

Annexes de projet d'ingénierie informatique

Aménagement d'intérieur en réalité virtuelle

Auteurs :
Sacha Malray, Antoine Patoux, Alex Thayse

Cours :
Projet d'ingénierie informatique

Année académique :
2024-2025

Professeur :
Mohammed Benjelloun

Promoteur :
Mathis Delehouzée

Table des matières

A Aperçu des Actors pour l'éclairage artificiel	5
B Aperçu des Actions possibles et HUD	7
B.1 Implémentation	12
B.1.1 Actors	12
B.1.2 Game Instance	17
B.1.3 Save Game	21
B.1.4 Structures	21
B.1.5 User Widgets	21
B.1.6 InputMapping	33
B.1.7 VRPawn	33
C Fonctions BluePrint C++	45

Table des figures

1	Vue de viewport de BP_Weather_Sunny	5
2	Vue de viewport de BP_Weather_Rainy	5
3	Vue de viewport de BP_Weather_Snow	6
4	Vue de viewport de BP_Weather_Sunset	6
5	Vue de viewport de BP_Weather_Night	6
6	Actions - Mode Normal	7
7	Actions - Mode Placing	7
8	Actions - Mode Lighting	7
9	Actions - Mode Texture	8
10	Actions - Mode Moving	8
11	Actions - Mode Rotating	8
12	Actions - Mode Scaling	9
13	Actions - Mode Deleting	9
14	HUD - Mode Normal	9
15	HUD - Mode Placing	10
16	HUD Mode Lighting	10
17	HUD - Mode Texture	10
18	HUD - Mode Moving	11
19	HUD - Mode Rotating	11
20	HUD - Mode Scaling	11
21	HUD - Mode Deleting	12
22	Event graph de BP_WeatherManager	13
23	Fonction ChangeWeather de BP_WeatherManager	13
24	Variables de BuildActor	13
25	Fonction Place de BuildActor	14
26	Fonction CheckSpawn de BuildActor	14
27	Fonction Cancel de BuildActor	14
28	Variable de LightActor	15
29	Fonction GetIntensity de LightActor	15
30	Fonction ChangeIntensity de LightActor	15
31	Fonction ChangeColor de LightActor	16
32	Variable de DoorActor	16
33	Event Graph de DoorActor	17
34	Timeline DoorSpin de DoorActor	17
35	Event Graph de BP_GameInstance	18
36	Fonction SavePlayerData de BP_GameInstance - Partie 1	18
37	Fonction SavePlayerData de BP_GameInstance - Partie 2	19
38	Fonction SavePlayerData de BP_GameInstance - Partie 3	19
39	Fonction LoadPlayerData de BP_GameInstance - Partie 1	20
40	Fonction LoadPlayerData de BP_GameInstance - Partie 2	20
41	Fonction RetrieveSaveData de BP_GameInstance	20
42	Fonction ClearPlayerData de BP_GameInstance	20
43	Fonction Autosave de BP_GameInstance	21
44	Variables de WB_MenuInteractif	21
45	Graphe de WB_MenuInteractif	22
46	Variables de WB_TypeOfLighting	23
47	Graphe de WB_TypeOfLighting	23
48	Variables de WB_NaturalLight	23

49	Graphe de WB_NaturalLight - Choix de la combobox	23
50	Graphe de WB_NaturalLight - On Selection Changed	24
51	Graphe de WB_NaturalLight - BTN_Confirm et BTN_Back	24
52	Variables de WB_TypeOfPlacing	25
53	Graphe de WB_TypeOfPlacing	25
54	Event Graph de WB_SaveSlot	26
55	Fonction GetSlotIndex de WB_SaveSlot	27
56	Fonction GetSaveDate de WB_SaveSlot	27
57	Fonction UpdateText de WB_SaveSlot	27
58	Event Graph de WB_SaveMenu	28
59	Event Graph de WB_ClearSlotConfirmation	29
60	Logique de suppression de sauvegarde dans WBS_SwitcherLobbyMenu	29
61	Exemple de logique de widget switcher	30
62	Exemple de logique d'apparition du nom du Mode Normal	30
63	Custom_Event_SucessfullSave	31
64	Logique de défilement	31
65	Exemple d'un bouton de WBS_SwitcherControllerDisplay	32
66	Fonction ResetPage de WBS_SwitcherControllerDisplay	32
67	ChangeHUD	32
68	Changemode	33
69	InputMapping	33
70	Save Logic	34
71	Affichage du menu sur la main	34
72	Logique du déplacement d'un objet lorsqu'il est attrapé	35
73	Event PlacingPreview	35
74	Event ChangeMode	35
75	Joystick gauche	36
76	Joystick droit - Rotation caméra	36
77	Joystick droit - Téléportation	36
78	Bouton X - Mode Placement	37
79	Bouton X - Mode Déplacement	37
80	Bouton X - Mode Rotation	37
81	Bouton X - Mode Mise à l'échelle	38
82	Bouton Y	38
83	Bouton A - Mode Placement	38
84	Bouton A - Mode Lumière	39
85	Bouton A - Mode Normal	39
86	Bouton A - Mode Déplacement	40
87	Bouton A - Mode Rotation	40
88	Bouton A - Mode Mise a l'échelle	41
89	Trigger gauche	41
90	Trigger droit - Mode Placement	42
91	Trigger droit - Mode Lumières	42
92	Trigger droit - Mode Suppression	42
93	Logique du changement de texture	42
94	Exemple de la logique de Grip gauche	43
95	Grip droit mode Normal, Déplacement, Rotation et Mise à l'échelle	43
96	Fonction GrabObject - Partie 1	43
97	Fonction GrabObject - Partie 2	44
98	Fonction ReleaseObject - Partie 1	44

99	Fonction ReleaseObject - Partie 2	44
100	Grip droit - Mode Placement	45
101	Fonction GetFBXAndOBJFilesInFolder	45
102	Fonction GetMaterialIndexFromHit	45
103	Fonction ImportScene	46
104	Fonction GetStaticMesh	47

A Aperçu des Actors pour l'éclairage artificiel

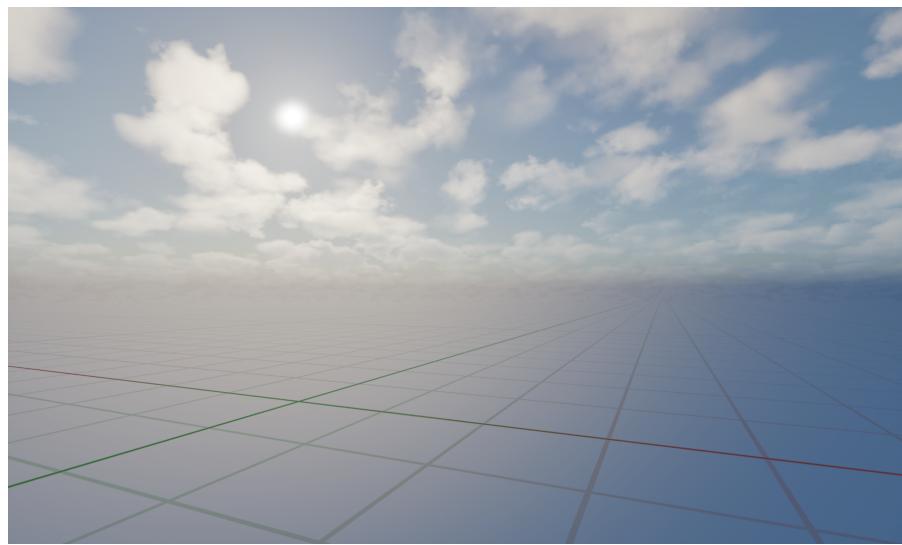


FIGURE 1 – Vue de viewport de BP_Weather_Sunny

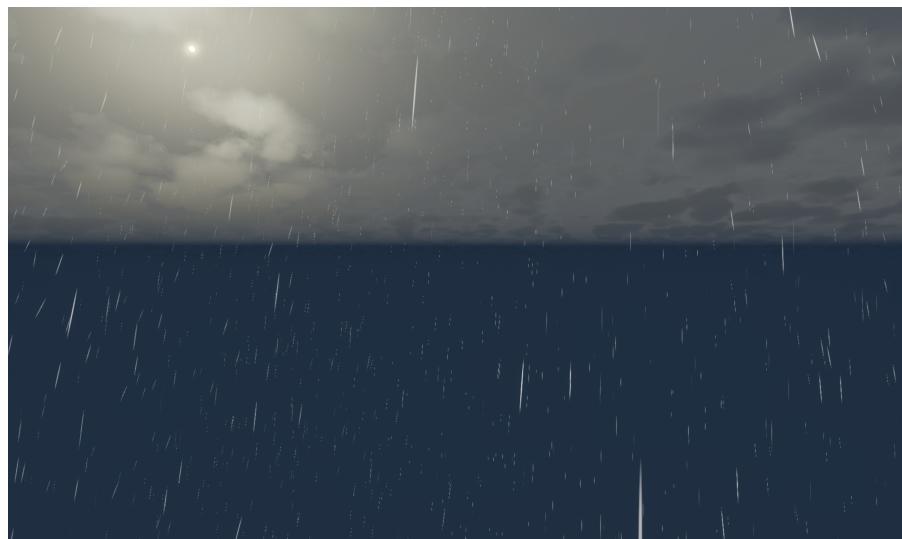


FIGURE 2 – Vue de viewport de BP_Weather_Rainy



FIGURE 3 – Vue de viewport de BP_Weather_Snow

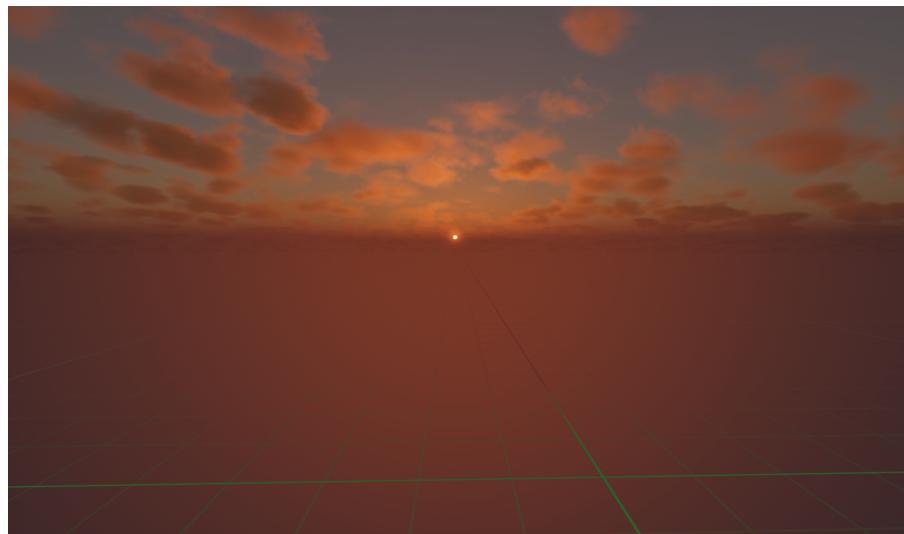


FIGURE 4 – Vue de viewport de BP_Weather_Sunset

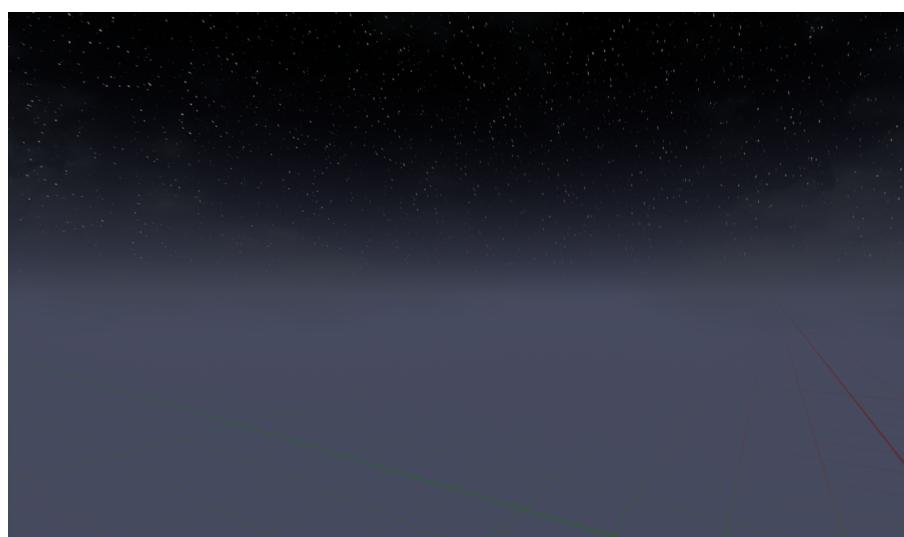


FIGURE 5 – Vue de viewport de BP_Weather_Night

B Aperçu des Actions possibles et HUD

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

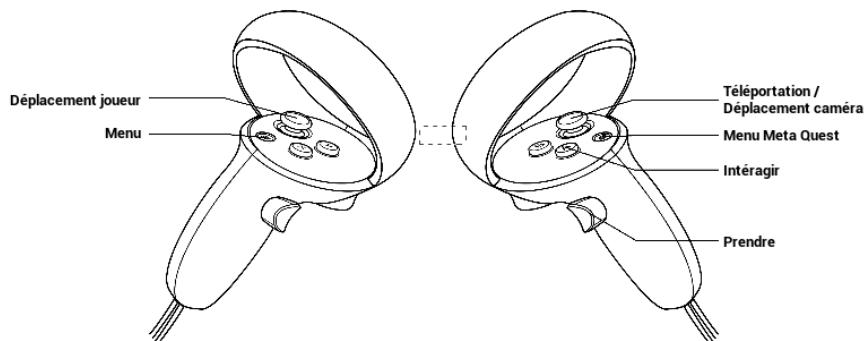


FIGURE 6 – Actions - Mode Normal

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

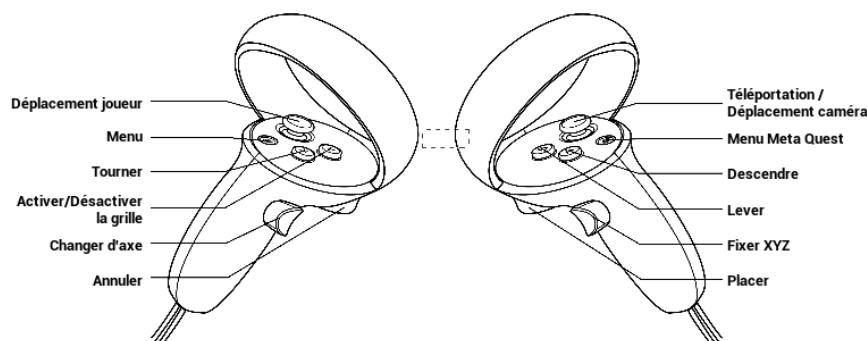


FIGURE 7 – Actions - Mode Placing

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

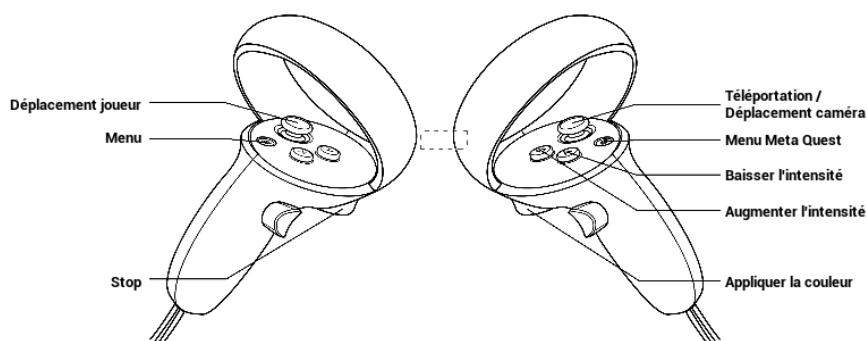


FIGURE 8 – Actions - Mode Lighting

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

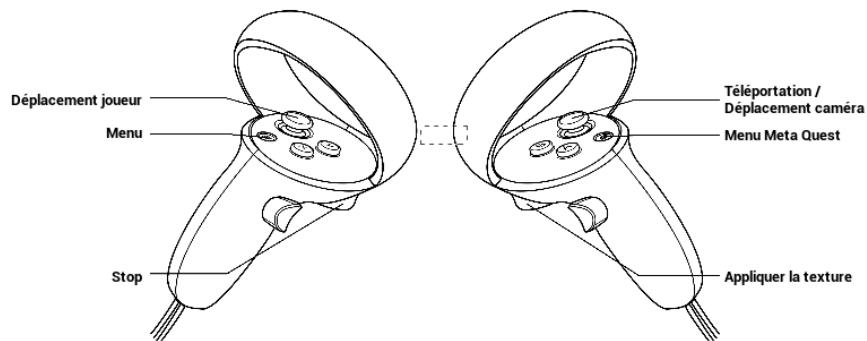


FIGURE 9 – Actions - Mode Texture

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

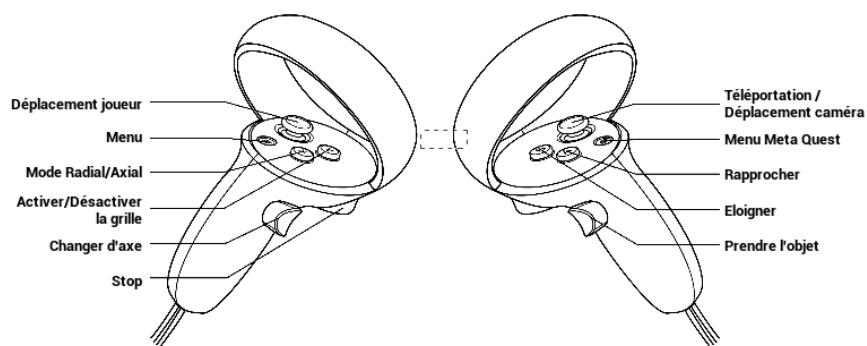


FIGURE 10 – Actions - Mode Moving

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

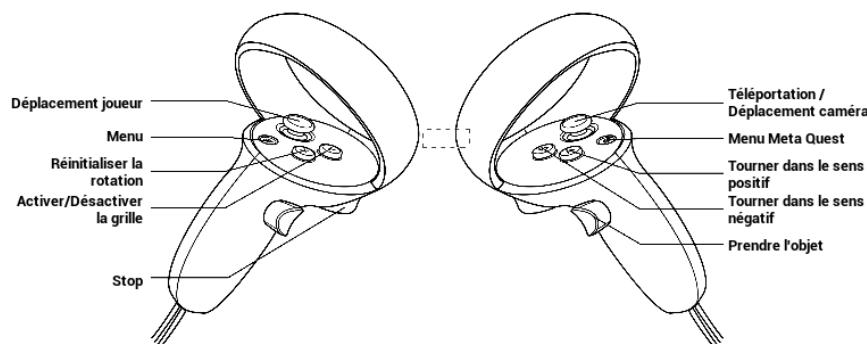


FIGURE 11 – Actions - Mode Rotating

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

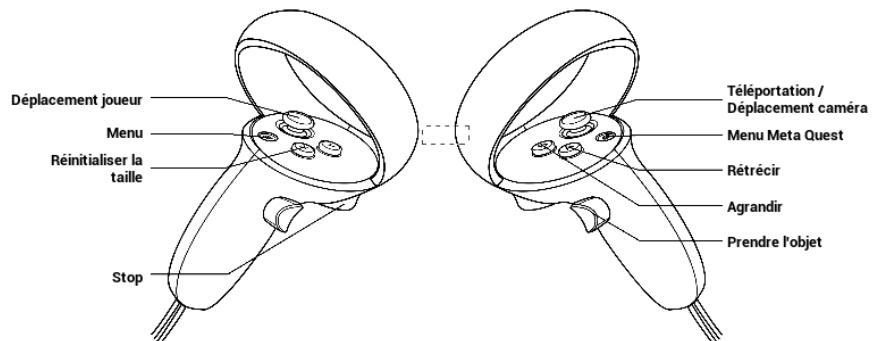


FIGURE 12 – Actions - Mode Scaling

Normal	Placing	Lighting	Texture	Moving	Rotating	Scaling	Deleting
--------	---------	----------	---------	--------	----------	---------	----------

