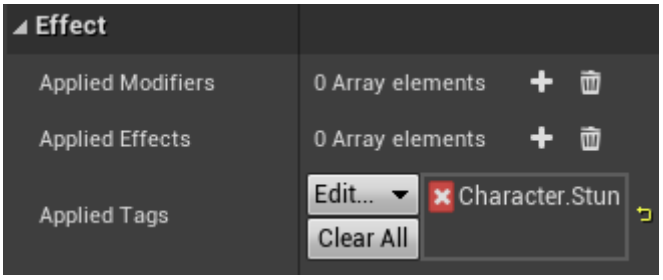
- **Root** and **SelectTarget** block movement
- **Stun**, **Knockback** block movement and ability casting

To apply control to target you need to apply Gameplay Tags to target

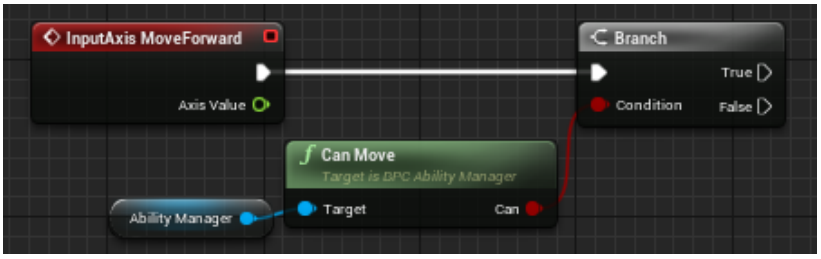


You can do this via blueprint.
See **SelectTarget** and **SelectLocation** tasks.
I apply the **Character.SelectTarget** tag while the task is active.
In **BP_PlayerController_TopDown**, when a player tries to move, I check if the player has this tag, and if he has, I stop movement.

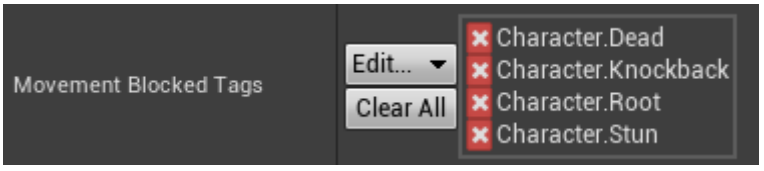
You can do this via Effect where you need set parameter **AppliedTags**



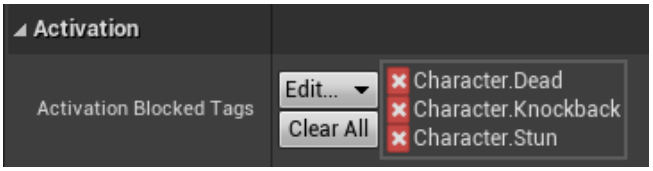
When Player or AI try to move, they ask Ability Manager about possibility to do this



Ability Manager has parameter MovementBlockedTags where you can set tags you need

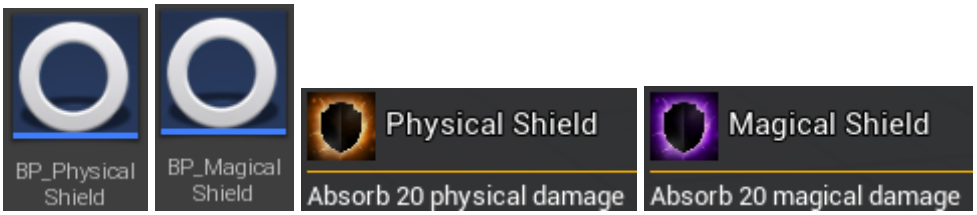


When you create new ability, you can set tags to **ActivationBlockedTags,** and if Caster will have these tags, ability won't be casted



Partial Damage Block

Let's look at **PhysicalShield** and **MagicalShield** effects.



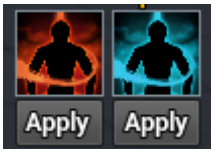
I handle this in execution class. See **Execution** and **DamageCalculation** sections.
I override **AdditionalAbsorption** function in these classes

and check if effect is applied, I minus fixed value

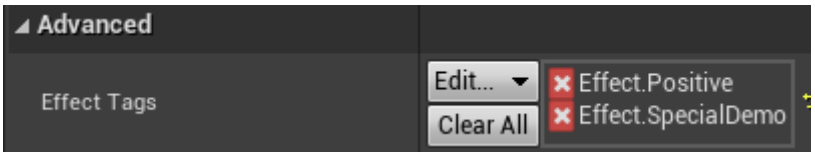
I will think about how to improve this method, because it is not automated.
With this method it is supposed there will not be many such effects in the game.

Interchangeable Effects

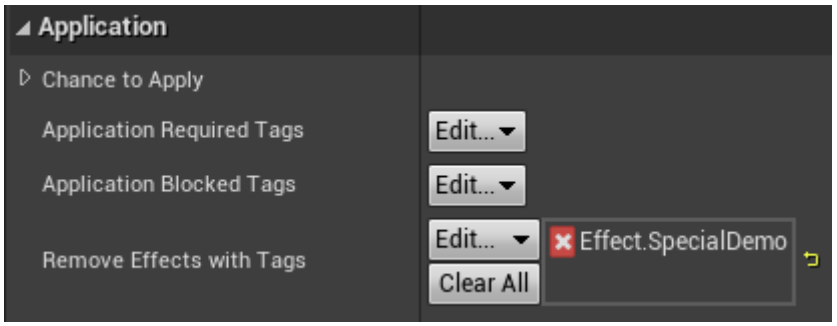
Sometimes It is necessary that one or more effects cannot be applied simultaneously.
For example, Attack Potion effect replaces the Defense Potion effect.
Let's look at AttackPower and DefensePower effects.



Effects.SpecialDemo tag is added to next parameter

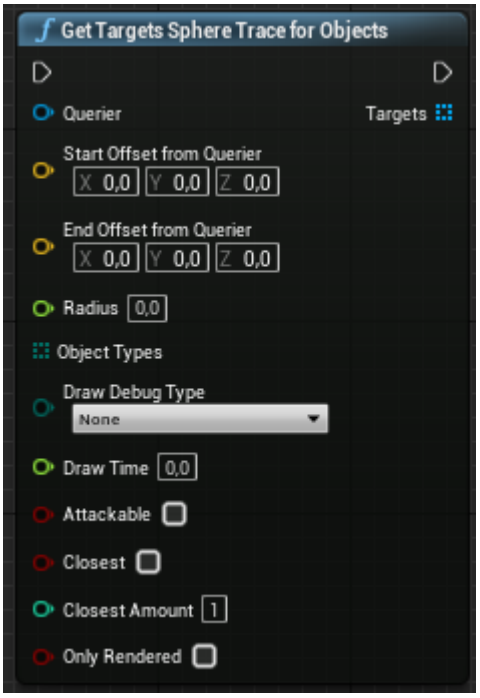


Also in Application section this tag is added to next parameter



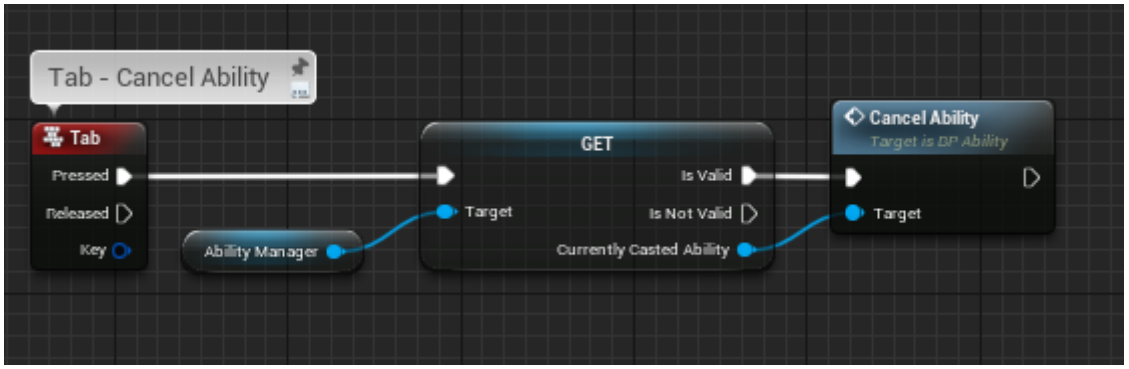
Now when Ability Manager will apply effect to target, first, it will remove all effects with specified tag, and then will apply new effect.

Get Targets

In several game situations, it may need to get a list of targets in a certain area. For example, when you want to deal damage to all targets in an area, or you need to find N the closest targets. In order not to write the same logic every time, I create library functions. I think it is enough to use library functions for this. I store such functions in **BPL_AbilitySystem**. I try to make them universal and at the same time not very difficult.	

Ability Interruption

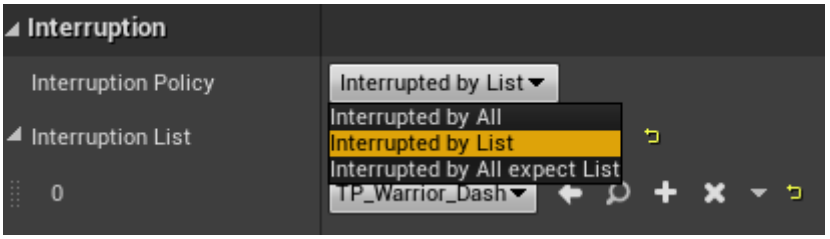
Ability can be canceled by pressing hotkey (in demo this is Tab).



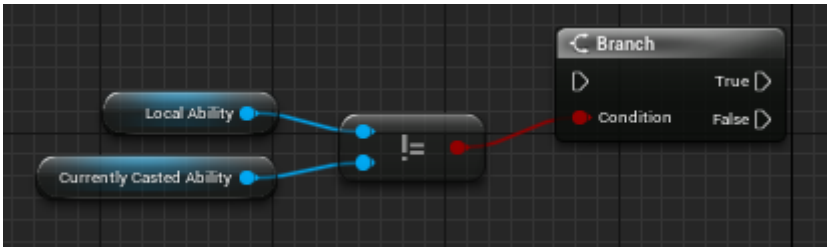
Also ability can be canceled by casting another ability. In this case you need to set appropriate settings for ability.

By default, ability cannot be interrupted by another ability, since **Interruption Policy** = **Interrupted By List**, and list is empty.

