Projet HackNSlash by Noah MAZARD

For this Project, all the c++ code is located in the Source folder and all my Blueprints under Content/HackNSlash/ (Except for the Player character and controller in Content/TopDown/)
All the main algorithm is done in C++ and all the visuals (animations, particles, etc.) are done in BP, as well as the level logic (can be made by the level designer, for example the quests implementation).

Character Movements (C++ : *HackNSlashPlayerCharacter*)

You move with ***ZQSD***.
The character will rotate toward the ***mouse*** and look at it.

Movements are computed using a navigation mesh where the destination of the character is based on his inputs.

Movements Animations (BP : *CustomAnimations/*)

I use characters from the assets pack “Polygon Dungeon” from Synty studio. Those characters don't have animation so I used one from a Paragon character. To do that, I use 2 animation blueprints (one for the Paragon character and one for the Synty character).
The first one (*ABP_TopDown*), will compute all the animations and the second one (*ABP_SyntyUE4*) will retarget them from the UE4 compatible mannequin to the synty skeletonMesh using a IK Retargeter (*RTG_SyntyUE4*).

Skill System (C++ : *Skill[...]* and Blueprints : *Skills/*)

To use the different skills : ***Left Click***, ***Right Click*** and ***E***.
The 3 skills implemented here for the Player are :
-A melee attack (*BP_MeleeAttack*) : Hit your enemies with your weapon.
-A range attack (*BP_SkillRange*) : Throw a projectile toward your mouse location.
-A shield spell (*BP_ShieldSkill*) : Summon Shields and reduce damage received by 75%.

This skill system works with a *SkillManagerComponent* on the character that will be using them. In this component, there is an Array of the skill the character can use. This system is made to easily create and use new skills. To do that, create a new class that inherits *SkillBase*, override the Execute() function and you can add it to the skill array and test it.

For the melee attack, I spawn a *SkillACollisionDetection*, an actor with collision that will call an event on my skill when being overlapped. Same for the range one, with a *SkillALinearProjectile* that implements a projectileMovementComponent.
When we receive this event, we call ApplyDamage on this actor.

In the blueprints implementation of the skill, we play an animation montage and spawn visual effects if needed.

Life System(C++ : *LifeComponent* and BP : *LifeSystem/*)

For the character's life points, I've made a *LifeComponent* that is bound to the OnTakeAnyDamage of his attached actor. The BP version also creates a LifeBar widget and adds it to the parent so we can see the character's life points.

Artificial Intelligence (C++ and BP : *AI/)*

The enemies characters inherit from *BP_EnemyCharacter* that himself inherits from the PlayerCharacter. This enemy is controlled by an *EnemyAICharacter* that will run a Behavior Tree (*BT_Enemy*) along with an AIPerceptionComponent that will set values in the tree's BlackBoard. When the enemy sees the player, it will run towards it and if it is near enough, try to hit it with a random skill in his skillManager skill array. Here the enemy only has a melee skill that is less powerful than the player one (*BP_EnemyMeleeAttack*). When it does not see him anymore, it will go to the last position it saw him and, after 2 sec, will go to his starting position.

Again, you can easily create new enemies by creating a child class of *BP_EnemyCharacter*, change his mesh and his skill and you get a new enemy.

Quest System (C++ : *Quest[...]* and BP : *QuestSystem/*)

The quest system is also meant to easily make new quests and use them.
In the *HackNSlashGameInstance* we stock the current quest. From where you want (Here the level BP) you call StartQuest(QuestClass) on this GameInstance and this will start the quest you specified.

Each quest inherits from *QuestBase*, has some FString getters used by the UI, an InitQuest(), an EndQuest() and their BP versions.

I have made two quest type :
-*QuestLocation*, work with a *QuestLocationComponent* attached to an actor in the level that will trigger the end of this quest.
-*QuestKill*, kill a certain number of enemies from a certain class.

To create a quest, you need to create a BP version of a quest, fill the variables or make a custom one from QuestBase by implementing InitQuest(), EndQuest() and the UI Getters.

You can also start a new quest when the current is finished (in the BP_EndQuest of the finished one).

User Interfaces (BP : *UI/*)

I have implemented 3 major type of UI :
-A basic main menu with a button for start or quit the game with a moving background because I found it cool.
-A pause menu when you press escape (or the space bar in editor) that pause the game, with a button for resume, restart and return to the main menu.
A loose and a Win Screen, that are just child classes of the pause menu with another title and a removed Resume Button.
-An in-game HUD : with an icon and cooldown representation for each skill and a display of the current quest.