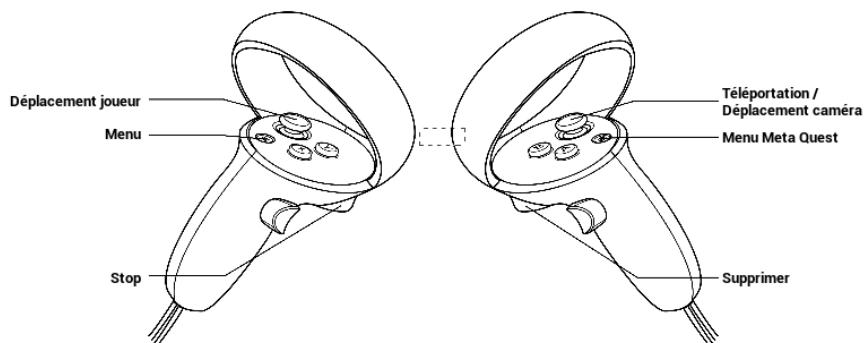


FIGURE 13 – Actions - Mode Deleting



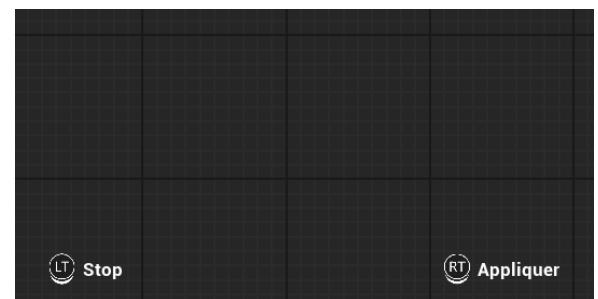
FIGURE 14 – HUD - Mode Normal



FIGURE 15 – HUD - Mode Placing



(a) HUD - Intensity



(b) HUD - Color

FIGURE 16 – HUD Mode Lighting



FIGURE 17 – HUD - Mode Texture

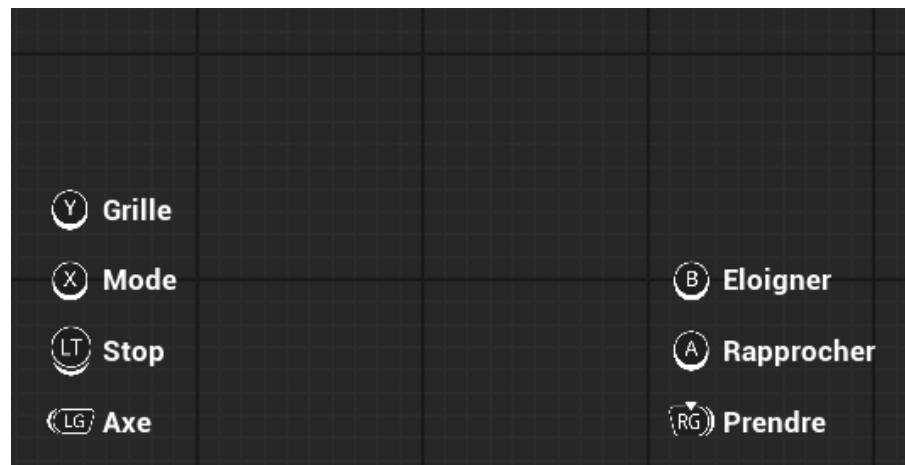


FIGURE 18 – HUD - Mode Moving



FIGURE 19 – HUD - Mode Rotating



FIGURE 20 – HUD - Mode Scaling



FIGURE 21 – HUD - Mode Deleting

B.1 Implémentation

B.1.1 Actors

BP_Weather_Sunny :

Cet Actor correspond à l'éclairage naturel "Ensoleillé". Ces composants sont : DirectionalLight, SkyLight, SkyAtmosphere, Volumetric Cloud et ExponentialHeightFog. Certains paramètres de ces composants ont été modifiés pour améliorer le réalisme. Pour ne pas alourdir ce rapport, nous vous invitons d'aller voir ceux-ci dans l'éditeur Unreal Engine 5.

BP_Weather_Rainy :

Cet Actor correspond à l'éclairage naturel "Pluvieux". Ces composants sont : DirectionalLight, SkyLight, SkyAtmosphere, Volumetric Cloud, ExponentialHeightFog et NiagaraSystem.Rain.

BP_Weather_Snow :

Cet Actor correspond à l'éclairage naturel "Neigeux". Ces composants sont : DirectionalLight, SkyLight, SkyAtmosphere, Volumetric Cloud, ExponentialHeightFog et NiagaraSystem.Snow.

BP_Weather_Sunset :

Cet Actor correspond à l'éclairage naturel "Coucher de soleil". Ces composants sont : DirectionalLight, SkyLight, SkyAtmosphere, Volumetric Cloud et ExponentialHeightFog.

BP_Weather_Night :

Cet Actor correspond à l'éclairage naturel "Coucher de soleil". Ces composants sont : DirectionalLight, SkyLight, BP_Sky_Sphere et ExponentialHeightFog.

BP_Sky_Sphere :

Cet Actor permet de créer l'environnement de nuit.

BP_WeatherManager :

Lorsque cet Actor est posé dans un level, il permet de faire apparaître BP_Weather_Sunny, dès le lancement de ce level. Il contient également ChangeWeather qui permet de changer d'Actor d'éclairage naturel. La variable CurrentWeather permet de sauvegarder la référence de l'Actor placé pour pouvoir le supprimer en cas de changement.

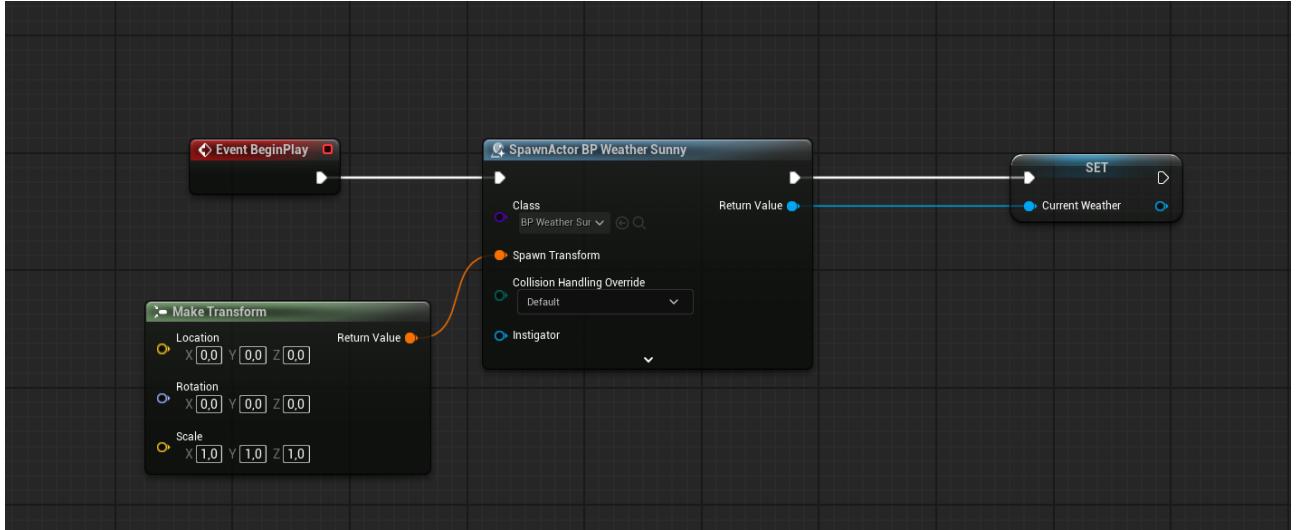


FIGURE 22 – Event graph de BP_WeatherManager

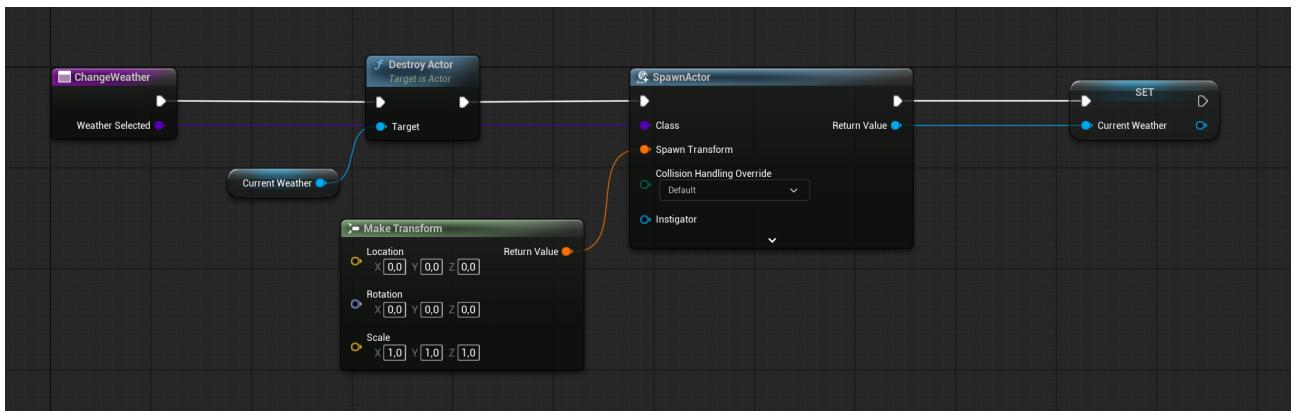


FIGURE 23 – Fonction ChangeWeather de BP_WeatherManager

BuildActor

Le BuildActor est simplement constitué d'un staticmeshcomponent. Il constitue l'ensemble des objets que l'utilisateur peut placer, modifier et supprimer.

Il possède deux variables, Un booléen, qui permet lors de la prévisualisation de placement de savoir si l'objet peut être placé ou non (en regardant la hitbox des acteurs qui se superposent). L'autre variable est simplement un array de materials, cela permet à l'objet d'être placé avec sa/ses texture(s) prévue(s).



FIGURE 24 – Variables de BuildActor

Il possède 3 fonctions qui permettent de le placer dans le niveau. La fonction CheckSpawn vérifie s'il ne se superpose pas avec un autre acteur. La fonction Place place le BuildActor à l'endroit spécifié. La fonction Cancel détruit simplement l'acteur.

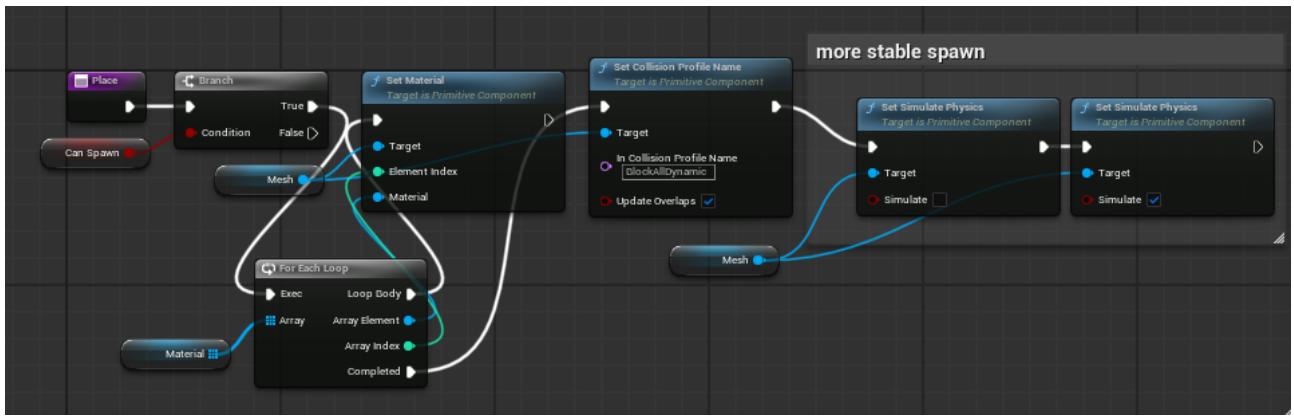


FIGURE 25 – Fonction Place de BuildActor

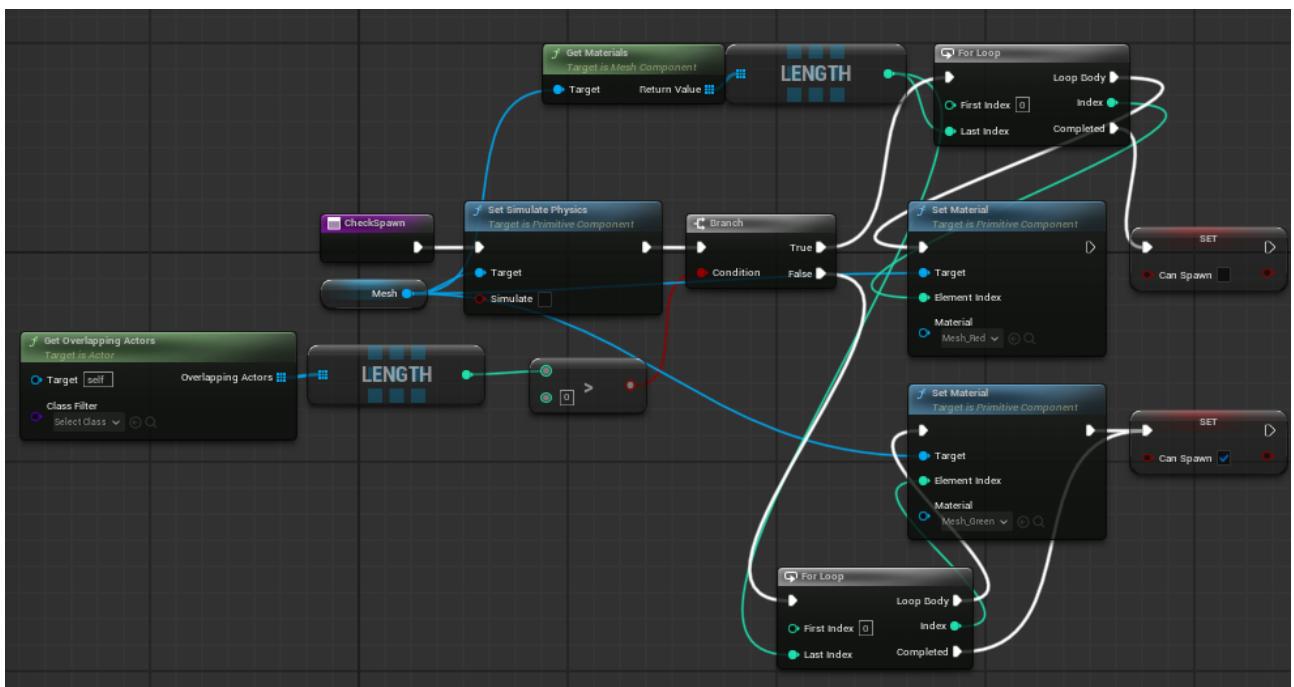


FIGURE 26 – Fonction CheckSpawn de BuildActor

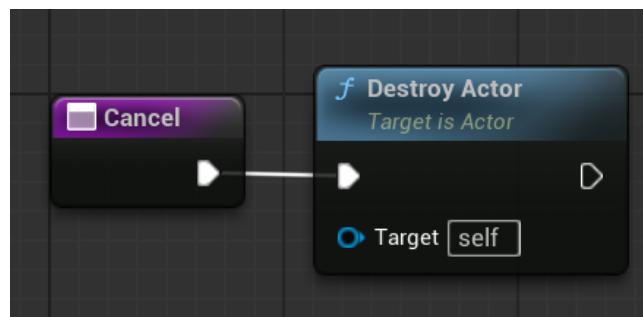


FIGURE 27 – Fonction Cancel de BuildActor

LightActor

Le LightActor, descendant de BuildActor, possède lui en plus d'un staticmeshcomponent, une PointLight. Il possède une seule variable, un float représentant l'intensité de la PointLight. Quoi ? Qu'est ce tu veux. Arrête de m'espionner tu vas faire quoi ? Jvais te trancher la

gorge je capte (comme l'antenne que je suis) Va check ta pp moodle btw Je te déteste. T'as fait ça quand ? Quand tu m'as chippé mon pc ah ouais je vois T'es vraiment un petit filou toi

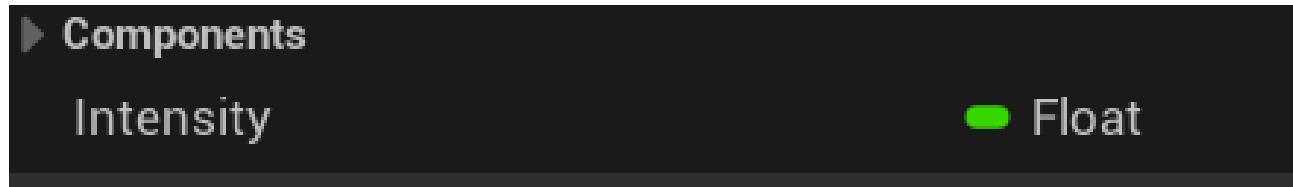


FIGURE 28 – Variable de LightActor

Il possède lui aussi 3 fonctions qui permettent de gérer différent paramètres de lumière. GetIntensity et ChangeIntensity pour l'intensité de la lumière et ChangeColor pour la couleur.