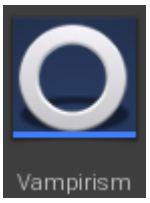
Open **TP_Warrior_Beam** ability, you will see that this ability can be interrupted by **Dash** ability,



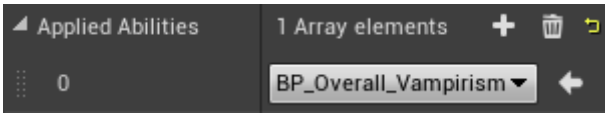
Also good to know that ability cannot be interrupted by itself. AbilityManager uses appropriate check in **TryActivateAbility** function.



Vampirism



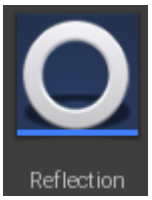
Vampirism is a passive effect that applies passive ability.



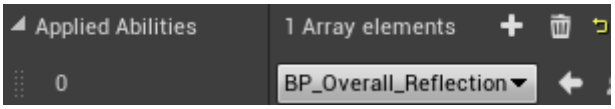
On Activation ability binds to **OnDealDamage** event, and when owner deals damage ability works and restores 25% damage to HP.

When effect is removed, EndAbility event is triggered in ability, and ability unbinds from **OnDealDamage** event.

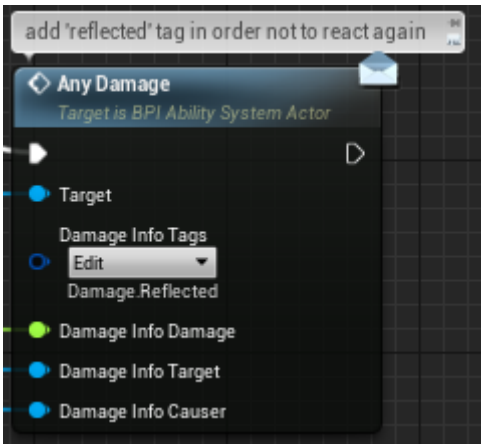
Reflection



Reflection is a passive effect that applies passive ability.



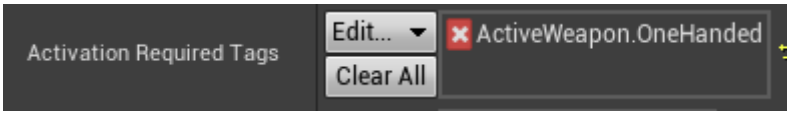
On Activation ability binds to **OnReceiveDamage** event, and when owner receives damage ability works and returns 25% damage to causer.
I added **Damage.Reflected** tag to damage info in order not to react again to this damage.



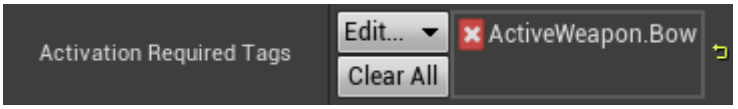
When effect is removed, EndAbility event is triggered in ability, and ability unbinds from **OnReceiveDamage** event.

Warrior and Archer

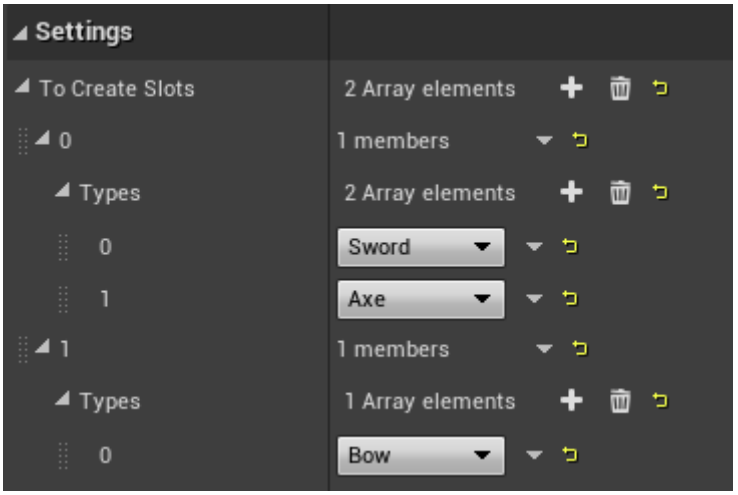
Here I describe how Warrior and Archer classes works in ThirdPerson mode in demo.
All Warrior abilities have required tag



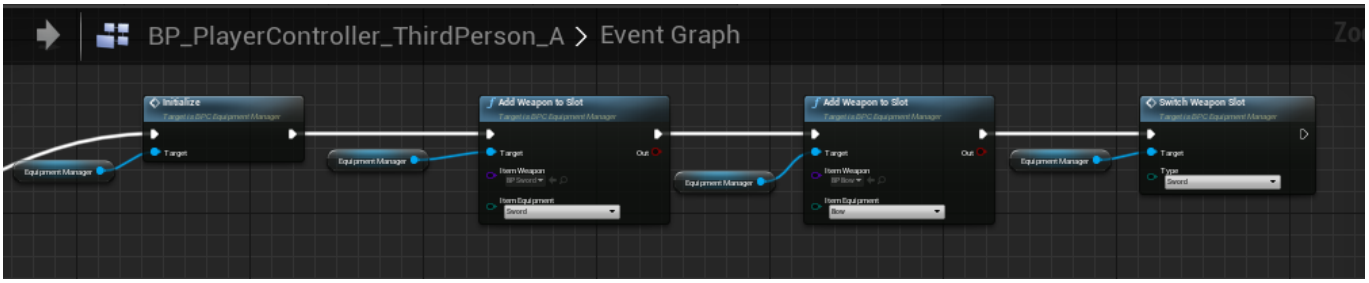
The same for Archer abilities



You cannot use warrior abilities if Bow is equipped and vice versa.
Equipment Manager is connected to **BP_PlayerCharacter_ThirdPerson** class with the next settings

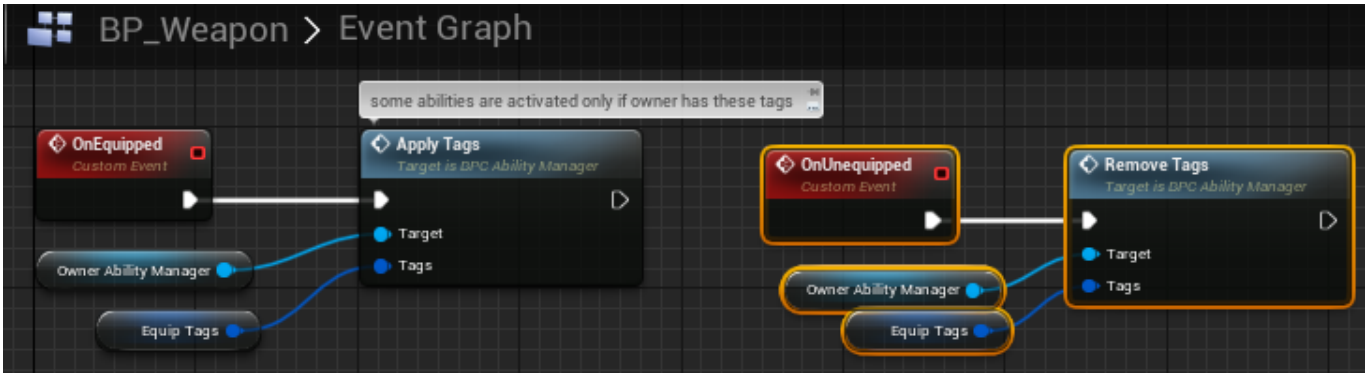


At start game, in player controller, I equip Sword and Bow weapons and make Sword is active.

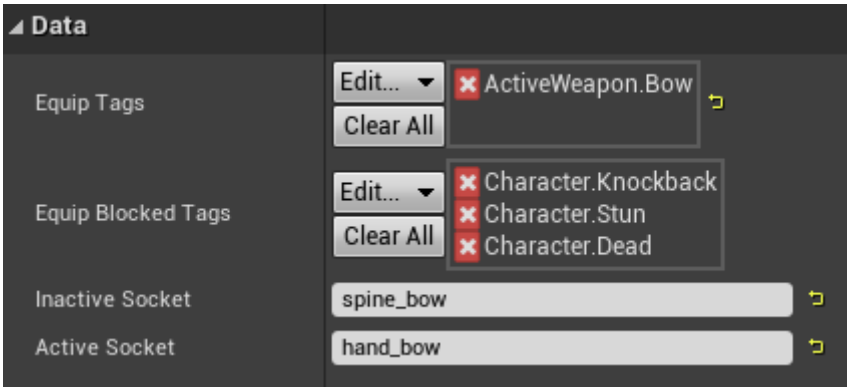


Each weapon is child of **BP_Weapon** actor class with several parameters

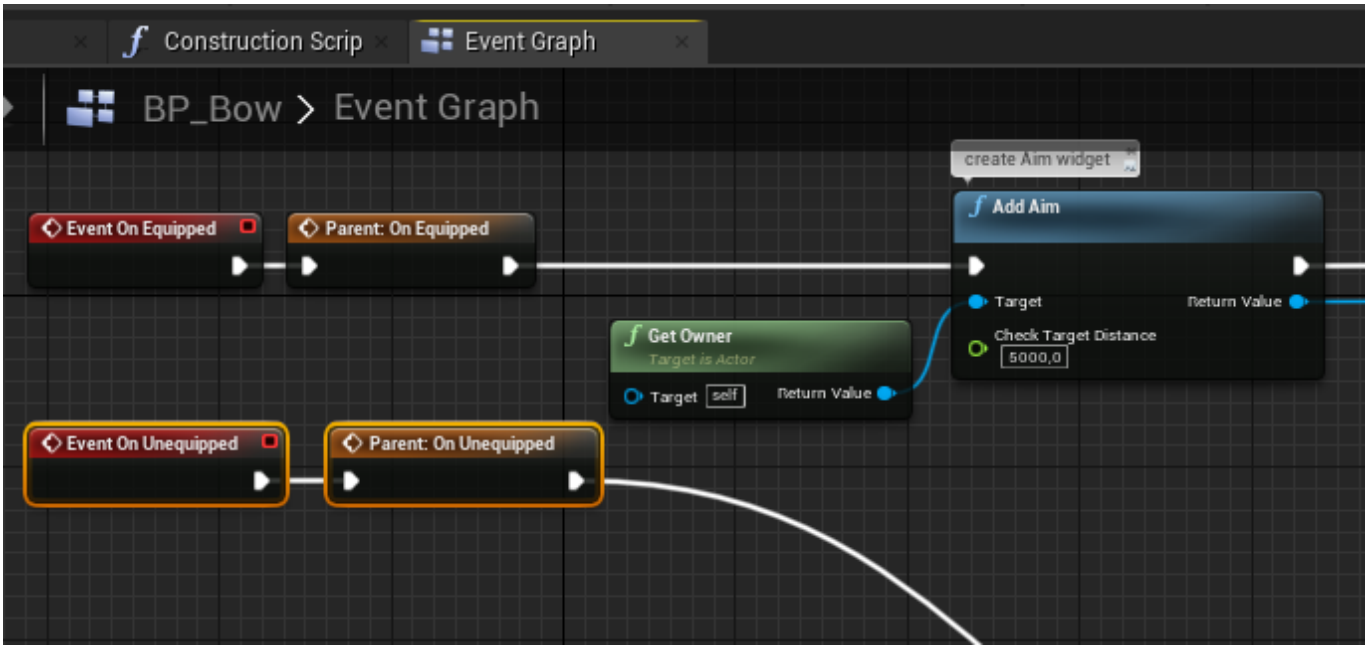
In **BP_Weapon** the next logic is implemented



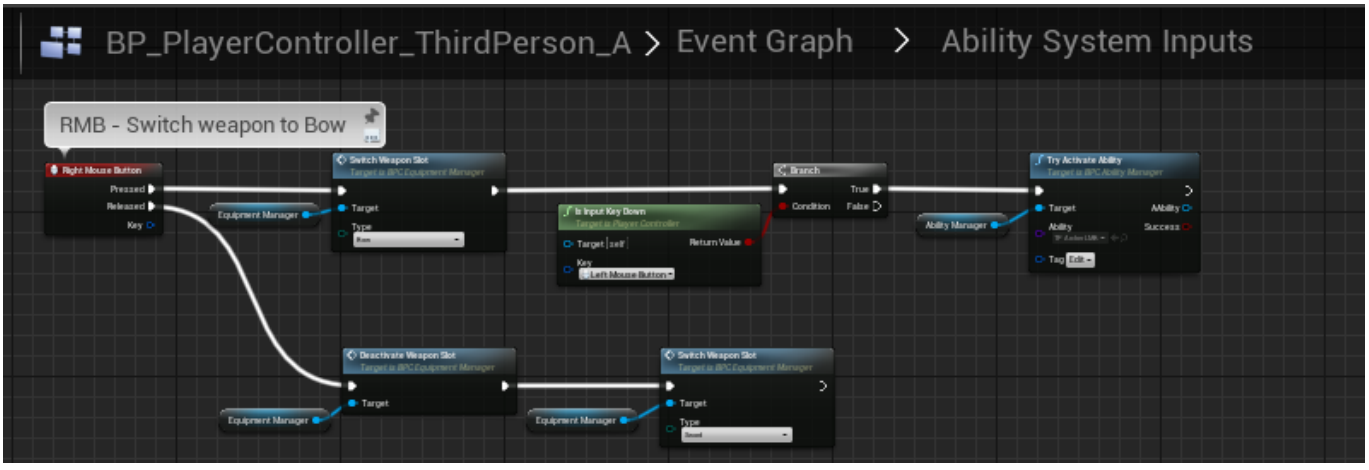
Let's open **BP_Bow.** In details panel we see parameters



And two events. Here we can change Animation instance, Camera Zoom or add Aim widget on HUD.



OnEquipped event is triggered when weapon slot becomes active or is equipped to already active slot.
OnUnequipped event is triggered when weapon slot becomes inactive or is unequipped from active slot.
For switching between weapon slots I use RMB



Everything is quite simple. We have Bow abilities that we cannot use because player hasn't required tags. Using RMB we activate Bow weapon slot => Player takes a bow in hand and OnEquipped event is triggered (tags are applied, animation instance are changed, aim is added on HUD, camera is zoomed). Now we can use Bow abilities.

When RMB is released, Sword slot is activated.

Change Log

1.1

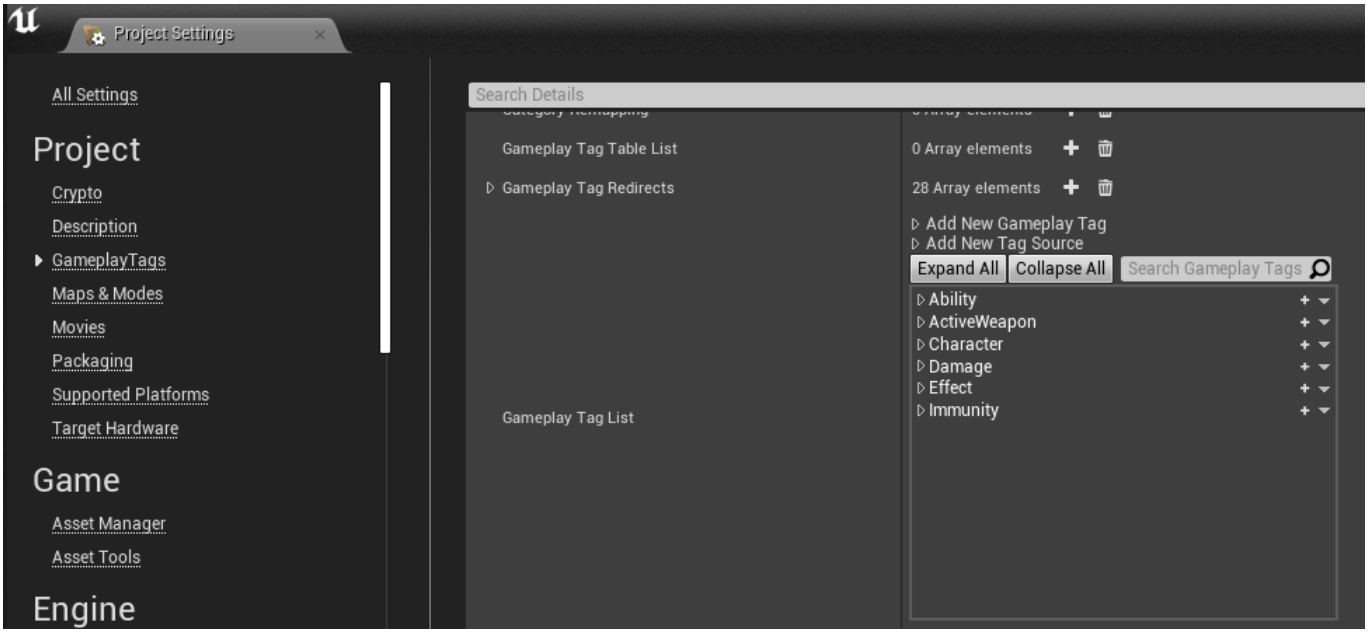
01.02.2022

- **Ability Hotkeys** renamed to **Ability Quick Slots**
- Now **Beam**, **Super Arrow**, **Rain of Arrows**, **Meteorite** and **Wrath of Heaven** deal damage to Tower
- Collision is removed from **VFX Rain of Arrows**, arrows no longer get stuck in the air when collide with other collision spheres
- **DeathAnimation** variable is added to BP_Character
- Fixed issue when Passive abilities **Reflection** and **Vampirism** are deactivated when owner is dead
  'Dead' tag is removed from ActivationBlockedTags
- Fixed issue when Player attacks himself in Top Down mode using LMB attack
  Check is added to Left Mouse Button event
- Fixed issue when Tower continues to attack Player even if he is dead
  *Attack timer is moved from **BP_Tower_Attack** to **BP_Tower**. Each tick (1 second) tower checks if the player is alive. If false, ability is canceled*
- Fixed issue. Now **Smart Multiple Arrows** selects 4 targets, not 10

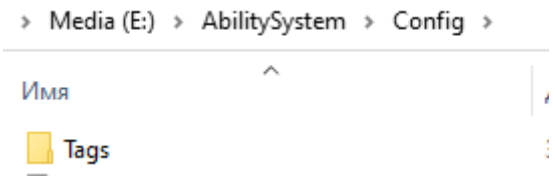
Connection + Video

Gameplay Tags

Before you will migrate Ability System, you need to copy gameplay tags to your project..
If you **create new project** from downloaded Ability System content and open it, in project settings you will find Gameplay Tag List. These tags are used in gameplay logic.



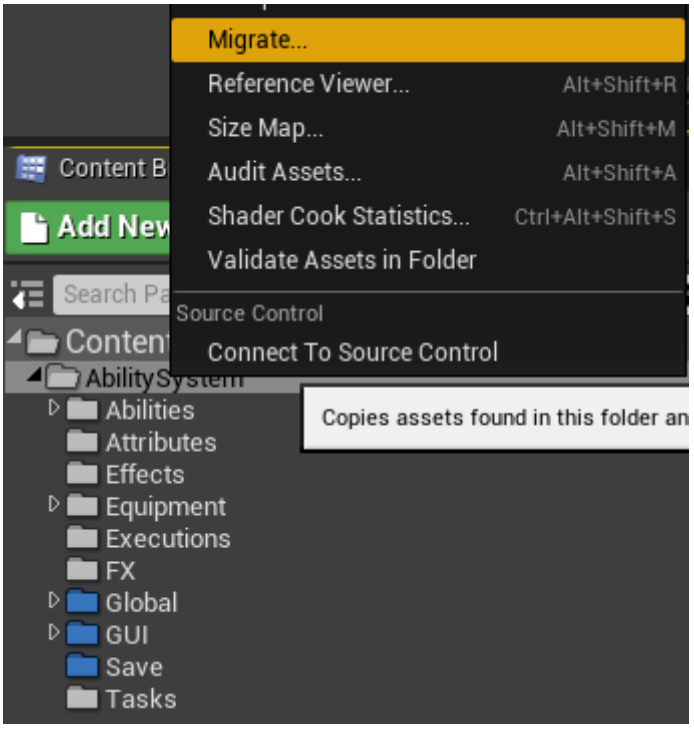
In order to copy them to your own project you need to open config folder and copy Tags folder to your project



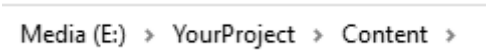
You can open your project and make sure the tags appear in project settings.
Only after this you can migrate all Ability System content to your project.