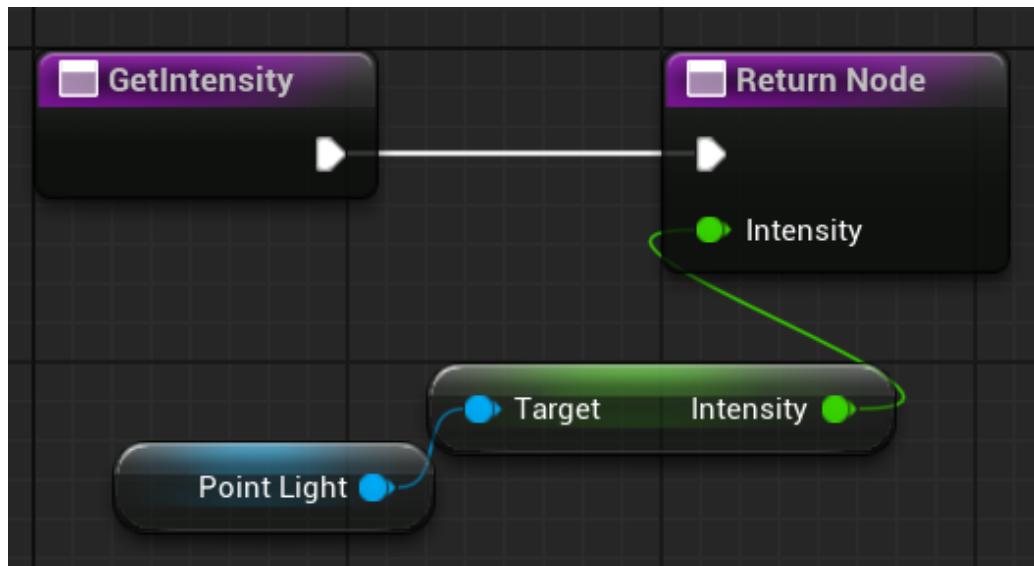


FIGURE 29 – Fonction GetIntensity de LightActor

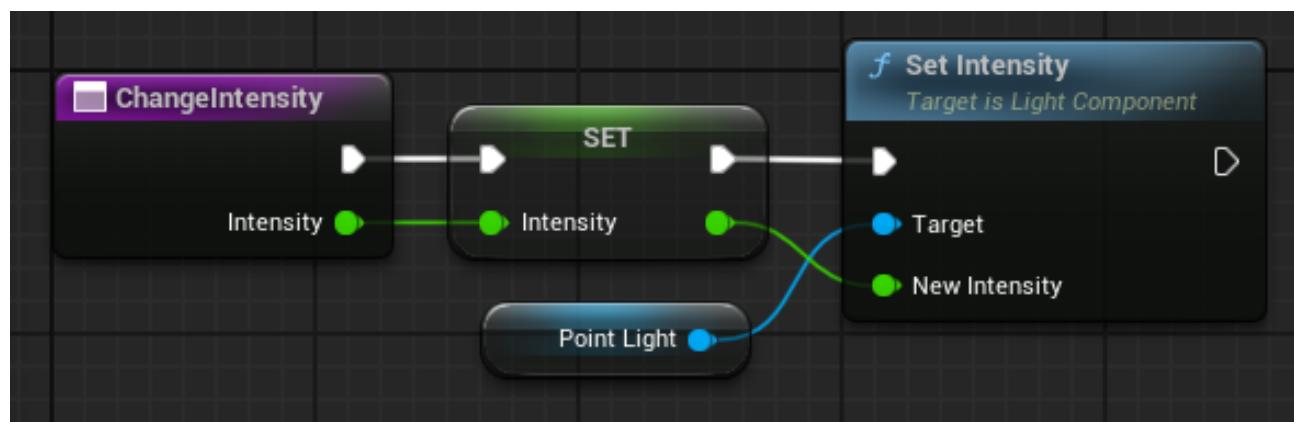


FIGURE 30 – Fonction ChangeIntensity de LightActor

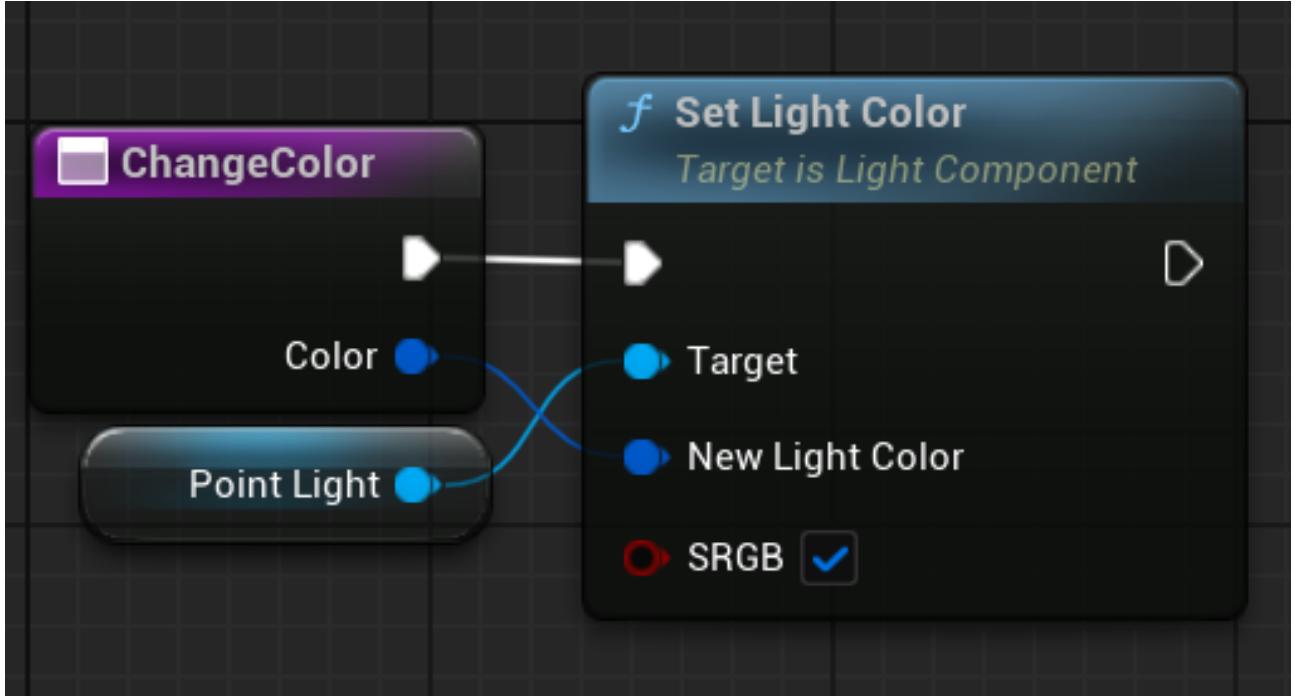


FIGURE 31 – Fonction ChangeColor de LightActor

DoorActor

Le DoorActor diffère du BuildActor en implémentant l'interface DoorInterface. Il possède une variable booléenne qui représente l'état de la porte (ouvert ou fermé).

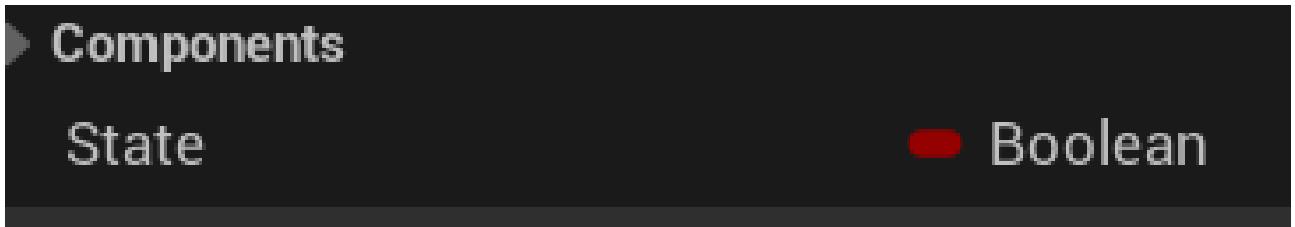


FIGURE 32 – Variable de DoorActor

L'interface possède une fonction interact. Cette fonction définie dans le DoorActor ouvre ou ferme la porte selon son état lors de l'appel de la fonction. Le DoorActor possède aussi un event OnSpawn qui est appelé quand le joueur charge sa partie.

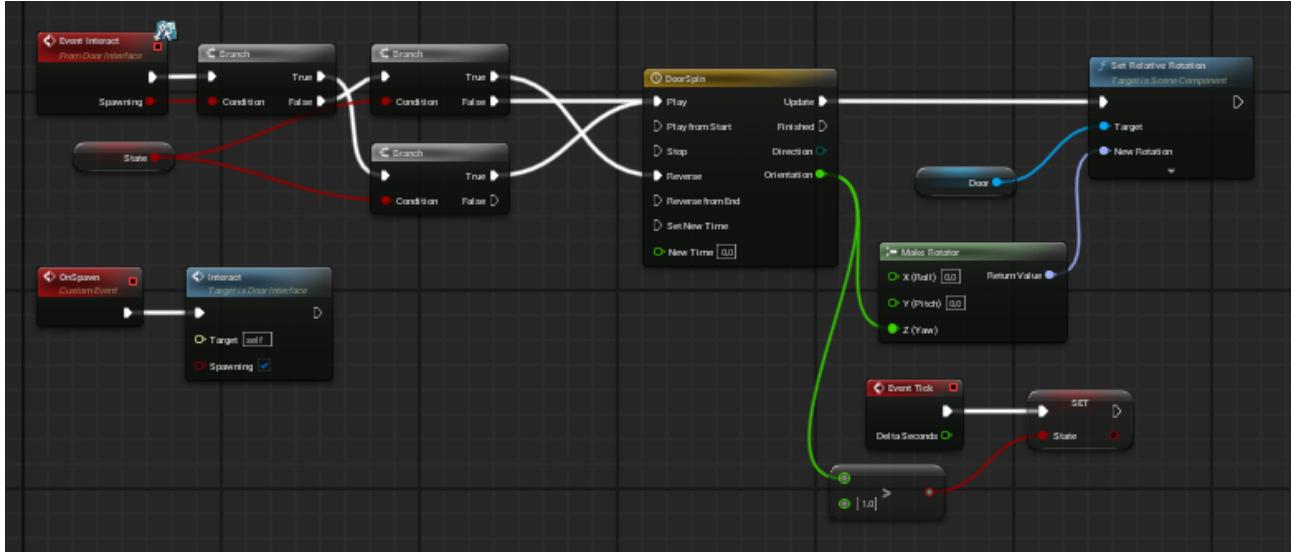


FIGURE 33 – Event Graph de DoorActor

Cette fonction utilise une Timeline pour l'animation de fermeture/ouverture pour un résultat plus naturel

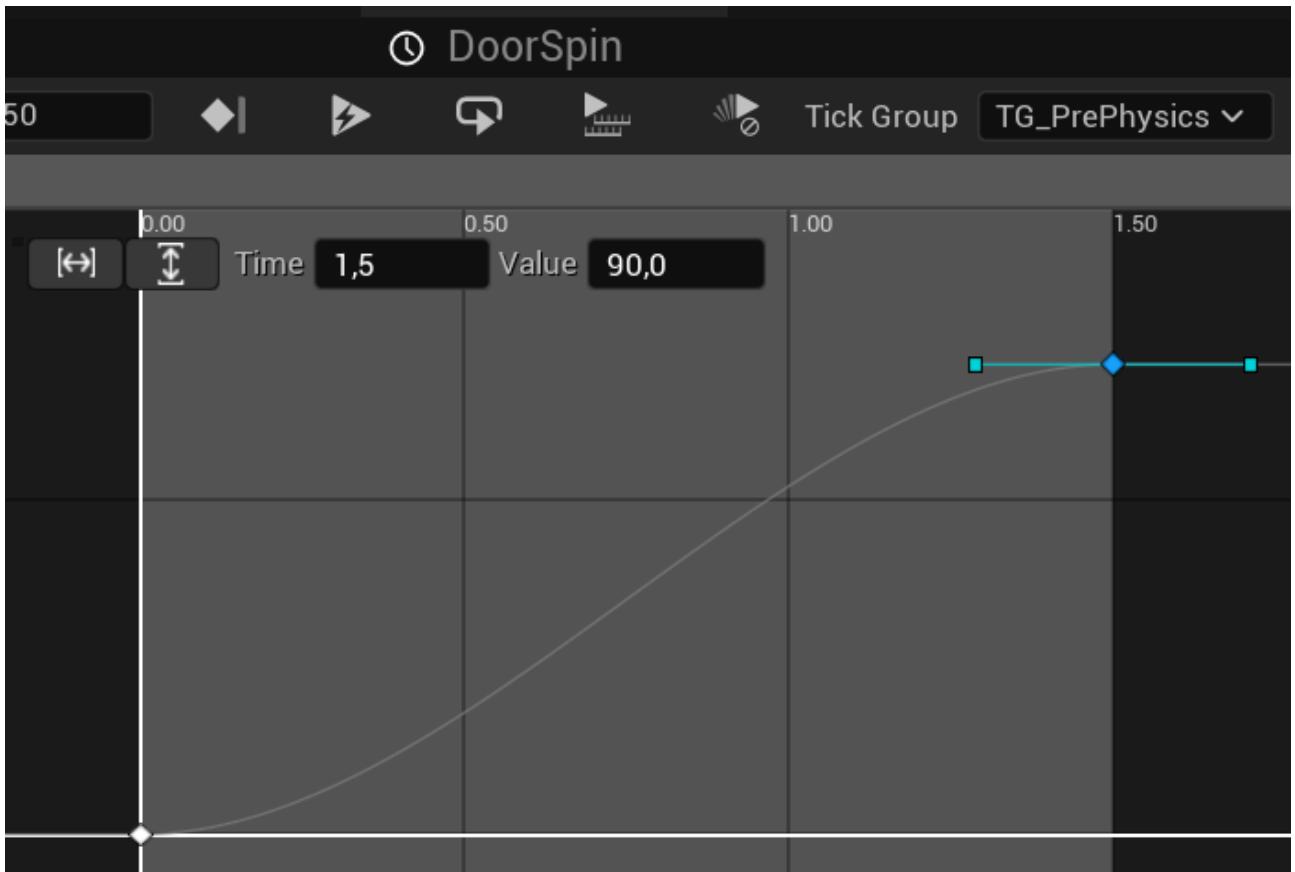


FIGURE 34 – Timeline DoorSpin de DoorActor

B.1.2 Game Instance

BP_GameInstance :

Dans l'Event Graph de BP_GameInstance, nous avons créé un événement "LoadGame". Une fois lancé, le système vérifie si une sauvegarde existe. Si la sauvegarde existe, alors elle

est chargée sinon une sauvegarde est créée. Set Timer by Function Name permet de lancer la fonction AutoSave toutes les 720 secondes. La fonction SavePlayerData stocke les informations dans un objet de BP_SaveGame. On stocke PlayerDataTransform en utilisant Get Actor Transform qui permet de retourner la position actuelle de notre personnage. Ensuite, nous sauvegardons les BuildActors, LightActors et DoorActors dans les 3 vecteurs correspondants. On identifie les classes des objets en vérifiant s'il implémente l'interface qui leur correspond. Ainsi, malgré que LightActor et DoorActor sont des classes filles de BuildActor, on détermine qu'un Actor est un LightActor s'il implémente LightInterface et est un DoorActor s'il implémente DoorInterface. Finalement, on récupère la date de la sauvegarde en utilisant le noeud Now. La fonction LoadPlayerData permet que, lors du chargement de la partie, le système aille rechercher ces informations pour les afficher et les utiliser dans le level. Au lancement, elle permet de déterminer la position du joueur avec Set Actor Transform et de faire apparaître les objets avec leurs caractéristiques grâce au noeud Spawn Actor. La fonction RetrieveSaveData permet de récupérer dans le BP_SaveGame la date pour pouvoir l'afficher dans le SaveSlot. La fonction ClearPlayerData permet de supprimer un BP_SaveGame. La fonction AutoSave permet de lancer la sauvegarde du système automatiquement.

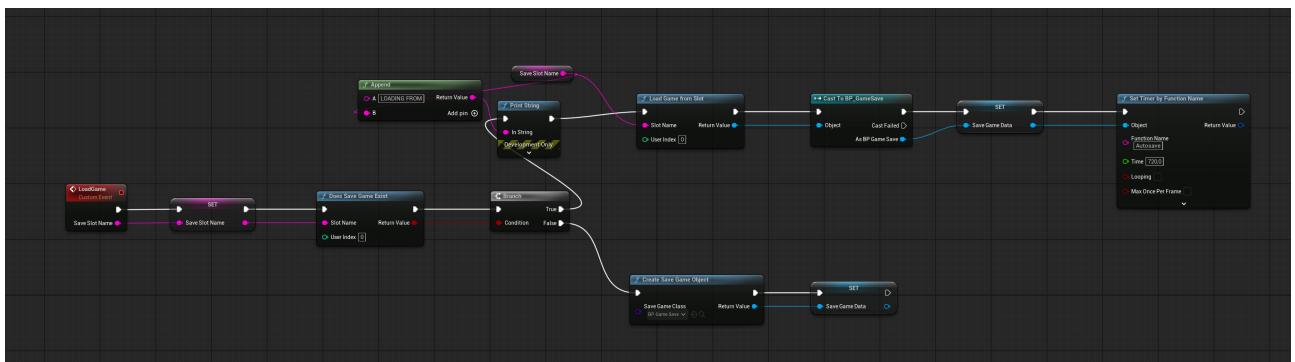


FIGURE 35 – Event Graph de BP_GameInstance

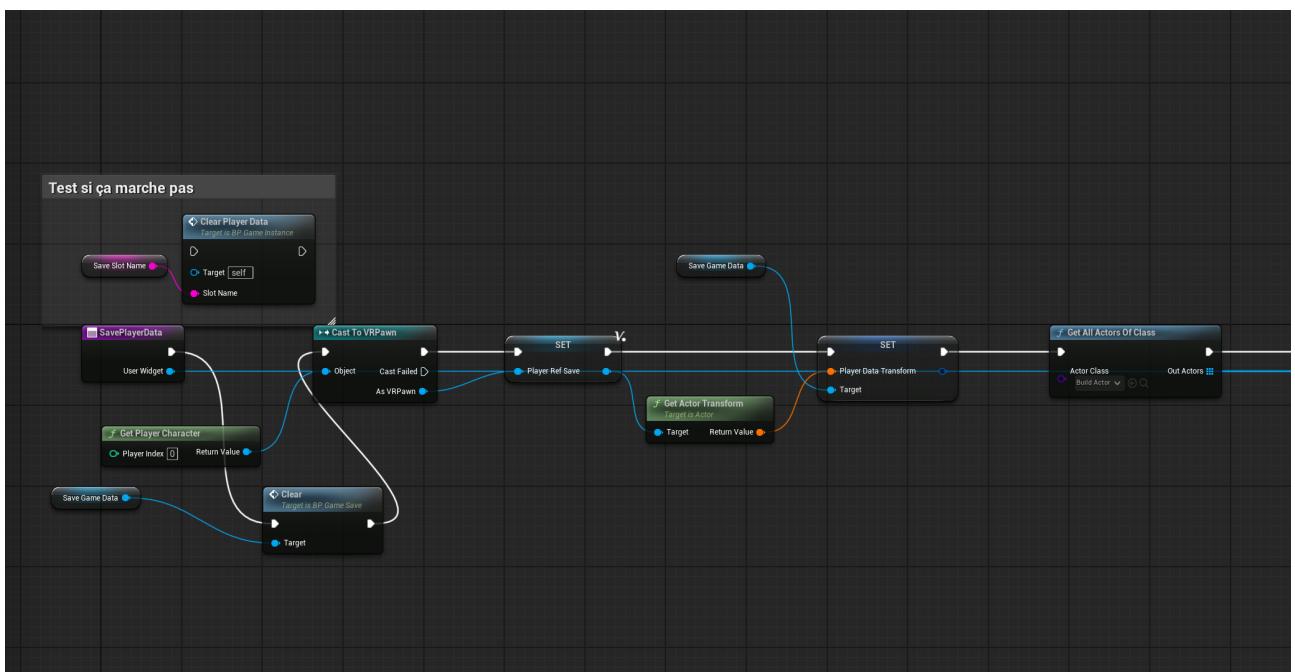


FIGURE 36 – Fonction SavePlayerData de BP_GameInstance - Partie 1

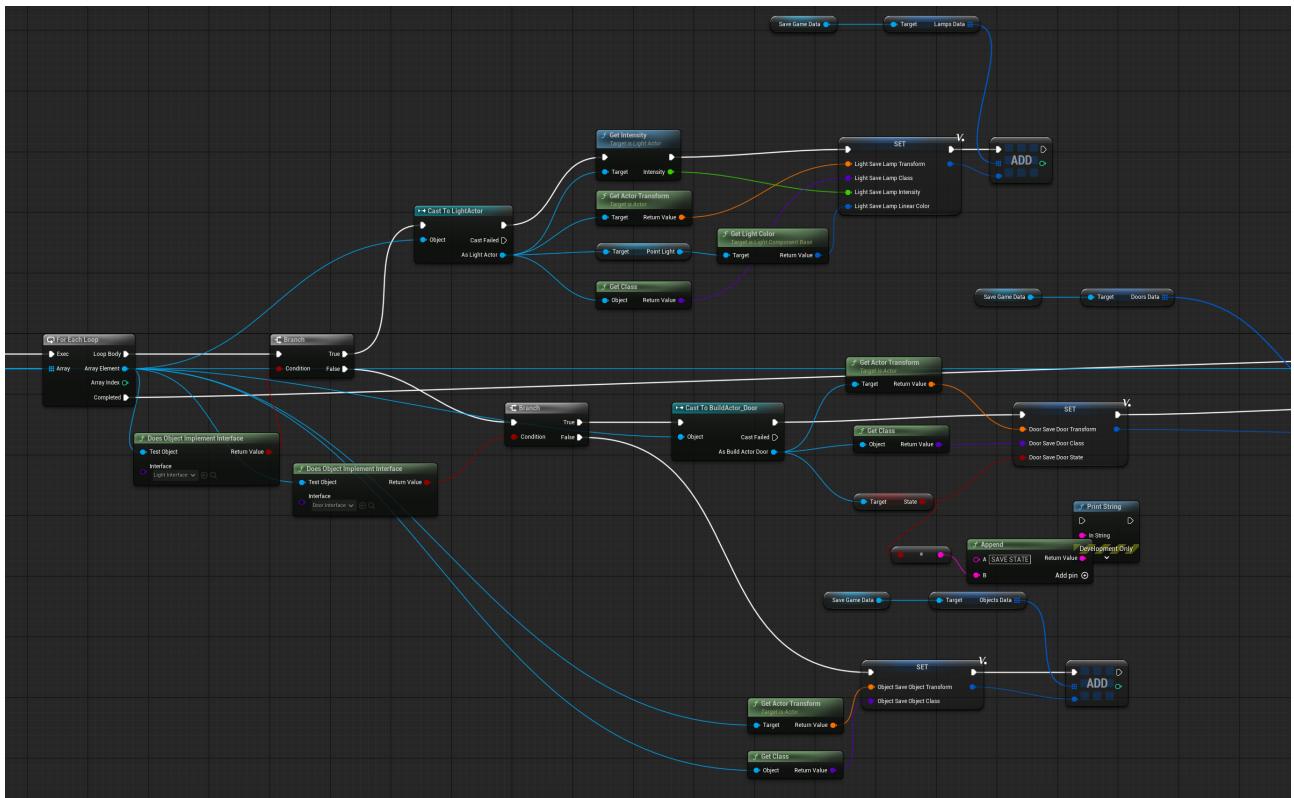


FIGURE 37 – Fonction SavePlayerData de BP_GameInstance - Partie 2

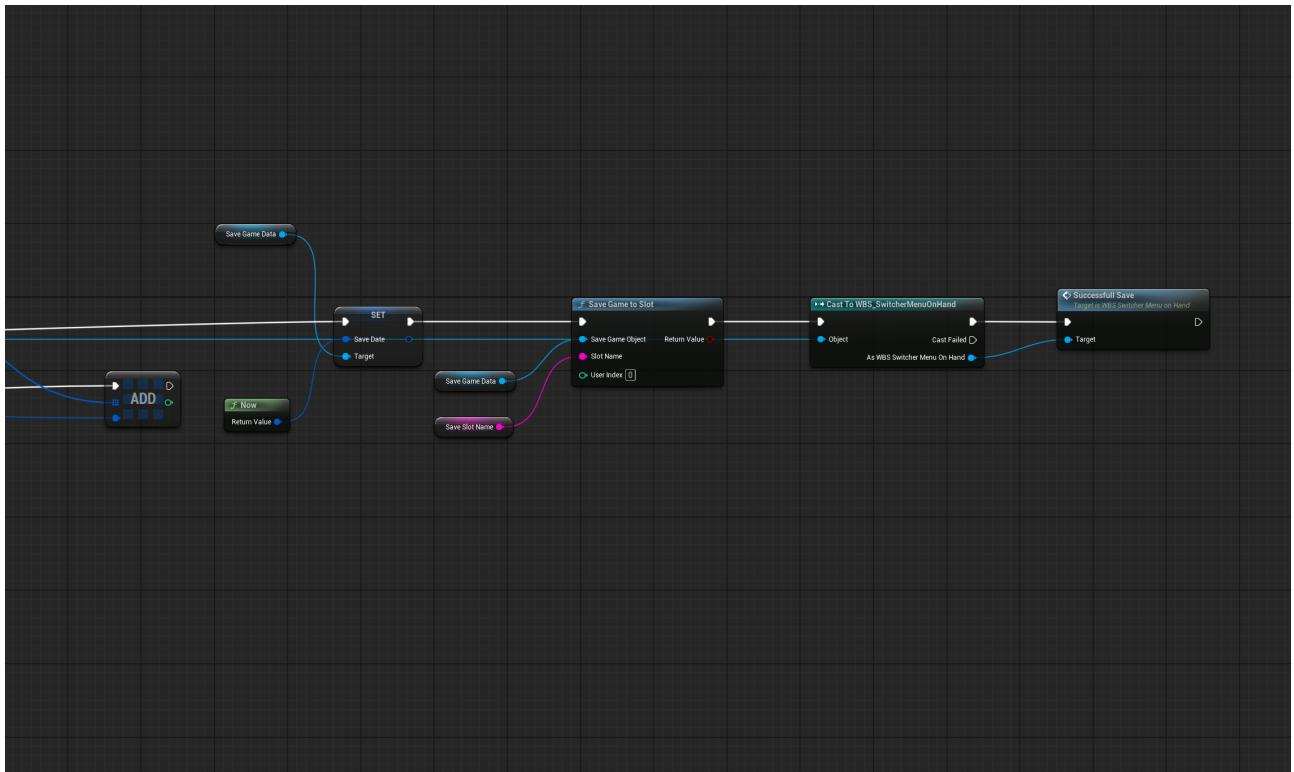


FIGURE 38 – Fonction SavePlayerData de BP_GameInstance - Partie 3

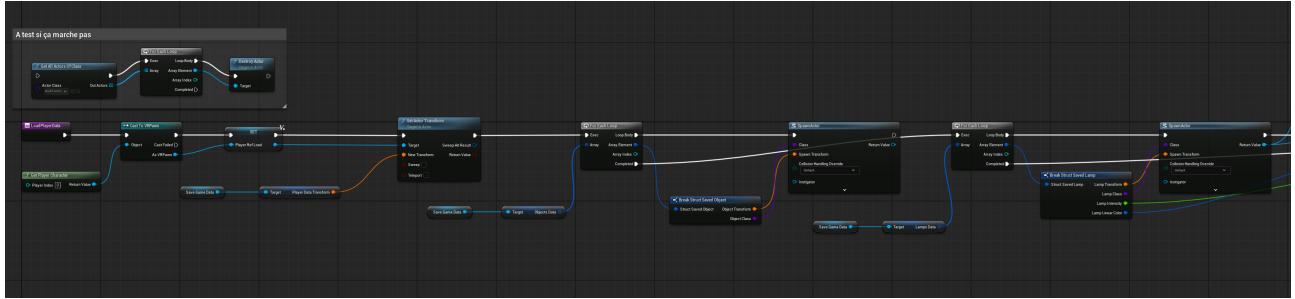


FIGURE 39 – Fonction LoadPlayerData de BP_GameInstance - Partie 1

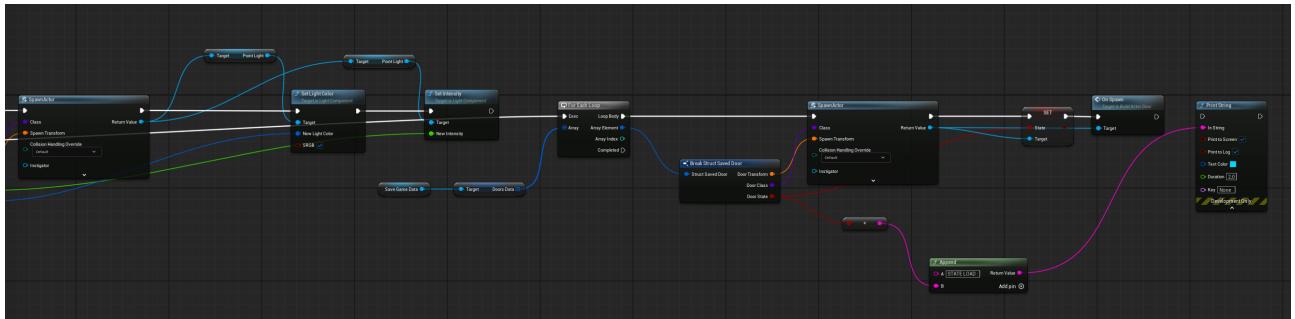


FIGURE 40 – Fonction LoadPlayerData de BP_GameInstance - Partie 2

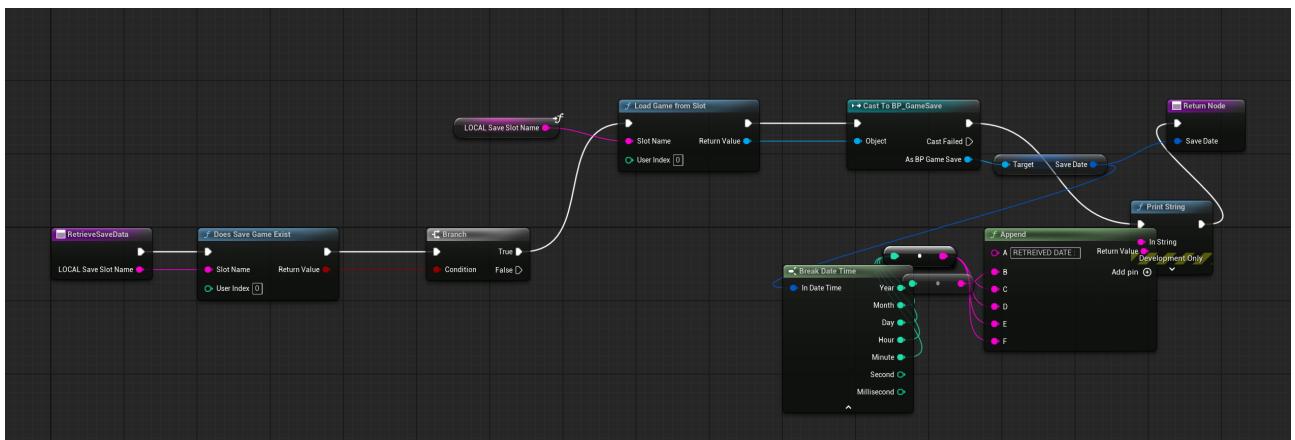


FIGURE 41 – Fonction RetrieveSaveData de BP_GameInstance

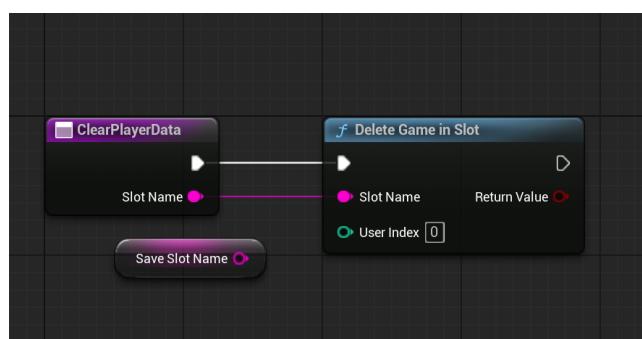


FIGURE 42 – Fonction ClearPlayerData de BP_GameInstance

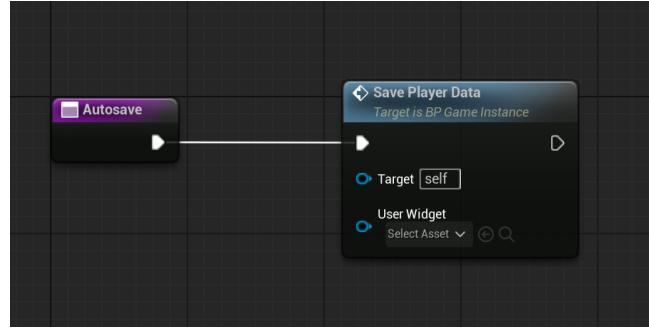


FIGURE 43 – Fonction Autosave de BP_GameInstance

B.1.3 Save Game

BP_GameSave :

Cet Actor correspond à une sauvegarde. On y stocke tout ce dont on a changé depuis le lancement de la partie. Ses variables sont détaillées sur la Figure 36.

B.1.4 Structures

Struct_PlayerSaveData :

Cette Structure est un conteneur permettant de sauvegarder les informations du joueur. Ses variables sont détaillées sur la Figure 32.

Struct_SavedObject :

Cette Structure est un conteneur permettant de sauvegarder les informations des BuildActors de la scène. Ses variables sont détaillées sur la Figure 33.

Struct_SavedLamp :

Cette Structure est un conteneur permettant de sauvegarder les informations des LightActors de la scène. Ses variables sont détaillées sur la Figure 34.

Struct_SavedDoor :

Cette Structure est un conteneur permettant de sauvegarder les informations des DoorActors de la scène. Ses variables sont détaillées sur la Figure 35.