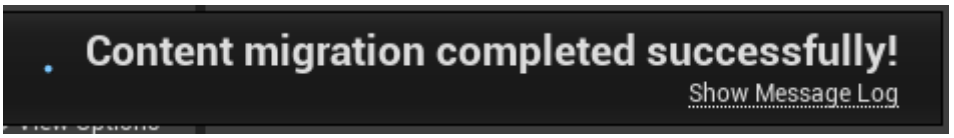
Migrate

Now when GameplayTags are copied to your project you can migrate AbilitySystem.
Press RMB on AbilitySystem folder and select **Migrate** option

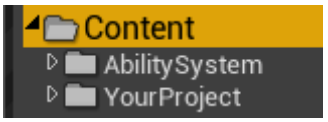


Press OK and select **Content** folder in your project



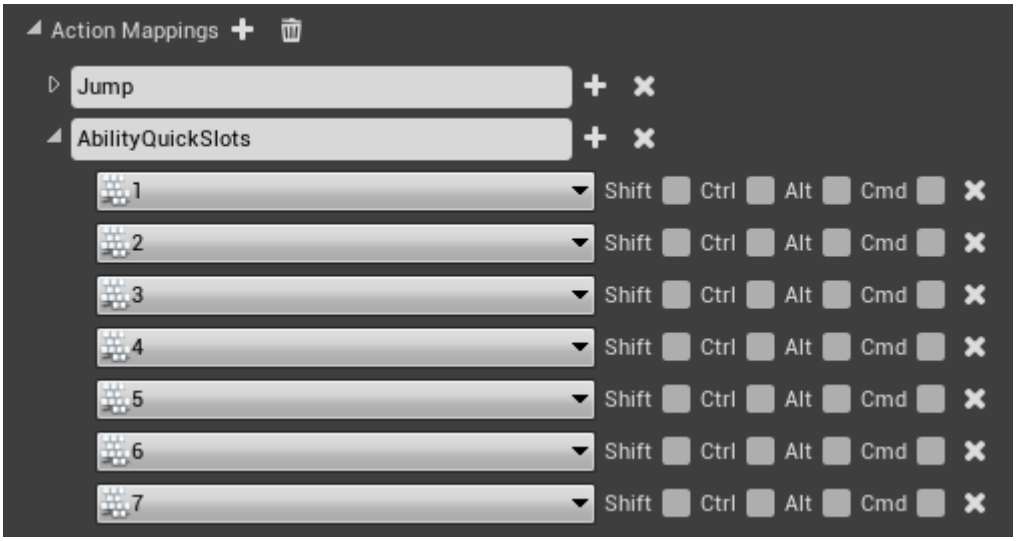


Now if you open your project you will see AbilitySystem folder with all content.



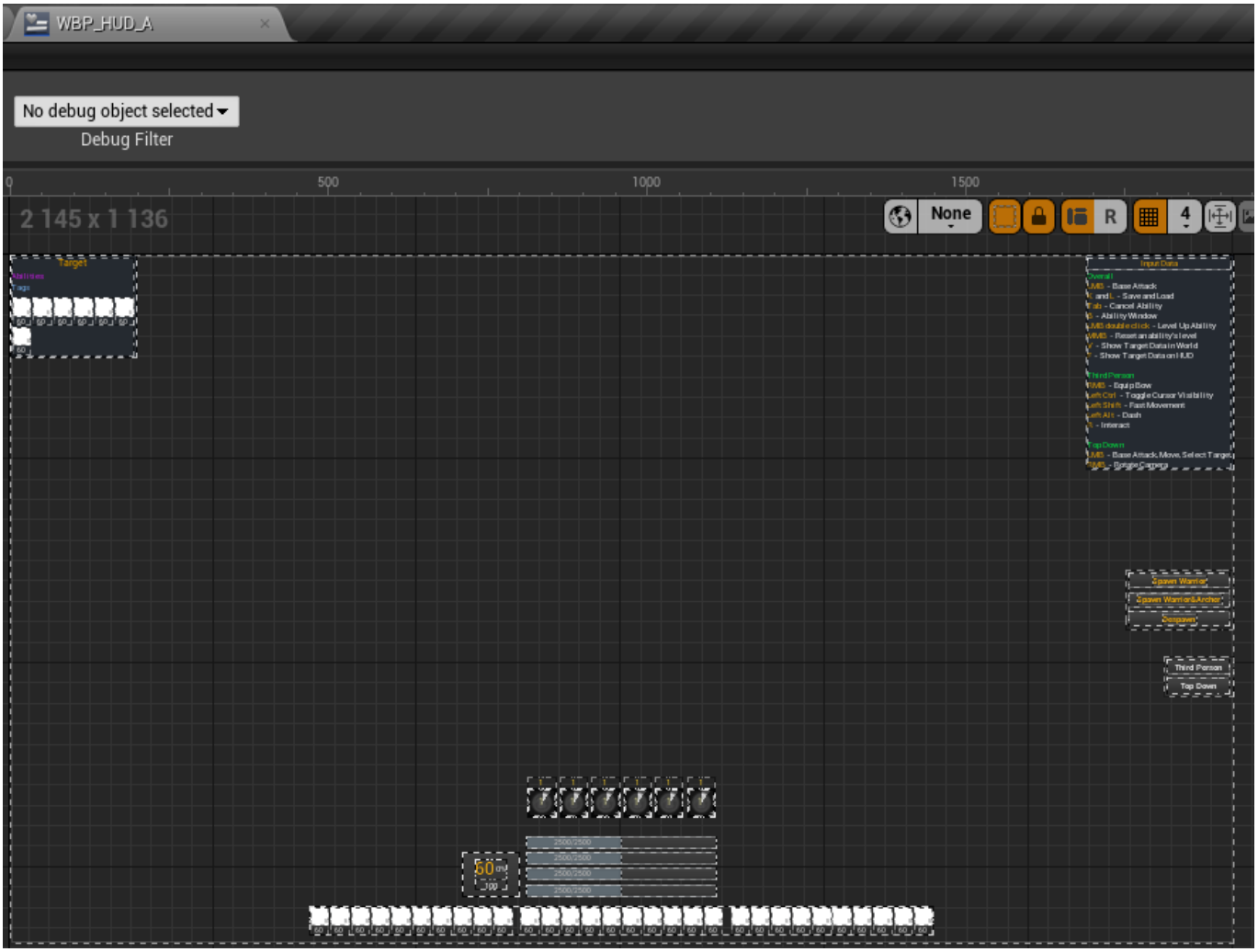
Inputs

Edit -> Project Settings -> Inputs -> Create AbilityQuickSlots Inputs

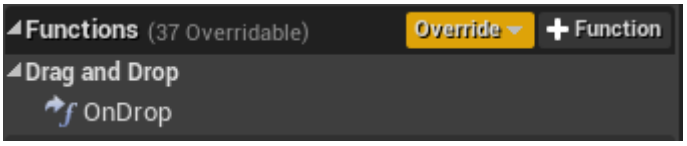


HUD

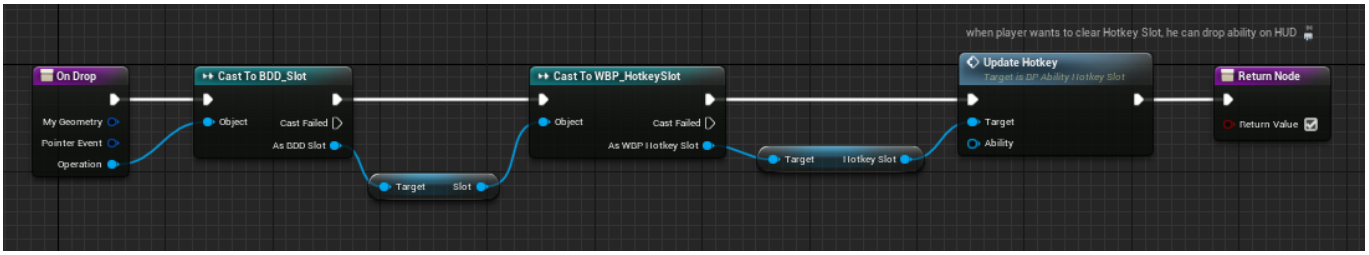
If you don't have HUD, you can use my HUD. Just note that I create HUD in Player Controller.
If you have HUD, copy widgets from my HUD to your HUD.



Open Graph and create OnDrop function.



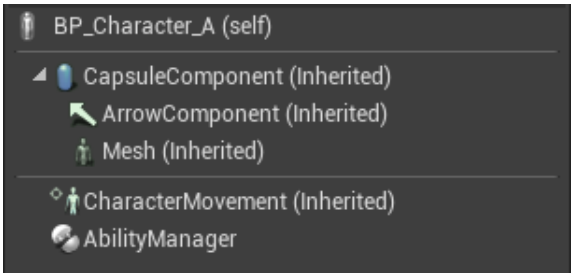
Copy logic from my HUD.



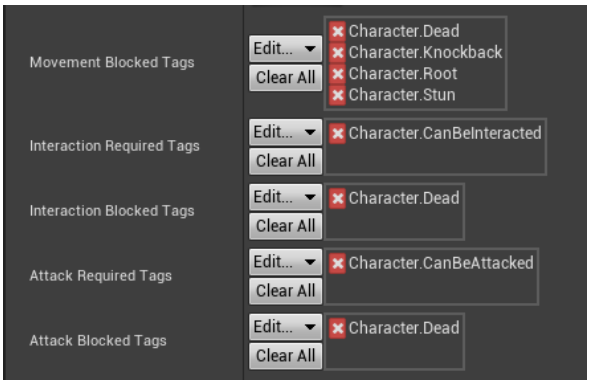
For the function to work make sure that Visibility = Visible for your HUD

Player Character

The simplest way is to use my player character classes. But ofc, most likely you want to use your own, so I will describe what logic you need to copy.
Add **BPC_AbilityManager**

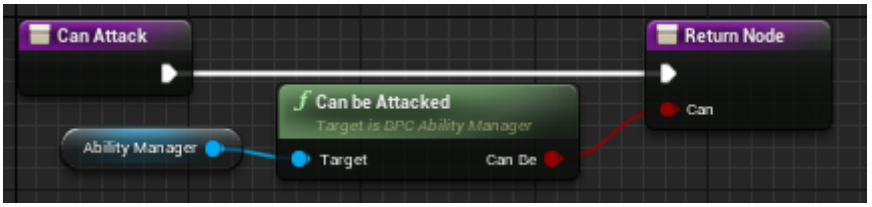


You don’t need to set any parameters in the details panel, but you need to know that the Ability Manager has a few parameters to determine when the owner can Move, can be Interacted, can be Attacked. If you open the manager, you will see these parameters and its default values. You can change them if you want. In the future I will try to move these parameters into a separate file.