B.1.5 User Widgets

WP_MenuInteractif :

Ce Widget correspond au menu interactif. Les éléments importants sont les 5 boutons. BTN_PlaceObject, BTN_ChangeTexture, BTN_ChangeLighting et BTN_SaveSession permettent de changer de Widget via le Widget Switcher WBS_SwitcherMenuOnHand car l'événement "On Clicked" est lié à "Call Clicked on". BTN_QuitSession ouvre le Lobby.

BTN_ChangeLighting	■	□	○
BTN_ChangeTexture	■	□	○
BTN_PlaceObject	■	□	○
BTN_QuitSession	■	□	○
BTN_SaveSession	■	□	○

FIGURE 44 – Variables de WB_MenuInteractif

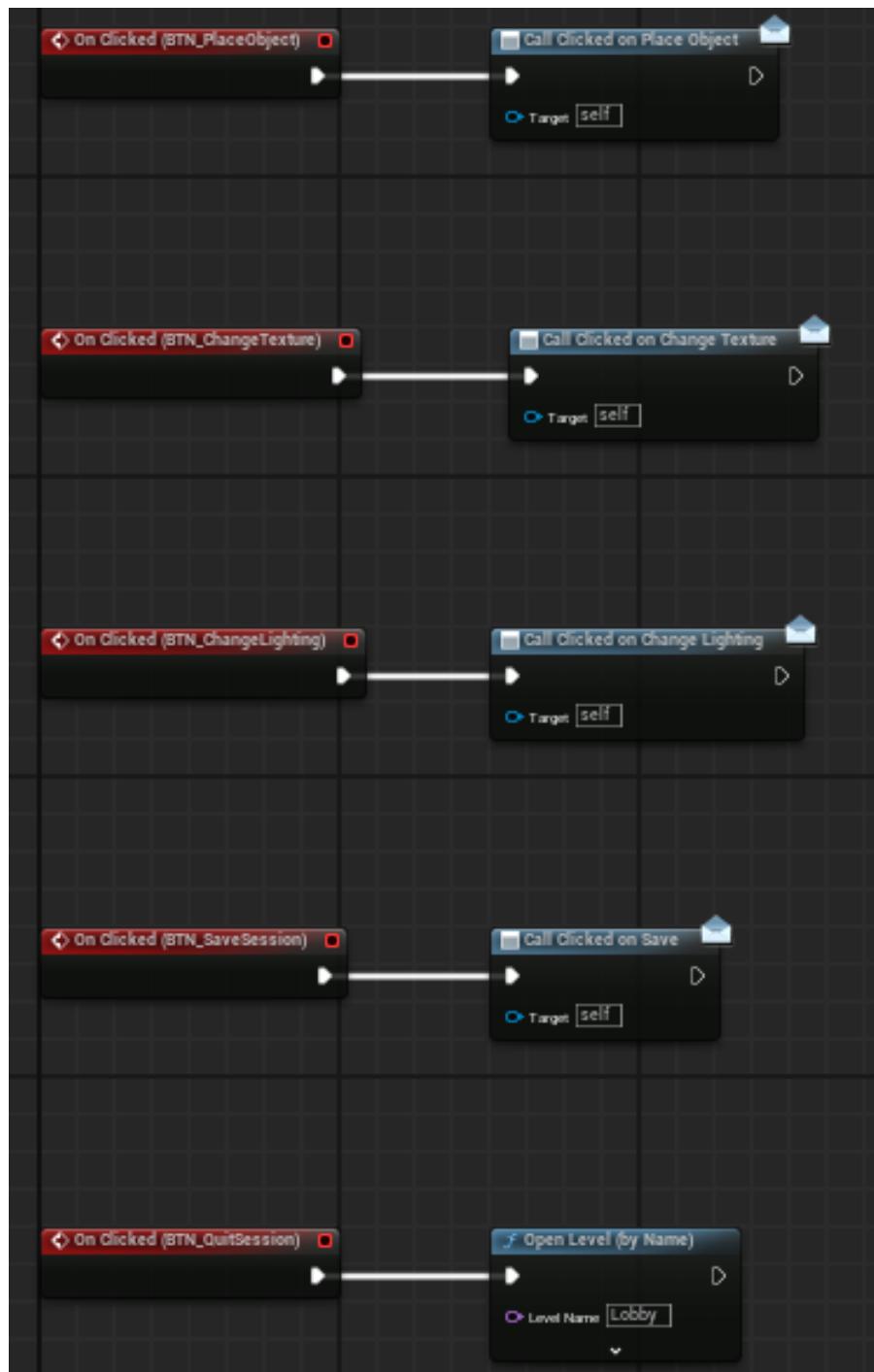


FIGURE 45 – Graphe de WB_MenuInteractif

WB_SauvegardeReussie :

Ce Widget correspond au Widget de "Confirmation de la réussite de la sauvegarde et ne contient pas de code. Il ne fait que s'afficher lorsqu'il est appelé.

WB_TypeOfLighting :

Ce Widget correspond au menu de sélection du type d'éclairage à modifier et possède 3 boutons qui permettent de naviguer dans les menus via le WBS_SwitcherMenuOnHand. BTN_NaturalLighting et BTN_ArtificialLighting permettent de choisir et mener au menu correspondant à l'éclairage choisi. BTN_Back permet de retourner au menu interactif.

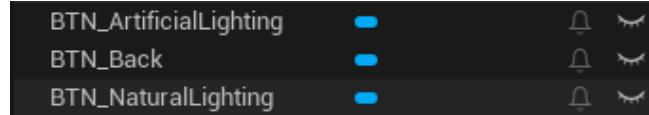


FIGURE 46 – Variables de WB_TypeOfLighting

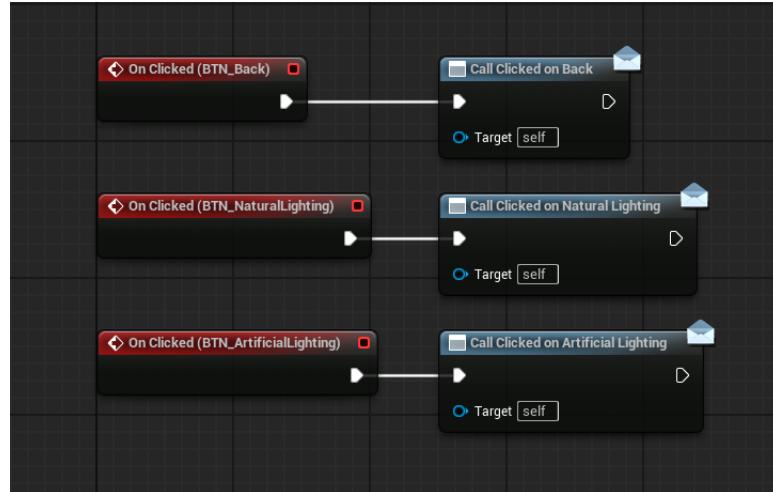


FIGURE 47 – Graphe de WB_TypeOfLighting

WB_NaturalLighting :

Ce Widget correspond au menu de modification de l'éclairage naturel contient une combobox et 2 boutons. ComboBox.Choice permet d'en cliquant dessus de dérouler les choix d'éclairage naturel. Ensuite, lorsque l'utilisateur clique sur son choix, il est directement gardé dans la box. BTN_Confirm et BTN_Back permettent respectivement de confirmer et de retourner au Widget précédent. La variable WeatherSelected est une référence à l'Actor correspondant à la météo choisie. A la suite de Event Construct, on remplit ComboBox.Choice avec les différents choix possibles. On Selection Changed permet d'associer le choix qui est un String à l'Actor correspondant. Par exemple, quand on clique sur Nuit, WeatherSelected stocke BP_Weather_Night. On Clicked (BTN_Confirm) permet d'appeler la fonction ChangeWeather qui provient du WB_WeatherManager afin d'appliquer sur la scène la météo stockée dans WeatherSelected. On Clicked (BTN_Back) permet de revenir en arrière à l'aide de WBS_SwitcherMenuOnHand.

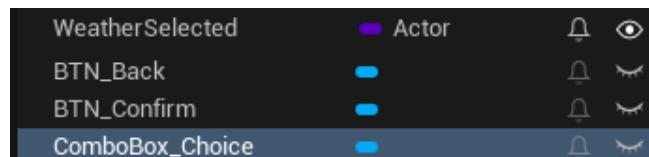


FIGURE 48 – Variables de WB_NaturalLight



FIGURE 49 – Graphe de WB_NaturalLight - Choix de la combobox

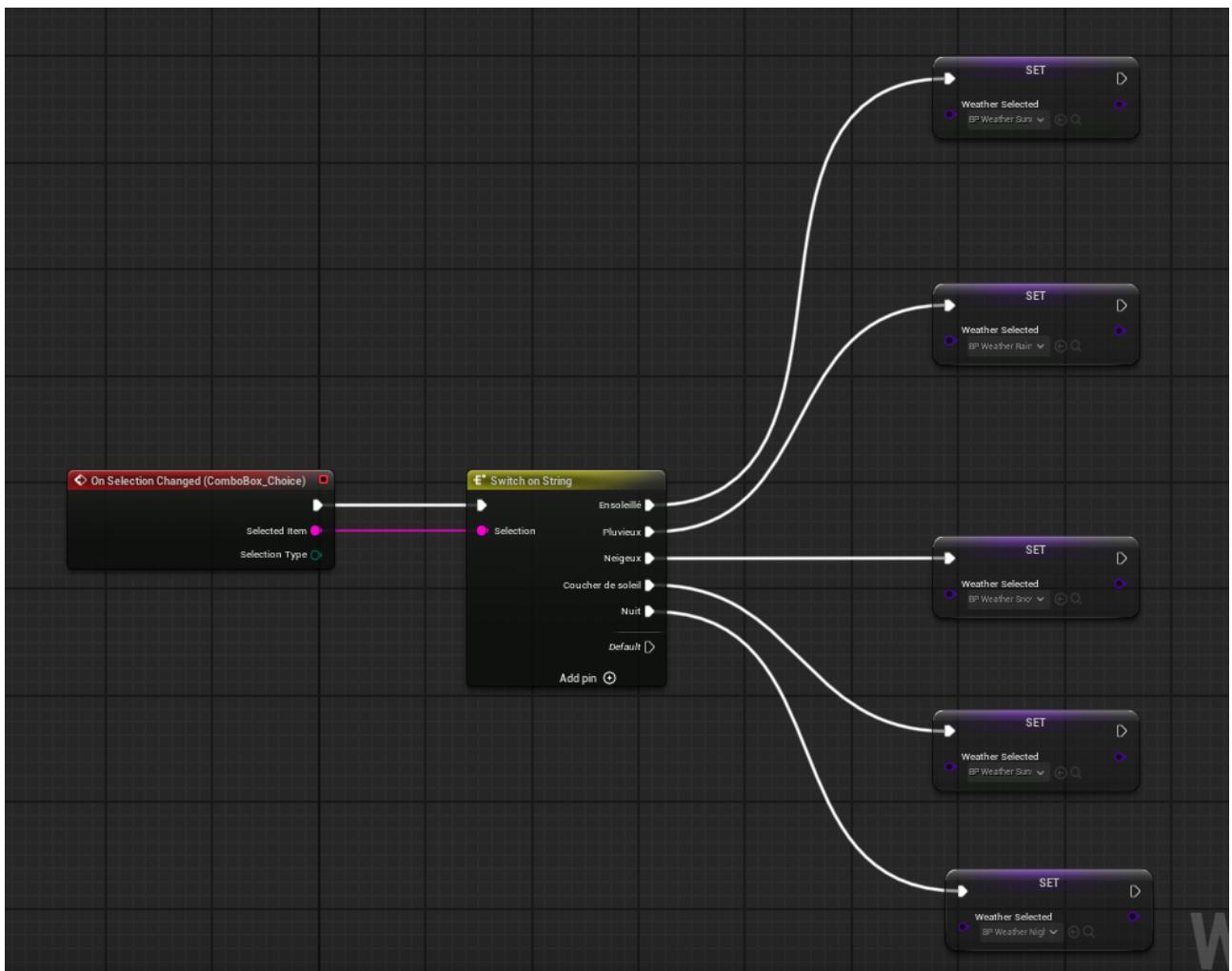


FIGURE 50 – Graphe de WB_NaturalLight - On Selection Changed

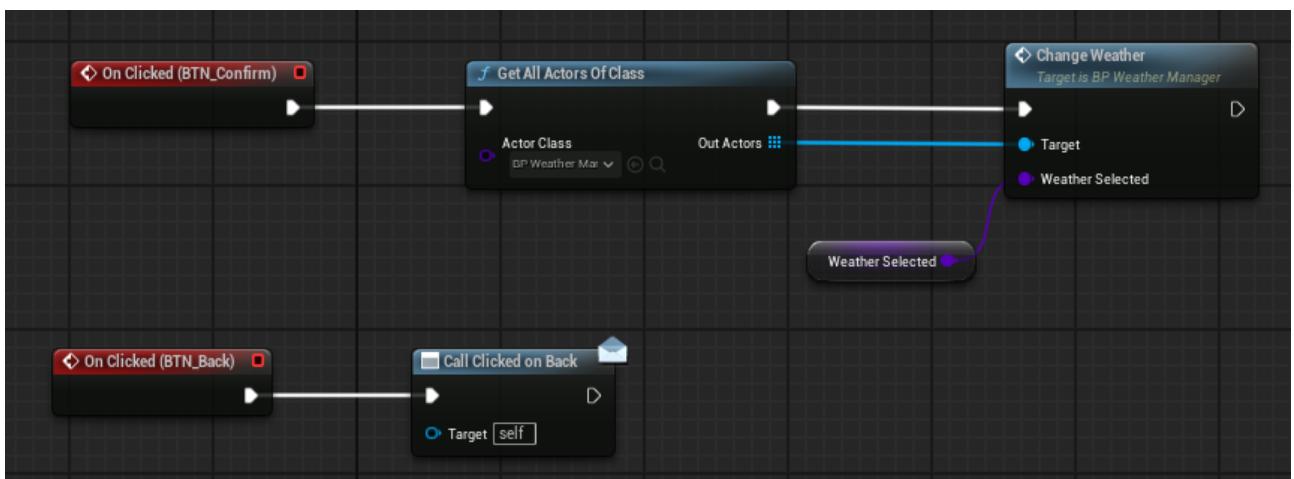


FIGURE 51 – Graphe de WB_NaturalLight - BTN_Confirm et BTN_Back

WB_TypeOfPlacing :

Ce Widget correspond au menu de sélection du type d'objet à placer et possède 3 boutons qui permettent de naviguer dans les menus via le WBS_SwitcherMenuOnHand. 'BTN_PlaceFurniture' et 'BTN_PlaceBuilding' permettent de choisir et mener au menu correspondant aux objets choisis. 'BTN_Back' permet de retourner au menu interactif.

BTN_Back		
BTN_PlaceBuilding		
BTN_PlaceFurniture		

FIGURE 52 – Variables de WB_TypeOfPlacing

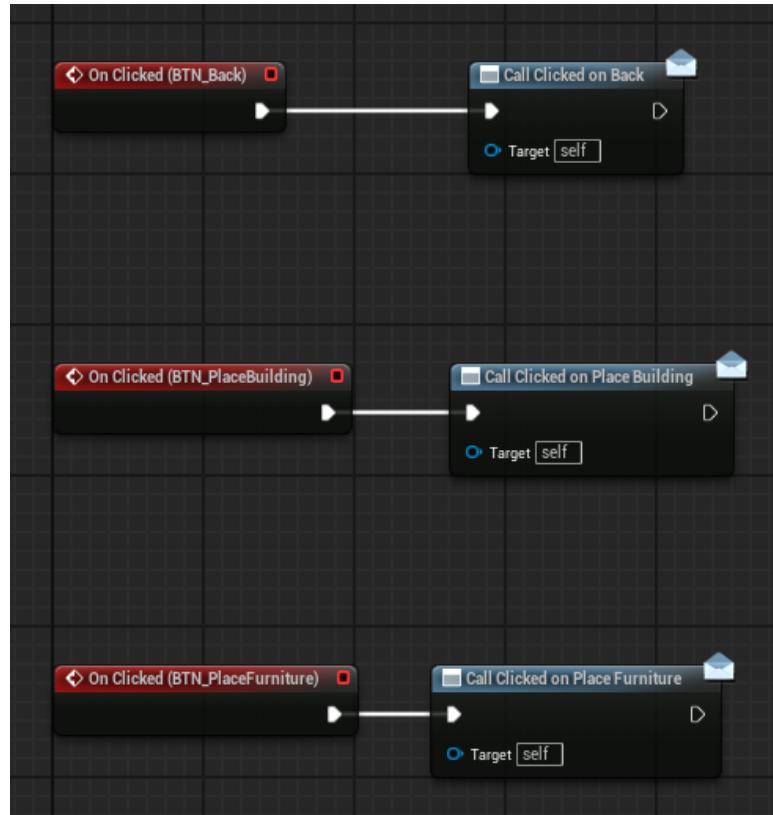


FIGURE 53 – Graphe de WB_TypeOfPlacing

WB_SaveSlot :

Dans l’Event Graph, l’événement EventConstruct permet qu’à chaque fois que ce Widget est créé le texte permettant d’afficher la date de la sauvegarde soit mis à jour. Lorsque l’utilisateur appuie sur le bouton BTN_LoadSave, le level BaseLevel est ouvert et les éléments de la sauvegarde sont exploités grâce à l’utilisation de l’événement LoadGame de BP_GameInstance. Le bouton BTN_DeleteSave permet de passer au menu de confirmation de suppression grâce au Widget Switcher. La fonction GetSlotIndex permet d’afficher, sur son emplacement, son index. La fonction GetSaveDate permet d’afficher la date sur l’emplacement. La fonction UpdateText permet de lancer la fonction RetrieveSaveData de BP_GameInstance afin de stocker la date de sauvegarde dans la variable SaveDate.