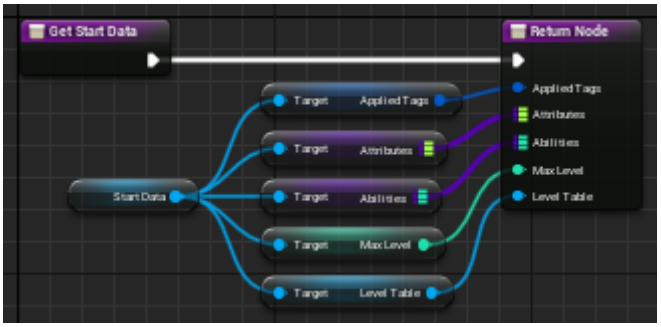


Add interface **BPI_AbilitySystem_Actor.**
Add logic to **GetAbilityManager** function.

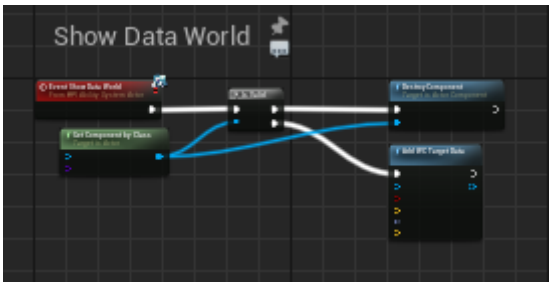
Copy logic to **GetAttack** function.



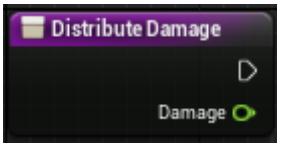
Copy logic to **GetStartData** function. You can use my presets as default value.



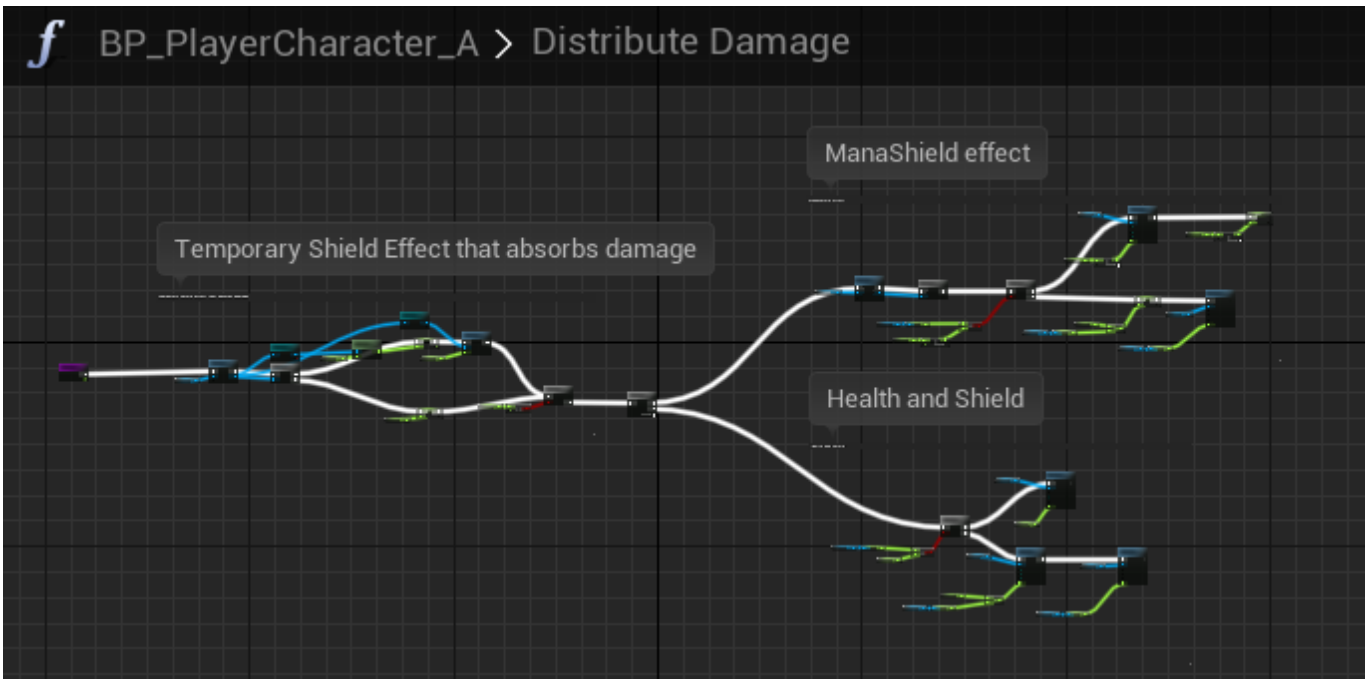
Copy logic from my **BP_Character_A** to your player character class. The logic is needed only for visual debugging.



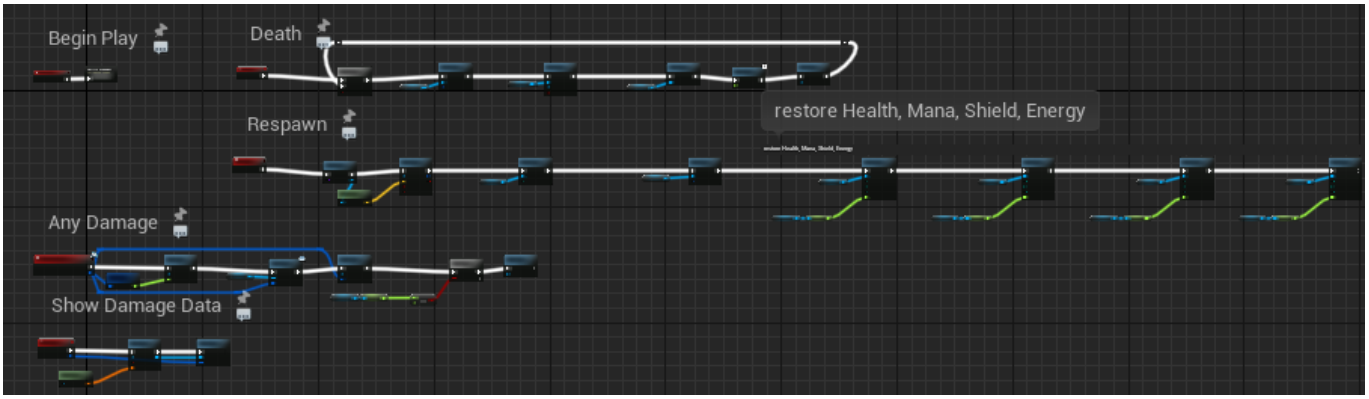
Create **DistributeDamage** function



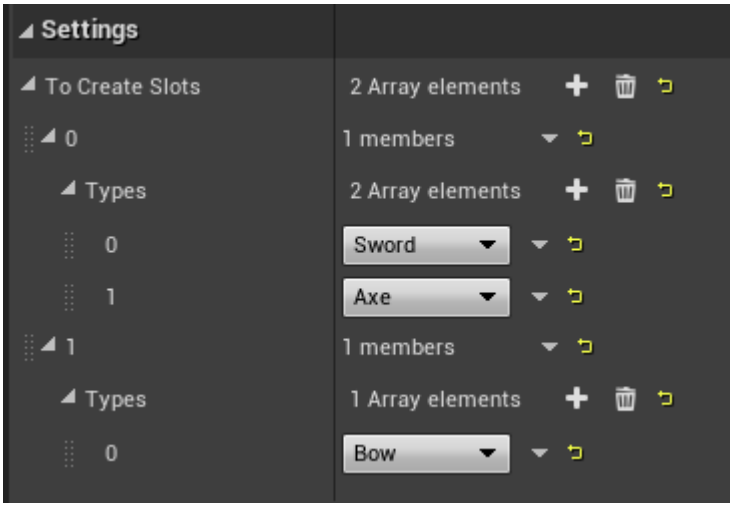
Copy logic from my **BP_PlayerCharacter_A** to your player character.



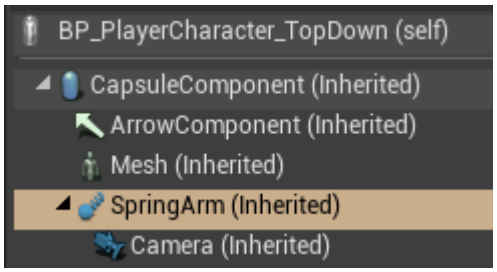
Copy logic from EventGraph (BP_PlayerCharacter_A).

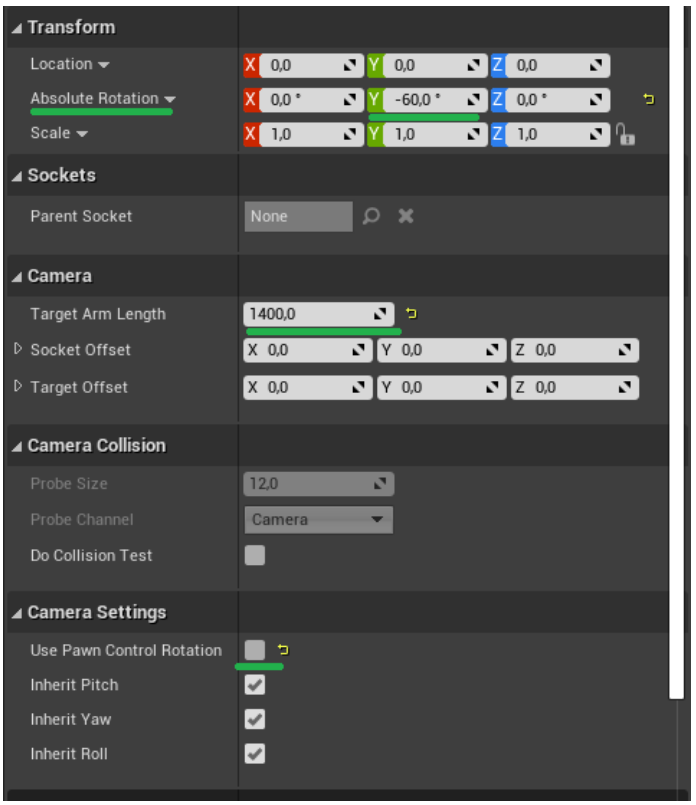


For **ThirdPerson** player character add **Equipment Manager** with the next settings.



For **TopDown** player character make sure that the next settings are specified

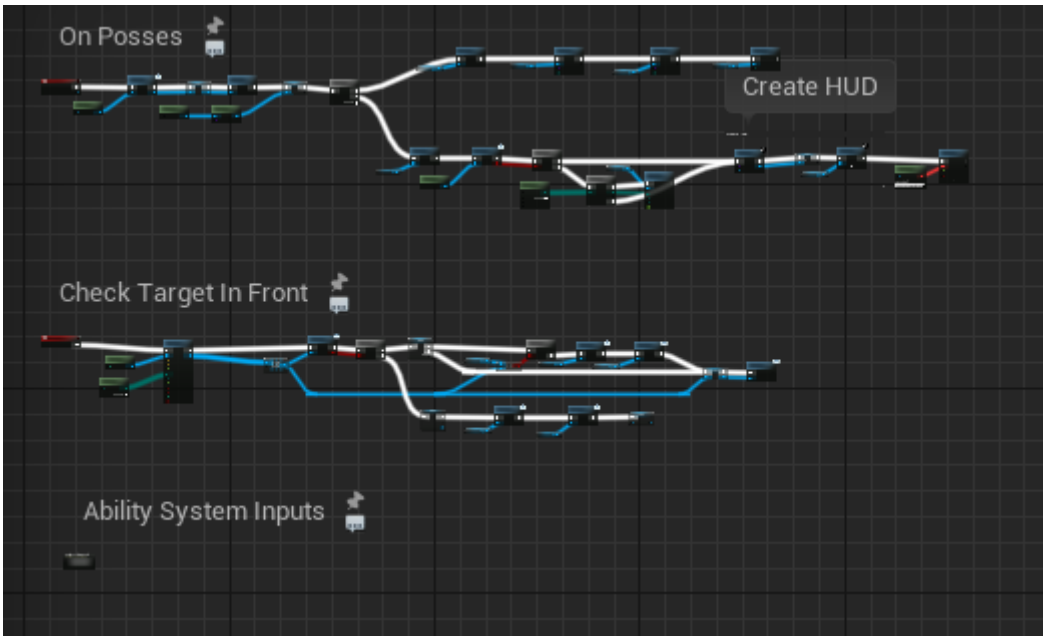




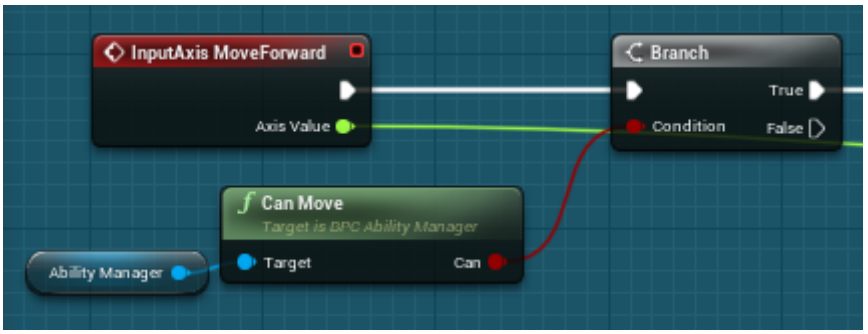
Player Controller Third Person

Add **QuickSlot Manager**

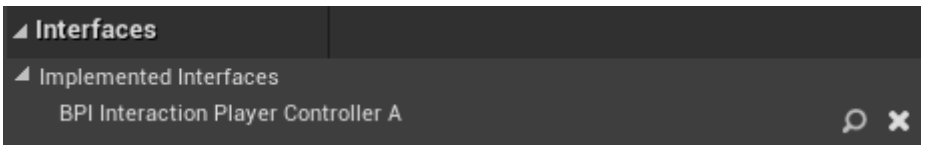
Copy the next logic to your player controller.
Note that I'm creating HUD widget in player controller.
Also note that some inputs can be the same as inputs in your project.

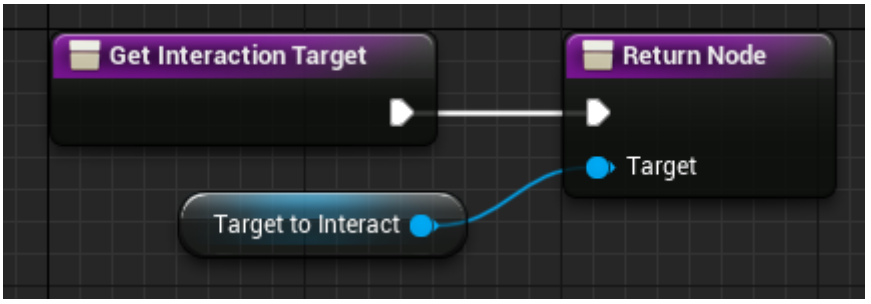


Add a such check to movement logic for both axis



Add Interface

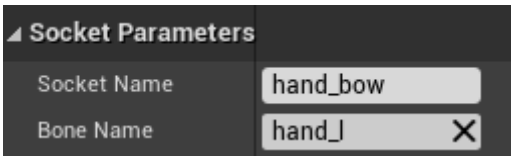




Player Character Skeleton

For demo content to work you need to add weapon sockets to your player and AI character skeletons. Open my **UE4_Mannequin_Skeleton** and find the next sockets.

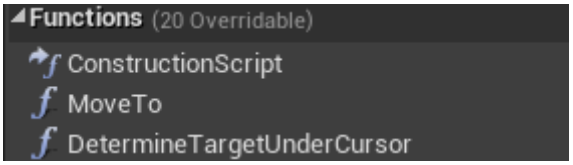
In details panel for each socket you can find bone name to which you need to attach sockets.



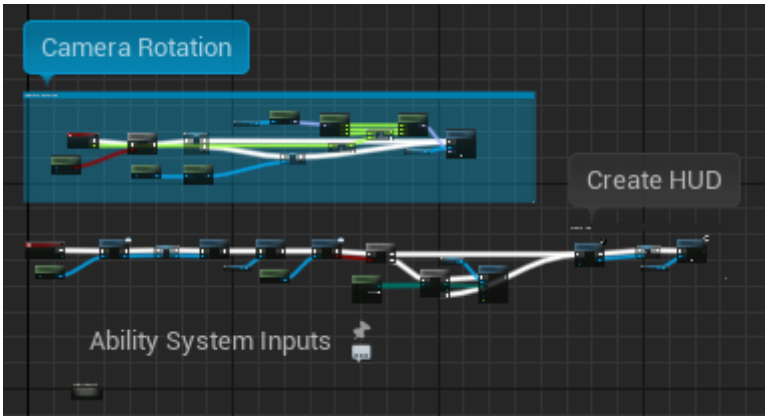
Player Controller Top Down

Add **QuickSlot Manager**

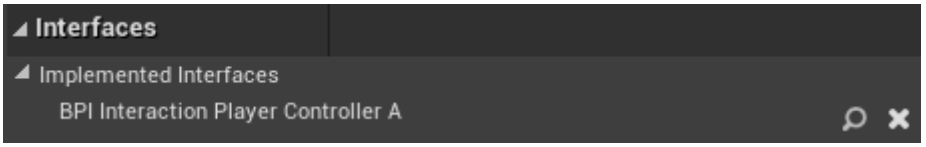
Create the next functions and copy logic from my controller

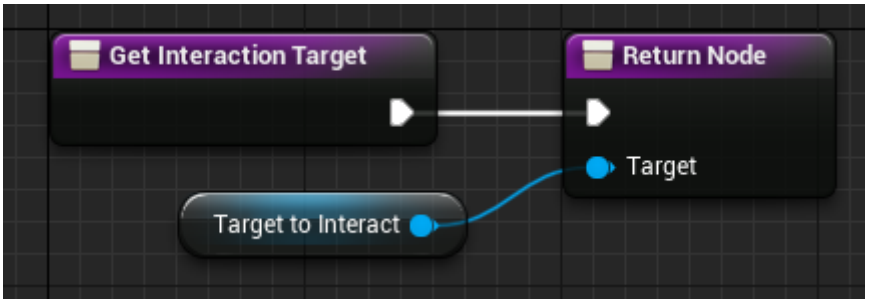


Copy the next logic to your player controller.
Note that I'm creating HUD widget in player controller.
Also note that some inputs can be the same as inputs in your project.



Add Interface

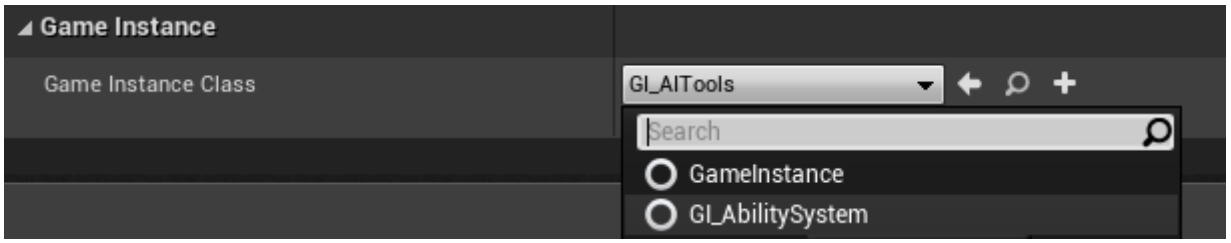




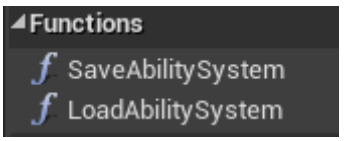
Save and Load

I use Save Game and Game Instance classes.

If you don’t have such classes, you can use my ones.
Open Project Settings -> Maps and Mode -> Set my Game Instance class

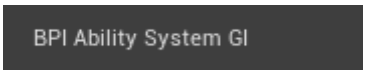


If you use your own classes, I suppose you already know how it works, and you understand that you need to copy all logic from my classes to your classes.
For **Save Game** class, create the next functions and copy logic.



Create the next variable. It is used to determine what mode (Third person or top down) to load.

For **Game instance** class, add interface and copy all logic for each function



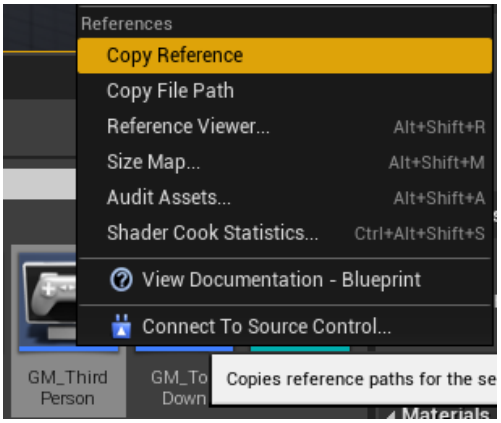
Copy logic from Event Graph

And copy logic from functions

Game Mode

If you want to use 2 modes (Third Person and Top Down) and switch between them at runtime, you need to set level options in some places.
Find Game Mode classes.

To get reference click RMB and select the appropriate option.