FIGURE 54 – Event Graph de WB_SaveSlot

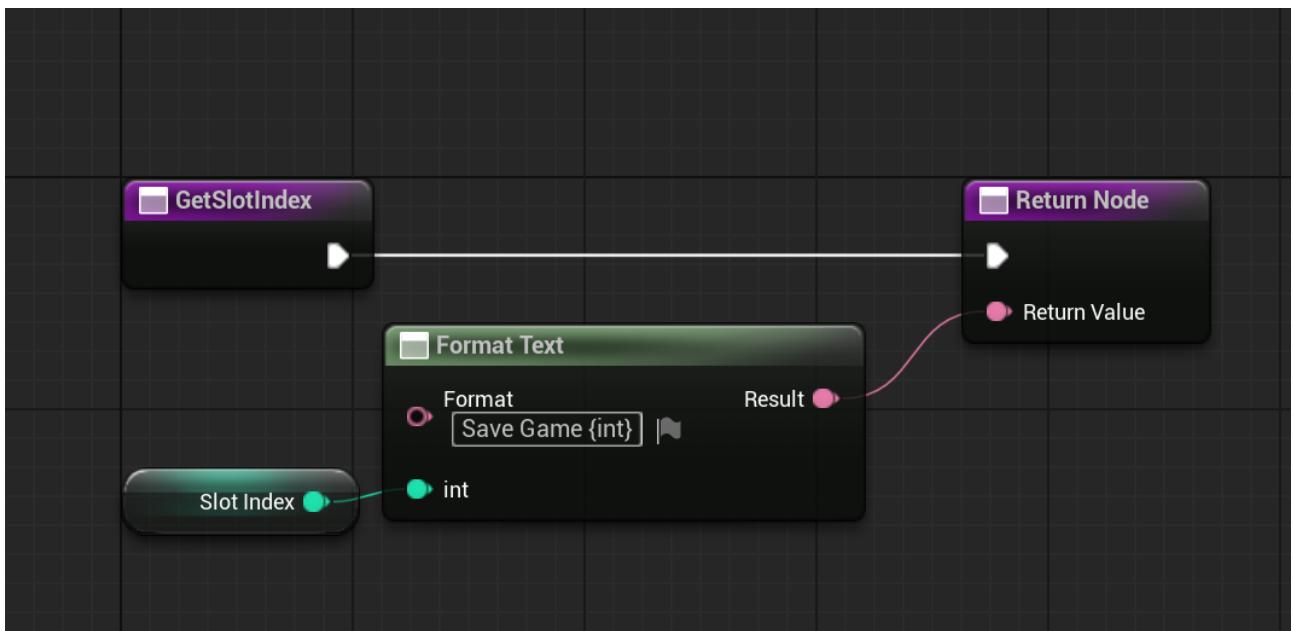


FIGURE 55 – Fonction GetSlotIndex de WB_SaveSlot

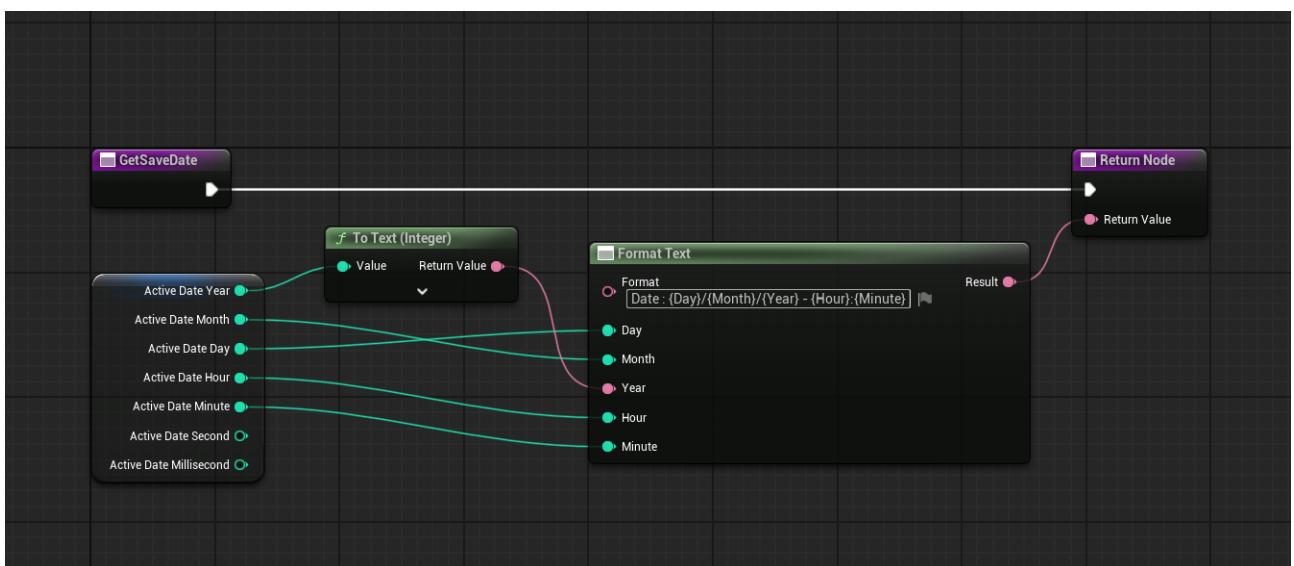


FIGURE 56 – Fonction GetSaveDate de WB_SaveSlot

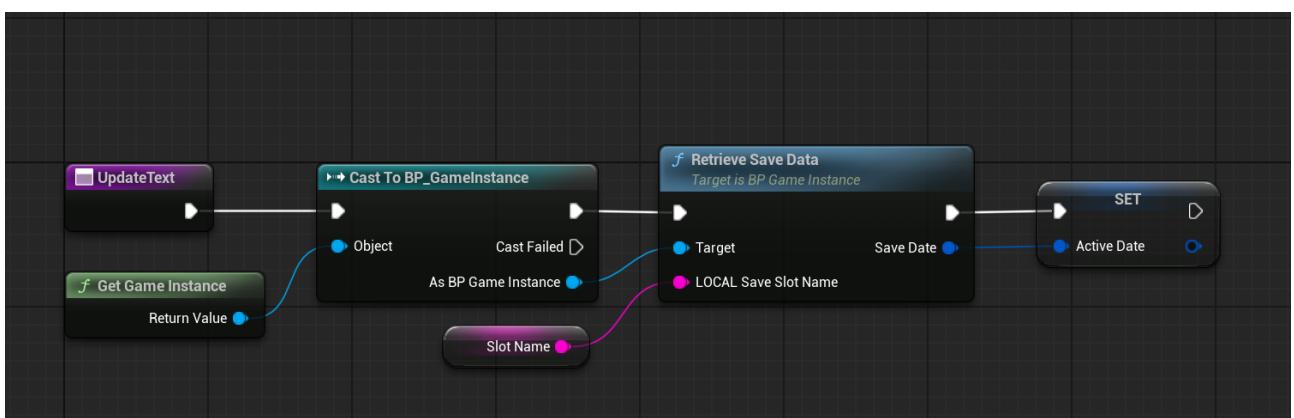


FIGURE 57 – Fonction UpdateText de WB_SaveSlot

WB_SaveMenu :

L'Event Graph associe chaque SaveSlot à son action Clear, qui enregistre le nom du SaveSlot et lance la logique dans le WBS_SwitcherLobbyMenu.

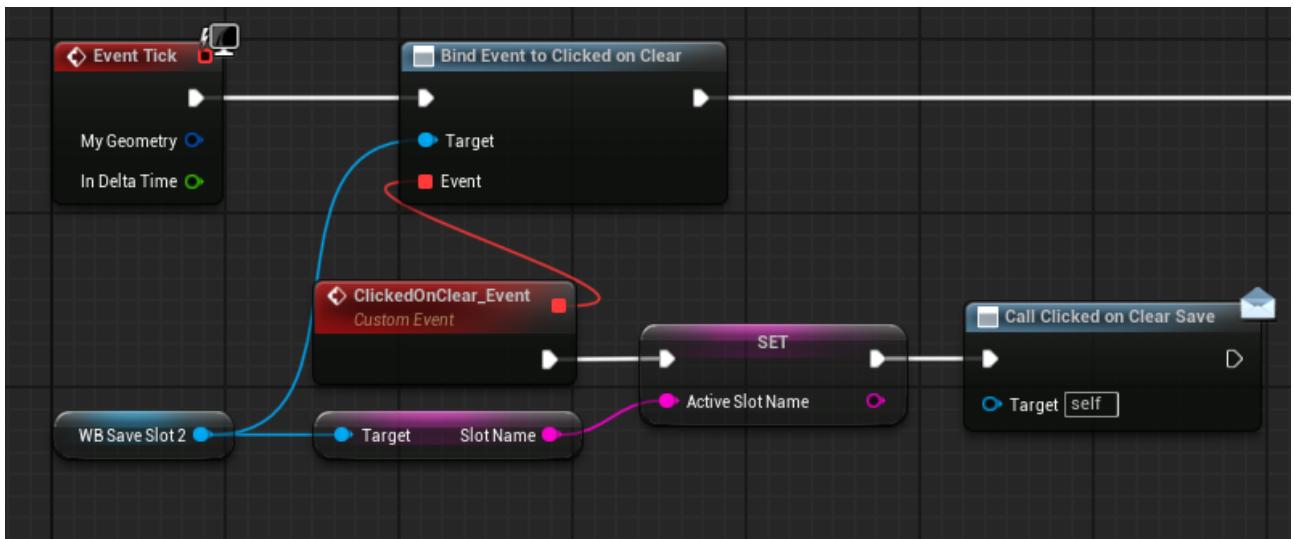


FIGURE 58 – Event Graph de WB_SaveMenu

WB_ClearSlotConfirmation :

Dans l'Event Graph de ce Widget, le bouton BTN_Cancel permet de revenir au WB_SaveMenu grâce au Widget Switcher. Le bouton BTN_Confirm permet également d'utiliser le Widget Switcher en nous faisant également retourner au WB_SaveMenu et en effectuant la suppression de la sauvegarde.

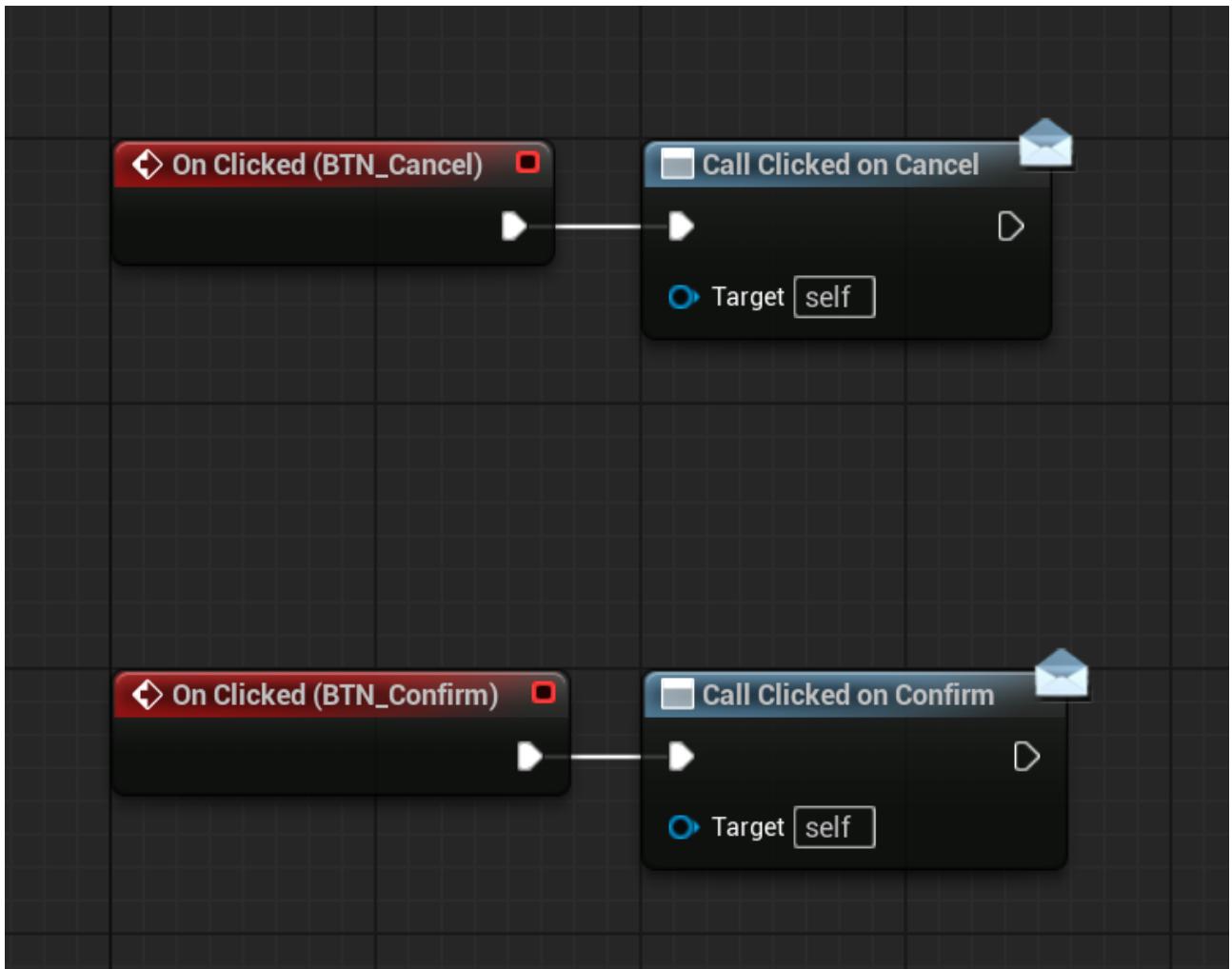


FIGURE 59 – Event Graph de WB_ClearSlotConfirmation

WBS_SwitcherLobbyMenu :

Ce Widget switcher permet de gérer l'affichage des différents widgets du menu principal. La logique reste la même pour cette tâche. Il y a en plus la logique de suppression de sauvegarde mentionnée précédemment.

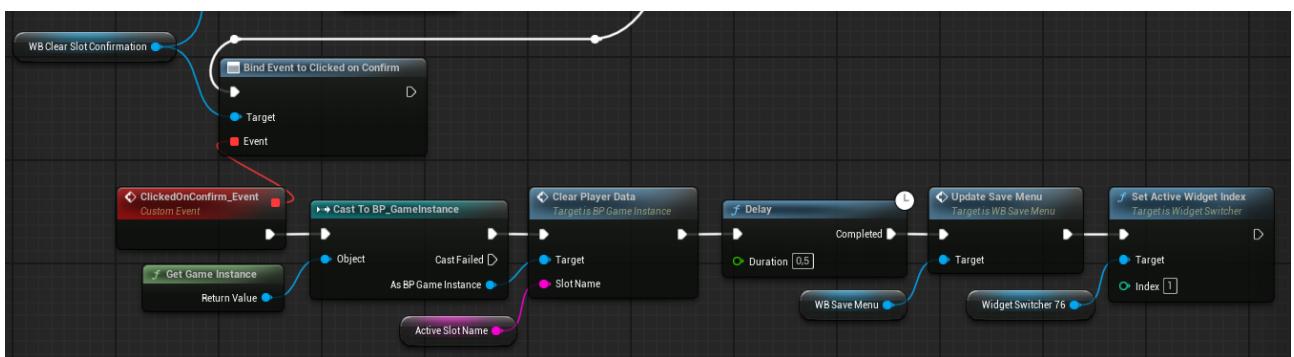


FIGURE 60 – Logique de suppression de sauvegarde dans WBS_SwitcherLobbyMenu

WBS_SwitcherMenuOnHand :

Ce Widget Switcher permet de changer de widget dans le menu sur la main. Il gère la logique en changeant l'index du widget switcher lorsqu'un bouton pressé dans l'un des widget appelle un event qui doit changer de page.

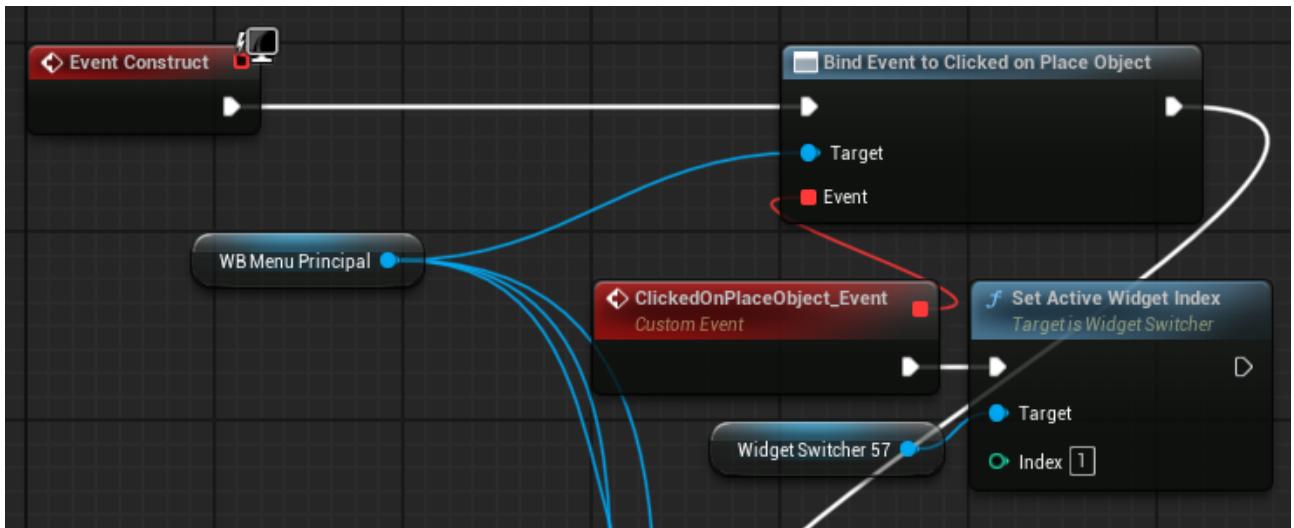


FIGURE 61 – Exemple de logique de widget switcher

Lorsque l'utilisateur passe son pointeur laser sur un des boutons pour changer de mode, le nom du mode correspondant s'affiche à côté.

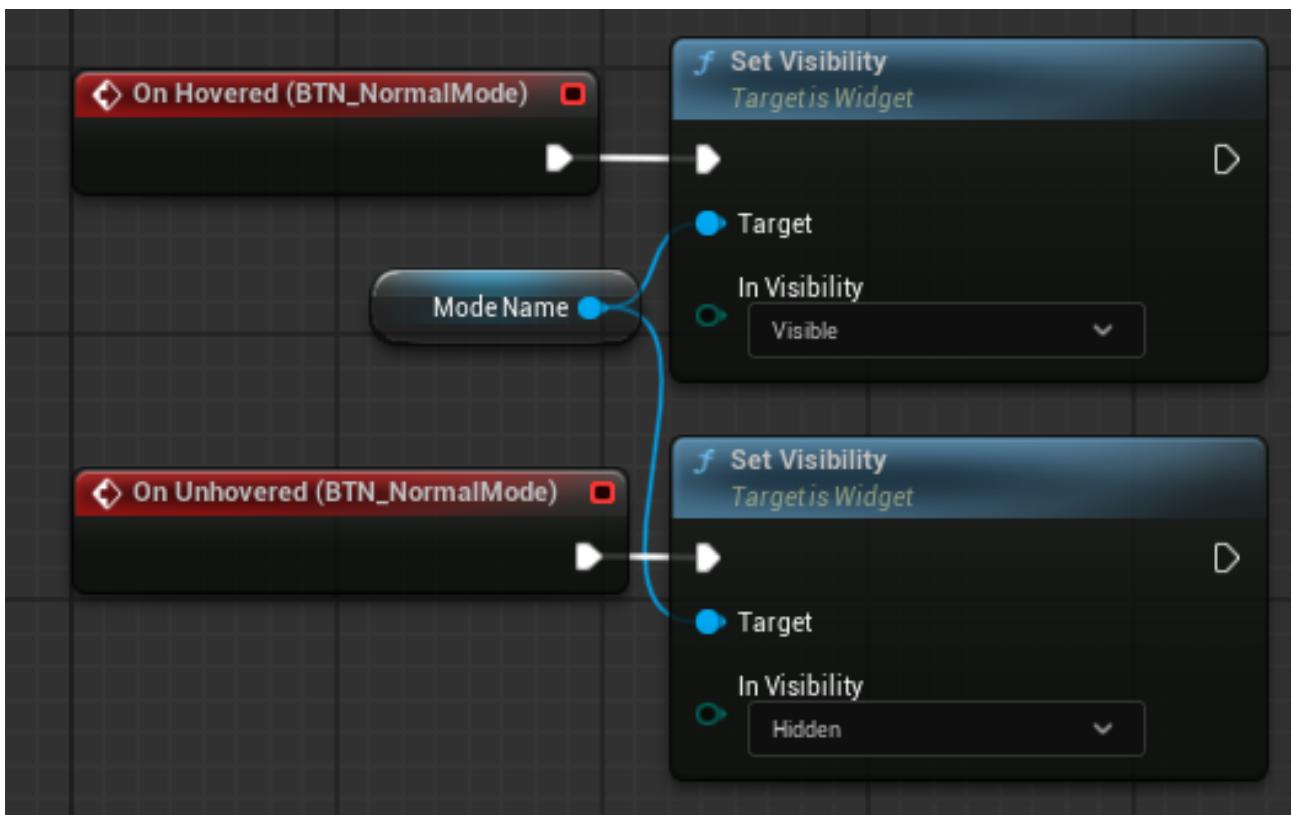


FIGURE 62 – Exemple de logique d'apparition du nom du Mode Normal

Lorsque la sauvegarde est réussie, l'événement SuccessfullSave se lance, affichant un message de confirmation pendant une seconde pour que l'utilisateur ait un retour clair.

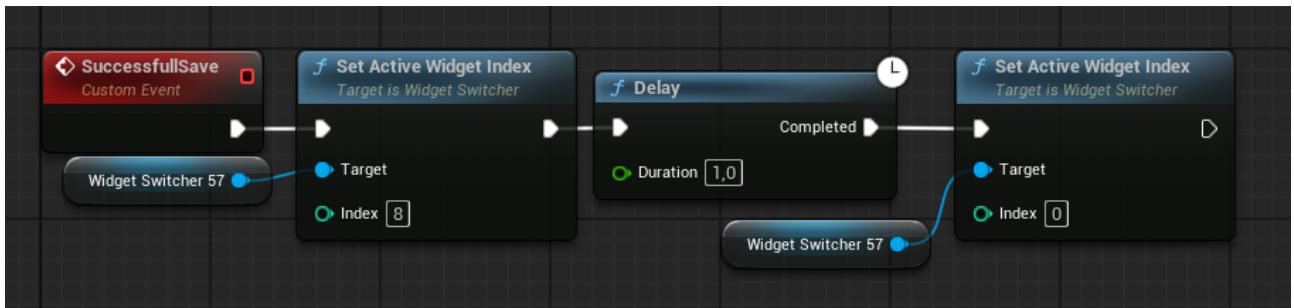


FIGURE 63 – Custom_Event_SucessfullSave

Lors du choix de l'objet à placer, le joueur est invité à choisir parmi une lilesrairie d'objets. Pour naviguer parmi ceux-ci, il dispose d'une barre de défilement. En temps normal, cette barre est utilisable comme telle mais comme nous devons l'utiliser en VR, il nous faut implémenter la logique qui permet son utilisation. Pour se faire, une fonction ApplyScroll est appelée avec le mouvement du joystick de la manette droite lorsque le menu est ouvert. Cette fonction applique simplement le défilement à la barre selon la valeur de l'input.

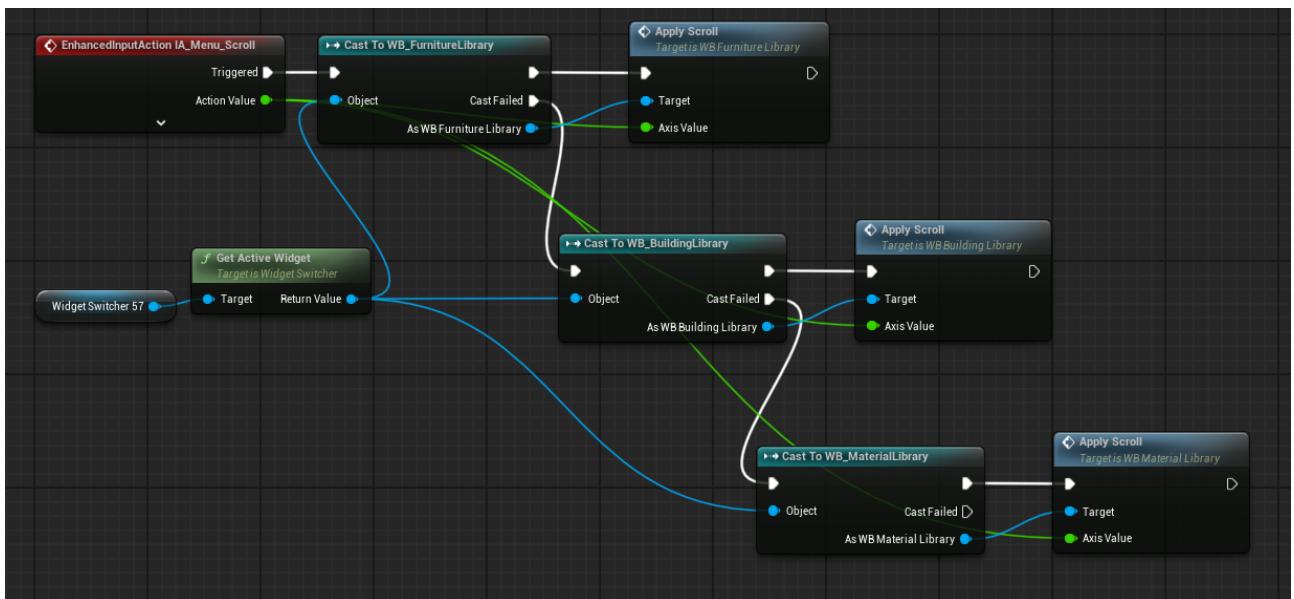


FIGURE 64 – Logique de défilement

WBS_SwitcherControllerDisplay :

Ce widgetswitcher permet de gérer quel widget afficher pour l'explication des différentes touches utilisables. Les touches de la manettes étant restreintes, nous avons du appliquer plusieurs actions sur les mêmes boutons. Pour simplifier l'affichage de ses explications, nous avons séparé cet affichage pour chaque mode que l'utilisateur peut utiliser. Nous avons donc créé des boutons cliquable en haut du widget switcher permettant de naviguer dans les différents modes.

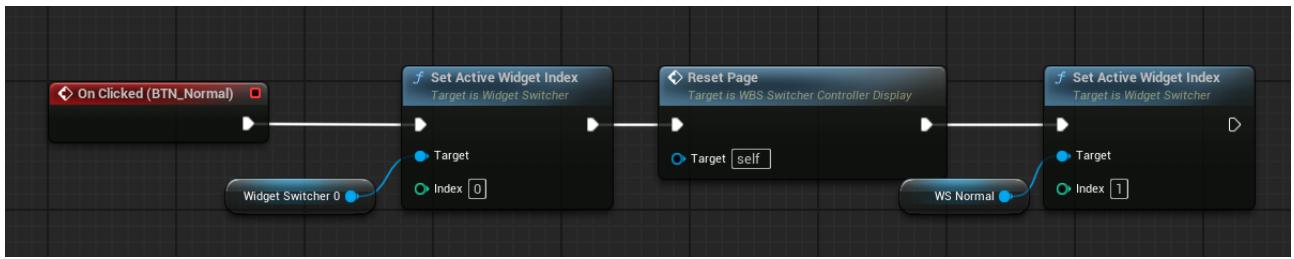


FIGURE 65 – Exemple d'un bouton de WBS_SwitcherControllerDisplay

Les autres boutons suivent exactement la même logique, seul l'indexe du Widget Swicher 0 change. Cet event fait appel à la fonction ResetPage et change l'indexe d'un autre swticher spécifique au mode choisi. Cela permet de changer la couleur du bouton du mode actif, partageant clairement au joueur quel mode est sélectionné.



FIGURE 66 – Fonction ResetPage de WBS_SwitcherControllerDisplay

HUD :

Le HUD est composé d'un widget switcher permettant d'afficher les bonnes images selon le mode dans lequel se trouve le joueur. Lorsque ce mode change, l'événement ChangeHUD est appellée avec en entrée le nom du mode. L'événement consiste simplement à changer l'indexe du widget switcher selon le nom du mode afin de faire correspondre le HUD au mode.

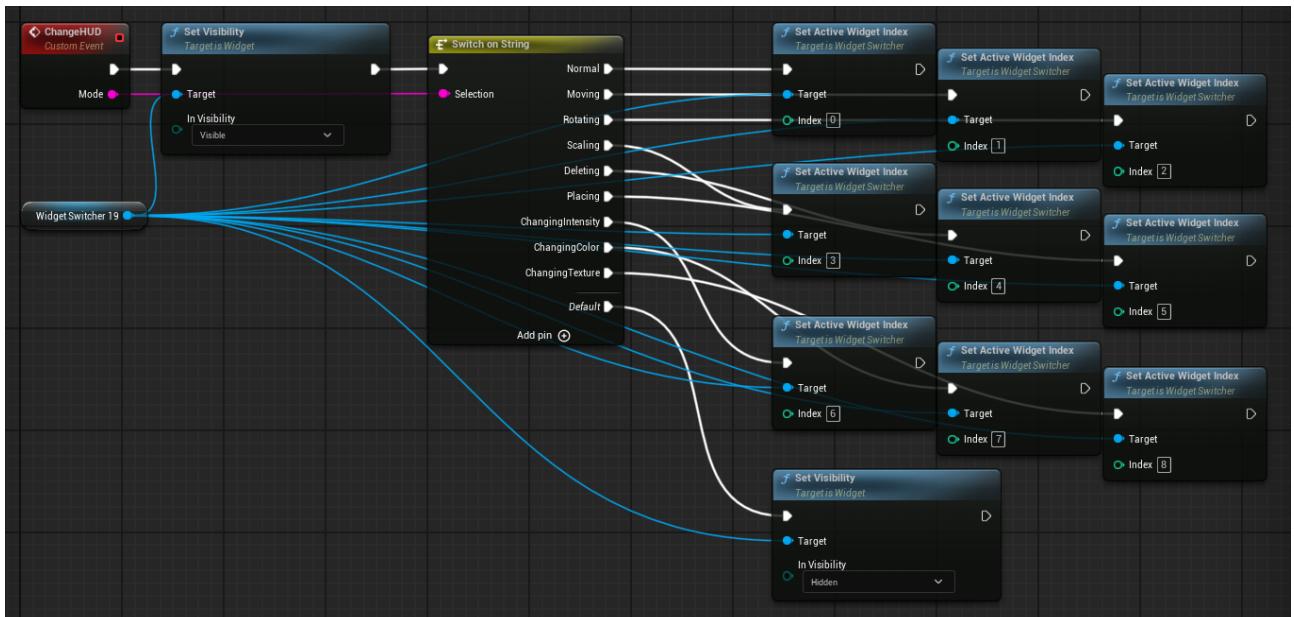


FIGURE 67 – ChangeHUD

L'appel de cet événement se fait à partir de l'événement ChangeMode au sein du VRPawn.

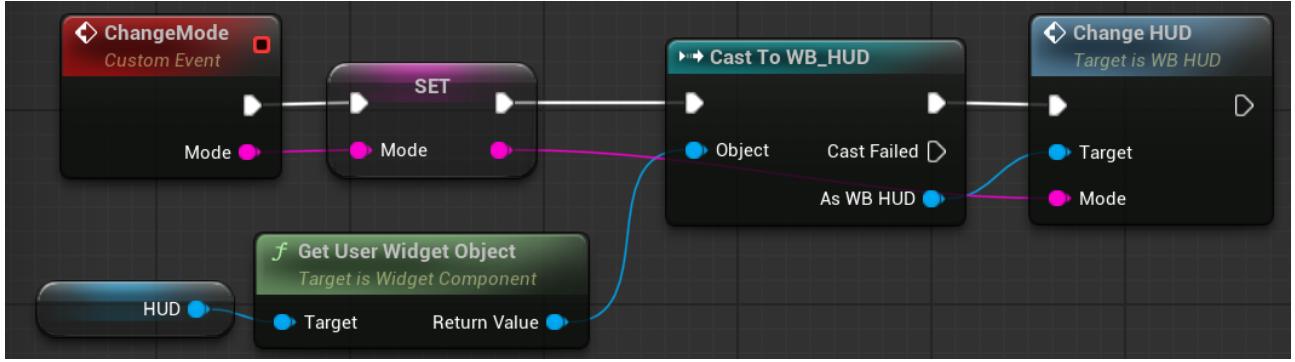


FIGURE 68 – Changemode

B.1.6 InputMapping

La plupart des logique blueprint sont lancées à partir d'un bouton de la manette. Il a donc fallu créer l'association de ces boutons aux actions. Nous avons donc créé le mapping suivant en associant chaque touche de la manette à un input action.

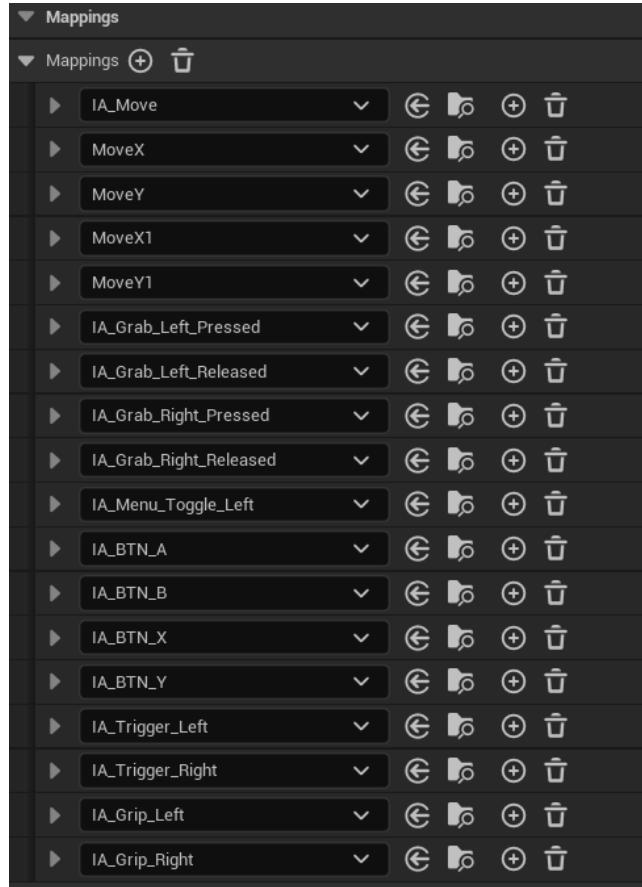


FIGURE 69 – InputMapping

B.1.7 VRPawn

SaveLogic :

Cette implémentation permet de lancer la fonction SavePlayerData lorsque l'événement PushButtonSave arrive.



FIGURE 70 – Save Logic

MenuLogic :

Pour accéder à toutes les fonctionnalités, l'utilisateur a besoin d'un menu. Ce menu, placé sur la main gauche, peut être appelé à tout moment en appuyant sur la touche "menu" de la manette gauche. Le menu apparaît alors tout en cachant la main gauche pour une meilleure visibilité. Appuyer une deuxième fois sur ce bouton ferme ce menu et rend visible la main cachée.