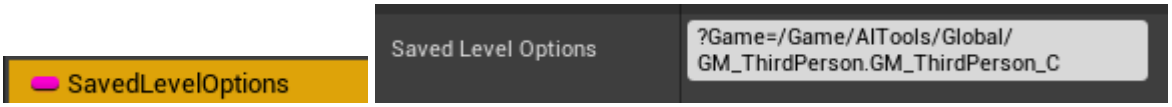
You will get this

Blueprint'/Game/AITools/Global/GM_ThirdPerson.GM_ThirdPerson

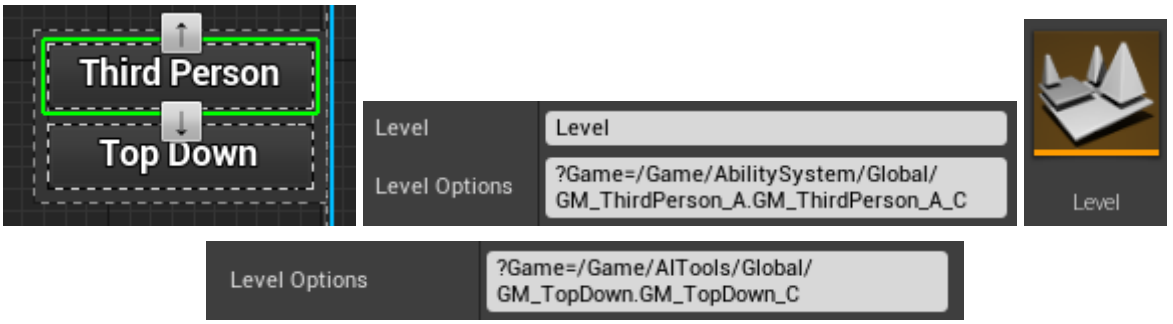
Replace **Blueprint'** on **?Game=** and add **_C** to the end
You will get this

?Game=/Game/AITools/Global/GM_ThirdPerson.GM_ThirdPerson**_C**

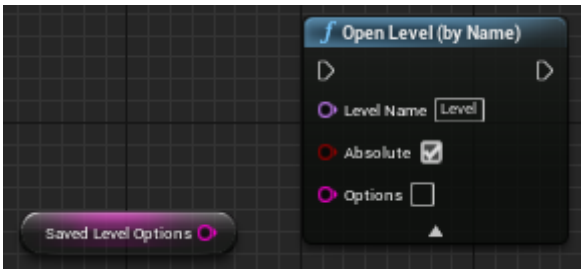
Open **Game Instance** class, select the next variable and set default value.



Open **HUD** widget, select each button and set the appropriate options in details panel.

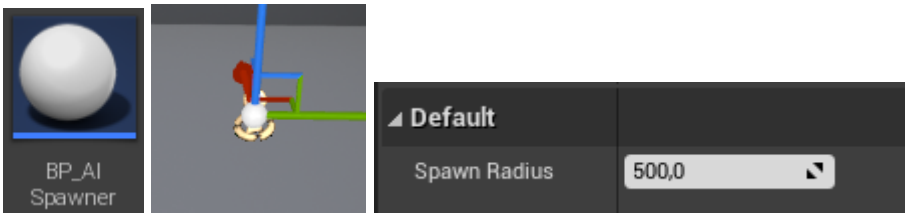


But most likely you will use only one mode, and in this case you need just and disconnect LevelOptions from input. And make sure you set correct level name.



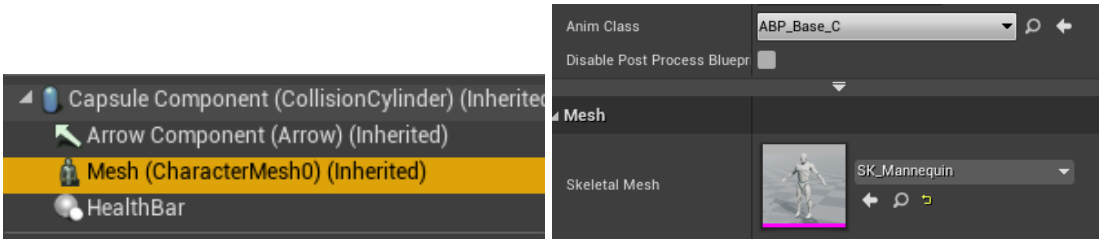
AI

If you want to use my AI you need just place the spawner on the scene and set spawn radius. Don't forget the place NavMeshBoundsVolume.

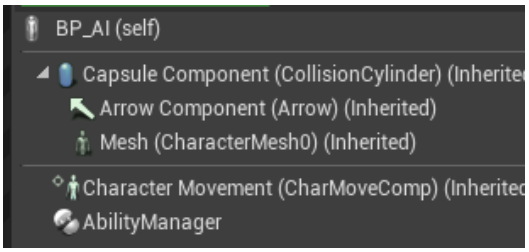


If you want to use your own AI class you need to do the following steps.
For example you have the next AI class

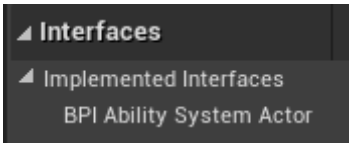
I set my character model since my equipped weapon changes anim instances that use this skeleton.



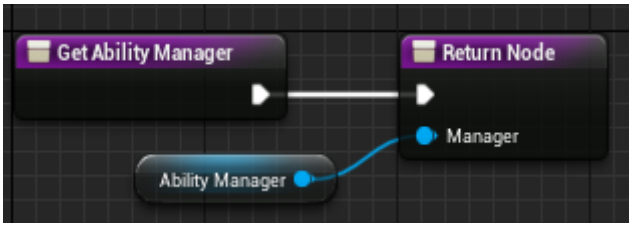
Add **Ability Manager**



Add interface



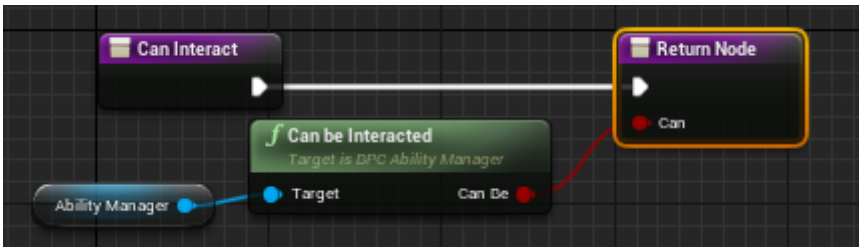
Implement **GetAbilityManager** function



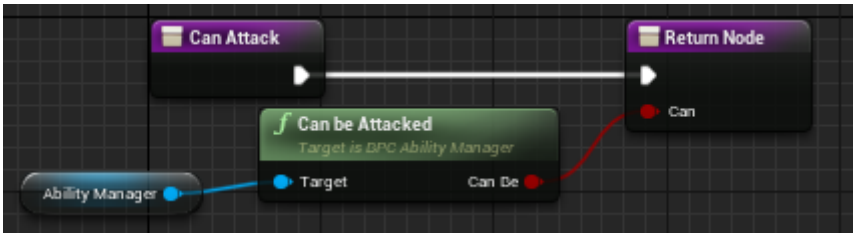
Copy variables from BP_AI_A and implement **GetStartData** function

Implement **GetInteractionDistance** function. I set 250 for all AI.

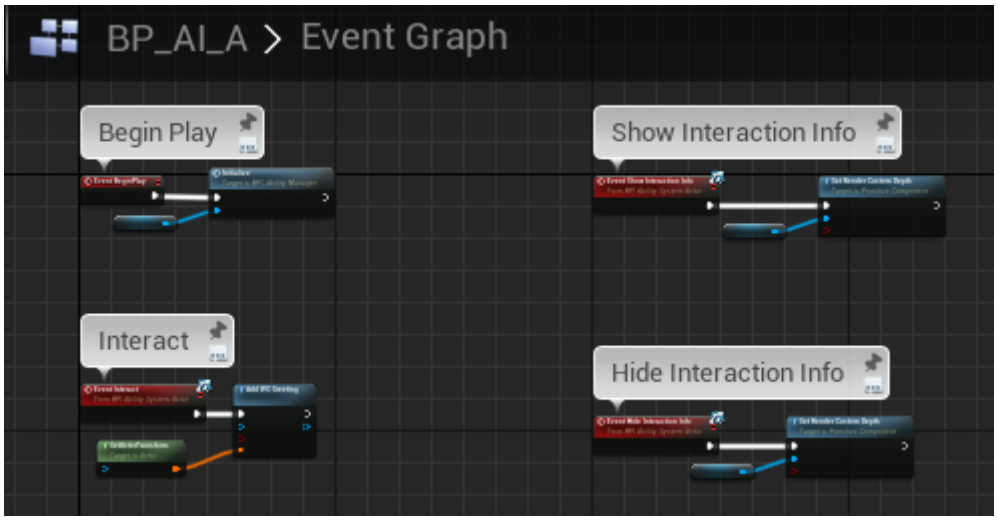
Implement **CanInteract** function.



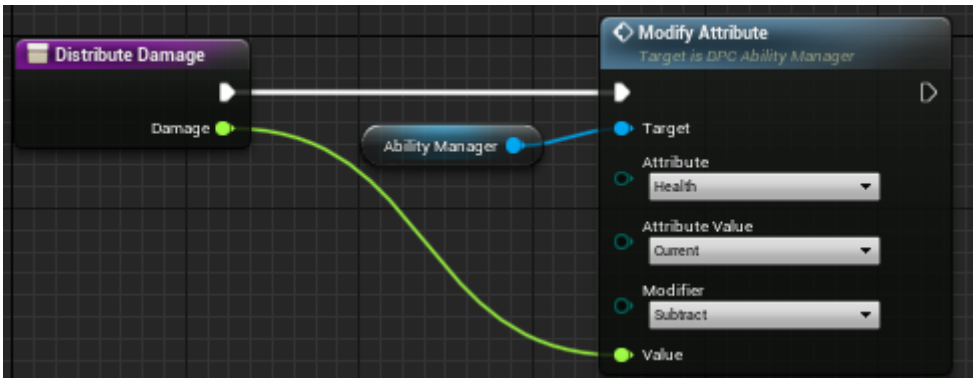
Implement **CanAttack** function.



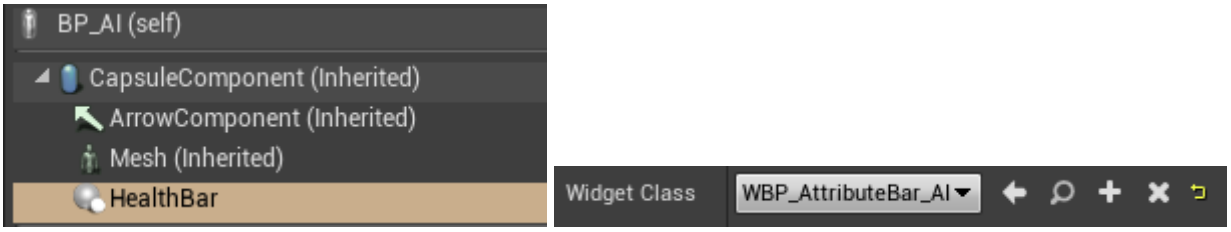
Copy logic from **BP_AI_A**



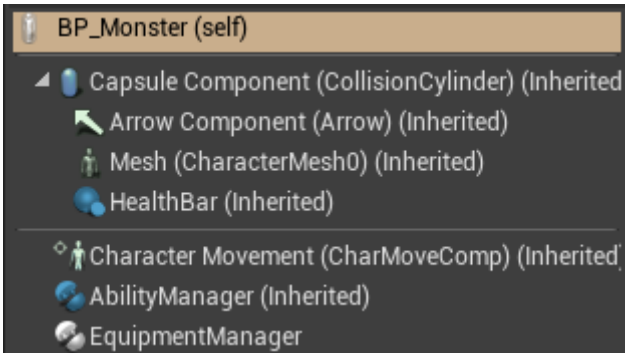
Create **DistributeDamage** function.



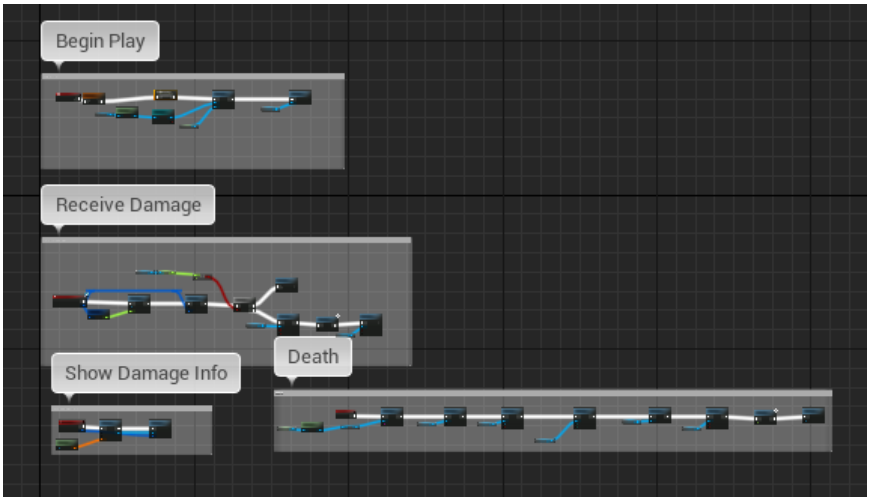
Add widget component or copy from my **BP_AI_A**



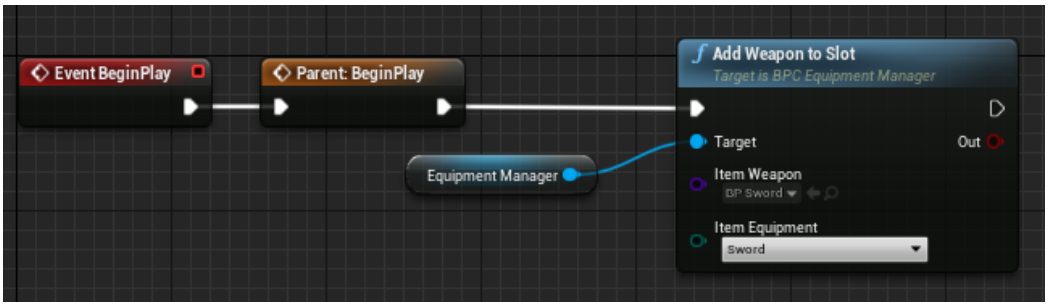
Then we will copy logic for Monster, so if you want you can create child class BP_Monster.
Add **Equipment Manager** to monster class with no settings.



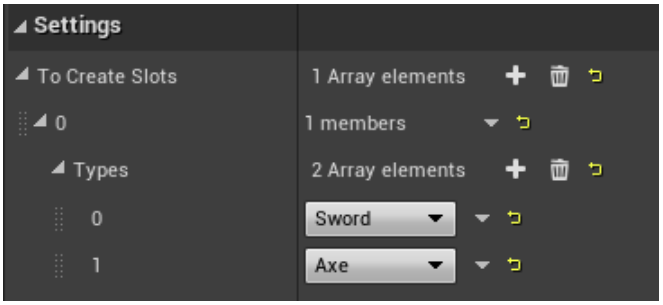
Copy logic for monster from BP_Monster_A



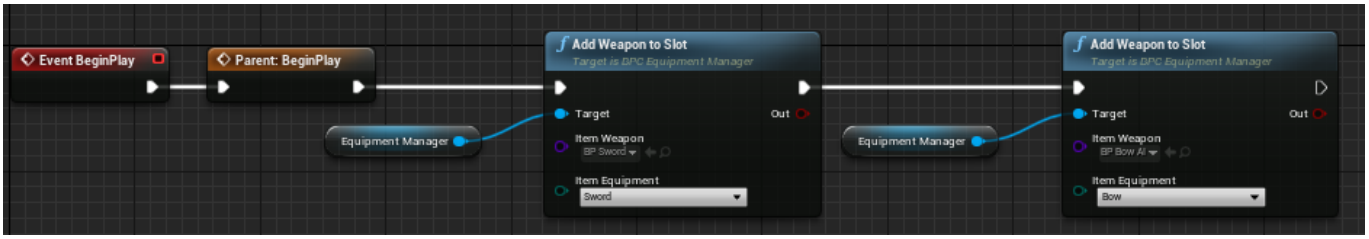
Then you can create child class BP_Warrior and override **Begin Play** event for Warrior



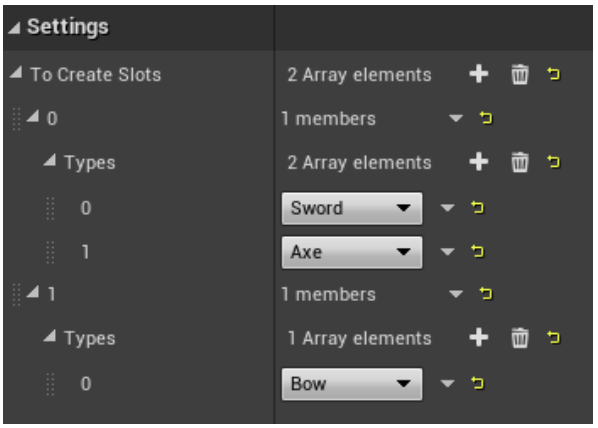
Equipment Manager settings for Warrior



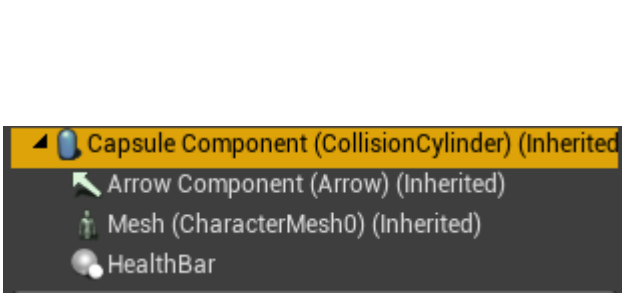
Then you can create child class and override **Begin Play** for Warrior&Archer

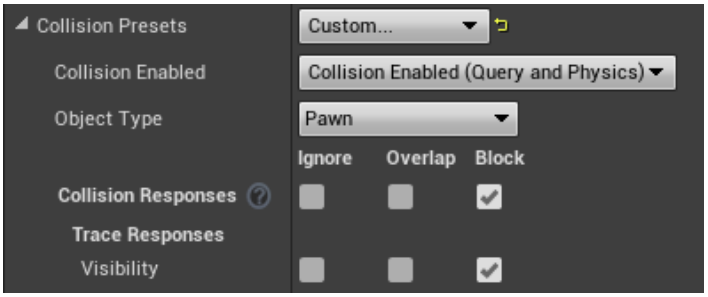


Equipment Manager settings for Warrior&Archer

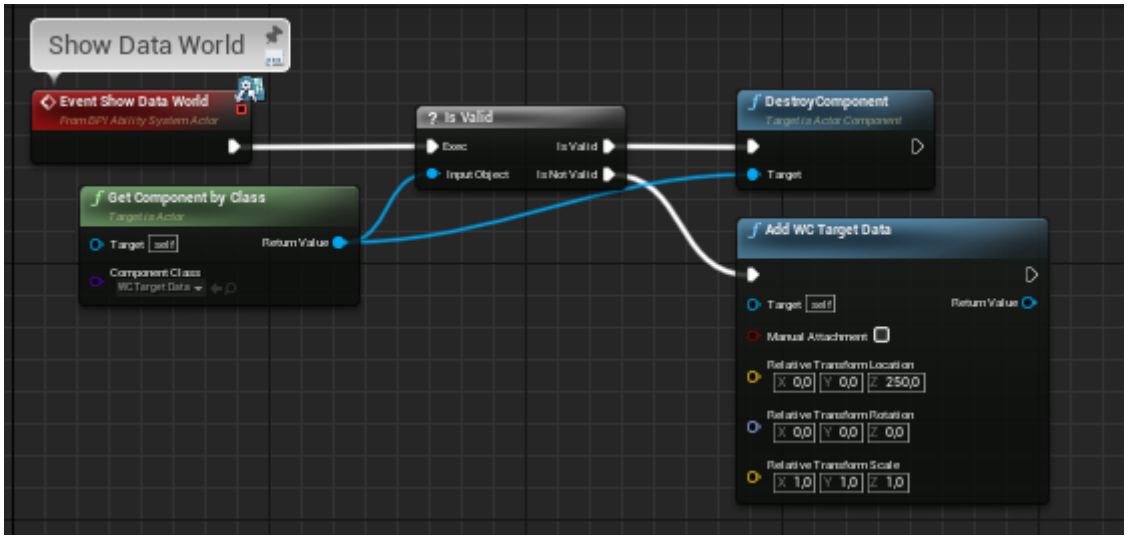


For visual debugging to work, make sure that character capsule collision blocks traces by visibility.





If your AI and Player character classes haven't the same parent class, copy this logic to AI class



If you will have any questions, please, let me know about it via Discord or email. Thanks!