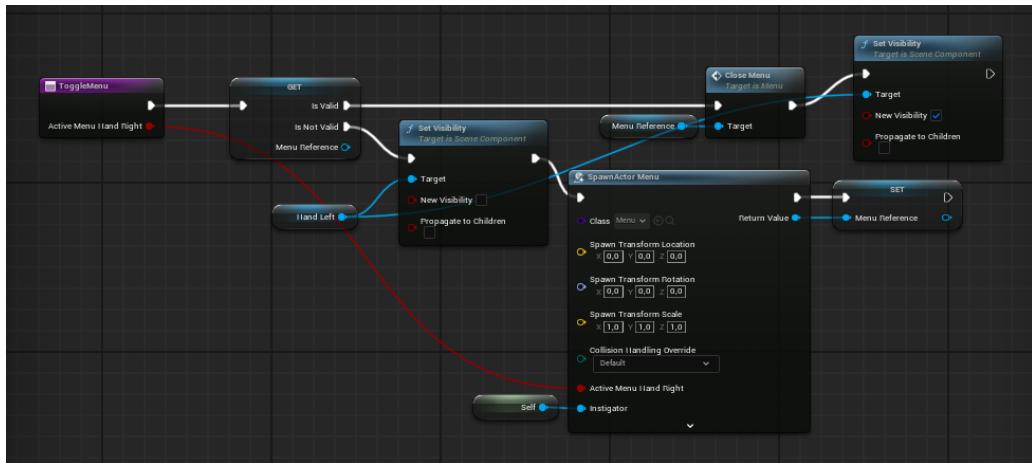
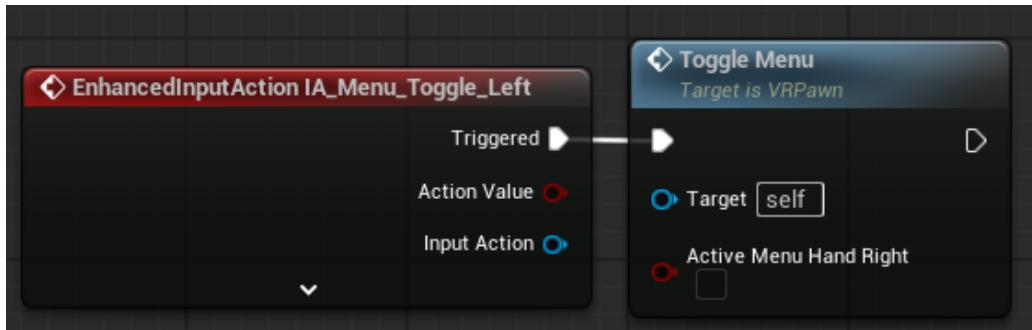


FIGURE 71 – Affichage du menu sur la main

DeplacementObjectWhileGrabbed :

Lorsque l'utilisateur est dans le mode Sélection, Déplacement, Rotation ou mise à l'échelle, il peut attraper un objet en appuyant sur la gâchette de la manette droite. Après cela, tant que l'utilisateur maintient la gâchette droite, l'objet se déplacera selon l'axe de la manette droite et pourra subir différentes opérations selon le mode choisi. Une fois la gâchette droite relâchée, l'objet retombera vers le sol.

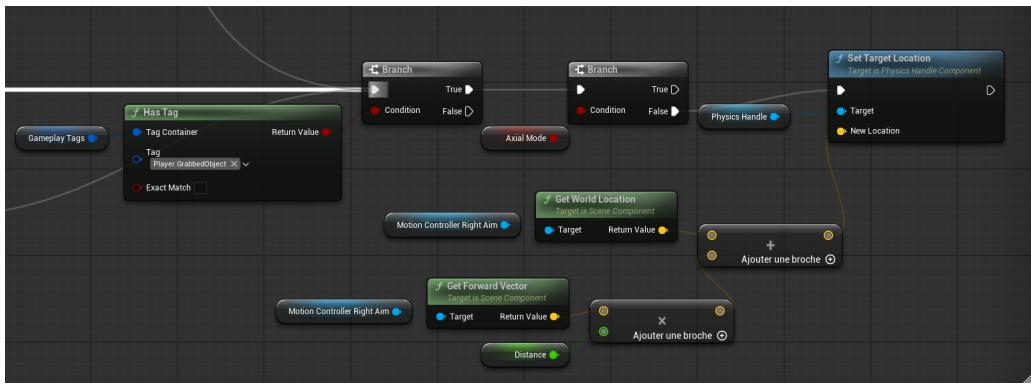


FIGURE 72 – Logique du déplacement d'un objet lorsqu'il est attrapé

PlacingPreview :

En allant dans le menu situé sur la main, l'utilisateur peut choisir de placer un objet. Une fois l'objet sélectionné, l'utilisateur pourra voir une prévisualisation de l'objet sur sa carte et lorsqu'il appuiera sur la gâchette droite, l'objet se placera.

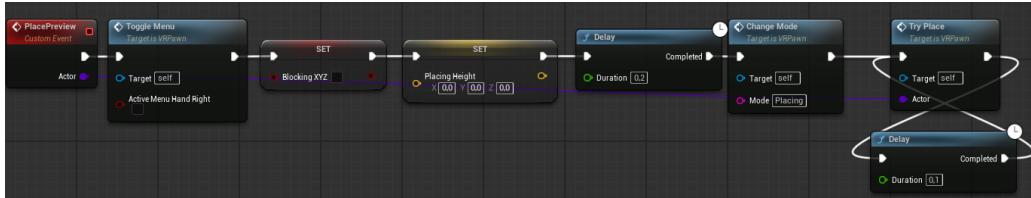


FIGURE 73 – Event PlacingPreview

ChangingMode :

Toujours dans le menu situé sur la main, l'utilisateur peut ajuster entre les différents en cliquant sur celui désiré. Cette logique permet également de changer le HUD en conséquence.

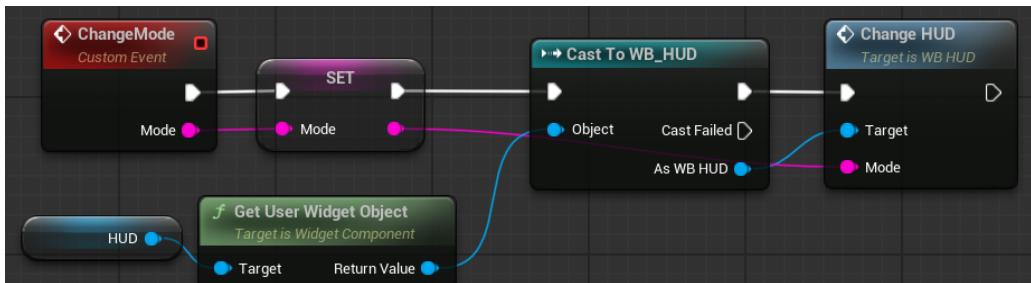


FIGURE 74 – Event ChangeMode

Joystick gauche :

Le joystick gauche est utilisé pour faire un système classique de mouvement. En fonction de la distance du joystick par rapport à son centre, l'utilisateur se déplacera plus ou moins vite.

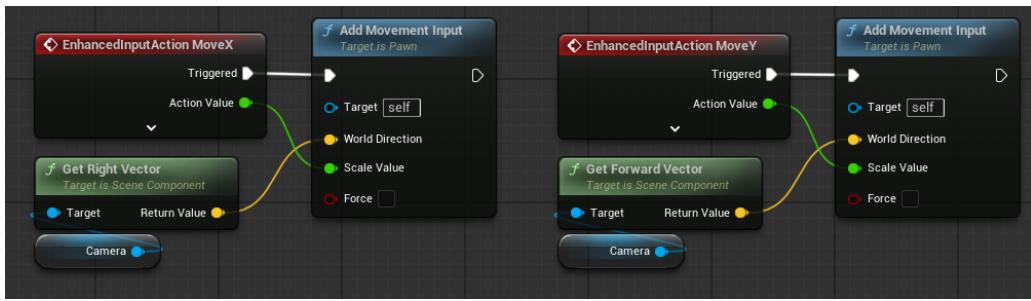


FIGURE 75 – Joystick gauche

Joystick droit :

Le joystick droit a 2 fonctions. La première est de pouvoir faire une rotation de la vue. En effet, notre caractère peut naturellement se tourner en fonction de comment se tourne le joueur mais nous avons remarqué que cela uniquement n'est pas toujours très pratique. Pour cette raison, nous avons choisi de rajouter une rotation manuelle avec un joystick lorsqu'on le déplace selon l'axe X.

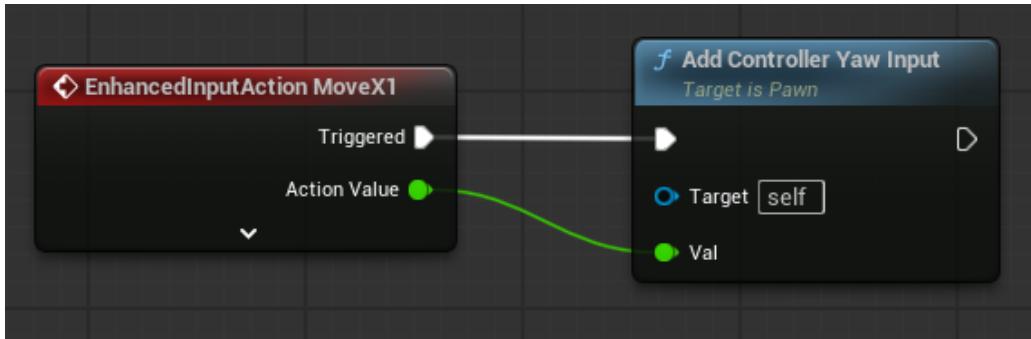


FIGURE 76 – Joystick droit - Rotation caméra

La deuxième fonction de ce joystick est de lorsqu'on le déplace selon l'axe Y, de se téléporter vers où on pointe la manette droite. Cela permet une plus grande latitude de déplacements que les seuls déplacements classiques avec le joystick gauche.

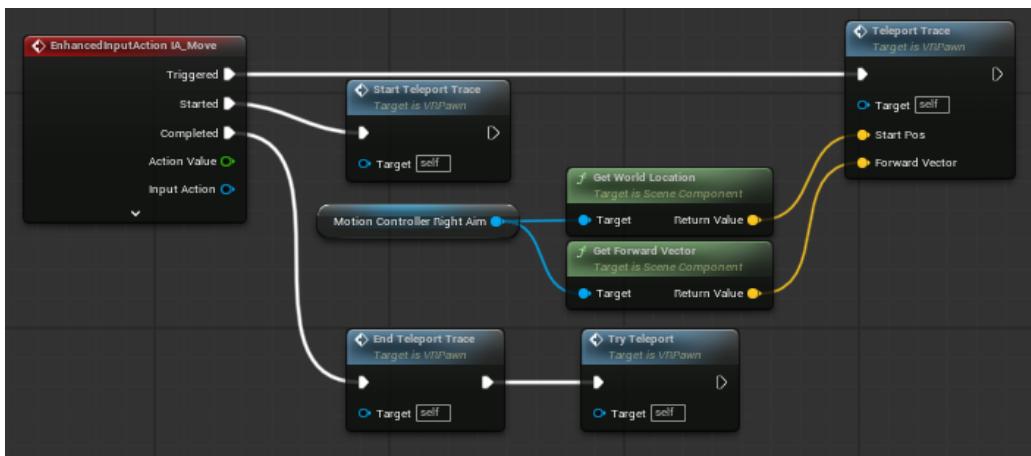


FIGURE 77 – Joystick droit - Téléportation

Bouton X :

- Mode Placement : Tourner l'objet prévisualisé d'un angle correspondant à la grille (de 90° par défaut) autour de l'axe Z.

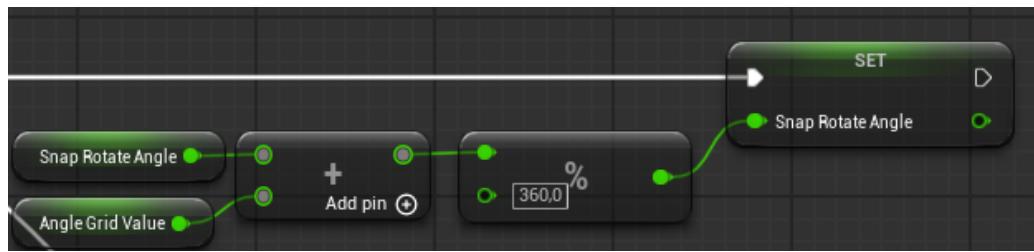


FIGURE 78 – Bouton X - Mode Placement

- Mode Déplacement : Changer la préférence de déplacement (activer ou désactiver le déplacement selon les axes XYZ).

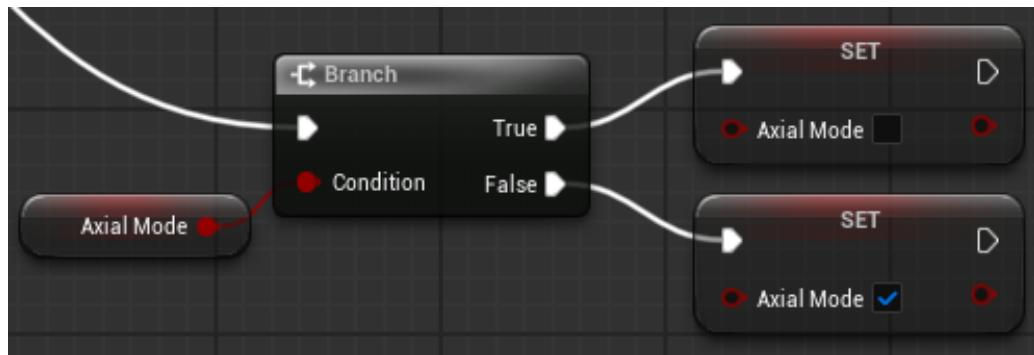


FIGURE 79 – Bouton X - Mode Déplacement

- Mode Rotation : réinitialise la rotation de l'objet choisi.

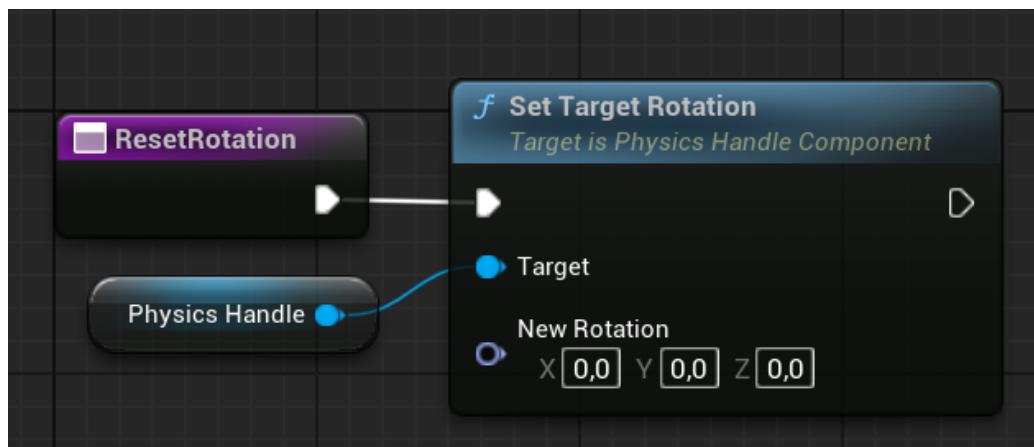


FIGURE 80 – Bouton X - Mode Rotation

- Mode Mise à l'échelle : réinitialise la taille de l'objet choisi.

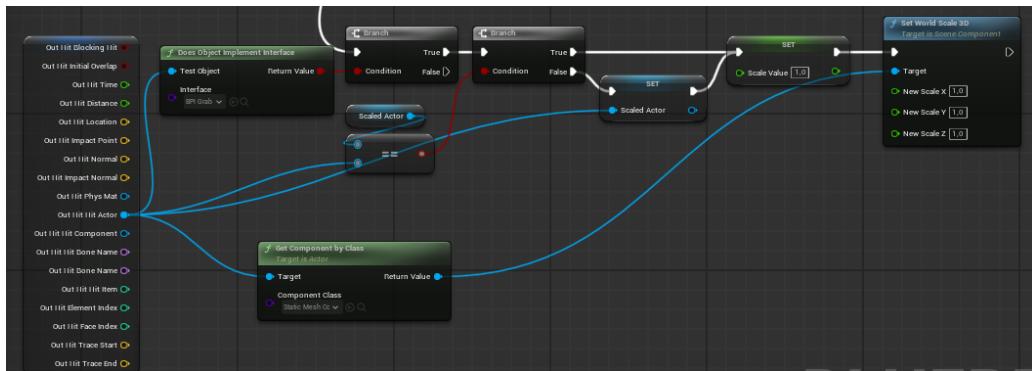


FIGURE 81 – Bouton X - Mode Mise à l'échelle

Bouton Y : Le bouton Y permet d'activer la grille lorsque le joueur se trouve dans le Mode Placement, Déplacement ou Rotation.

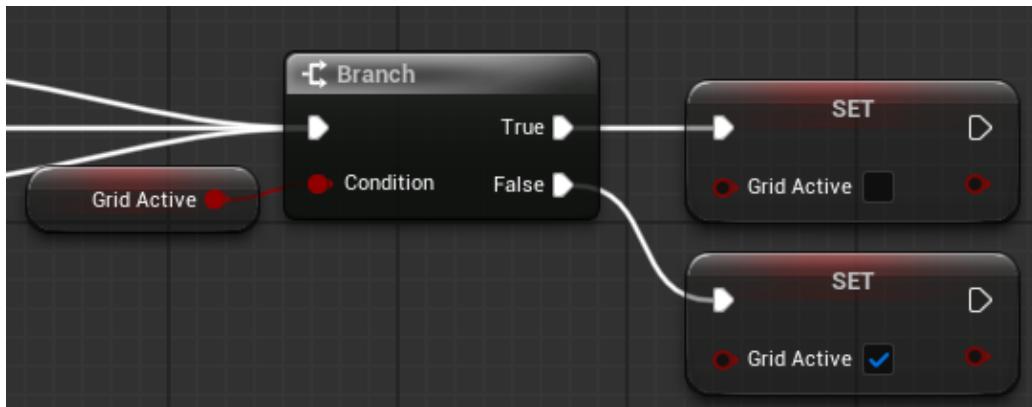


FIGURE 82 – Bouton Y

Bouton A :

- Mode Placement : Diminuer la valeur de la position de l'objet prévisualisé selon l'axe sélectionné

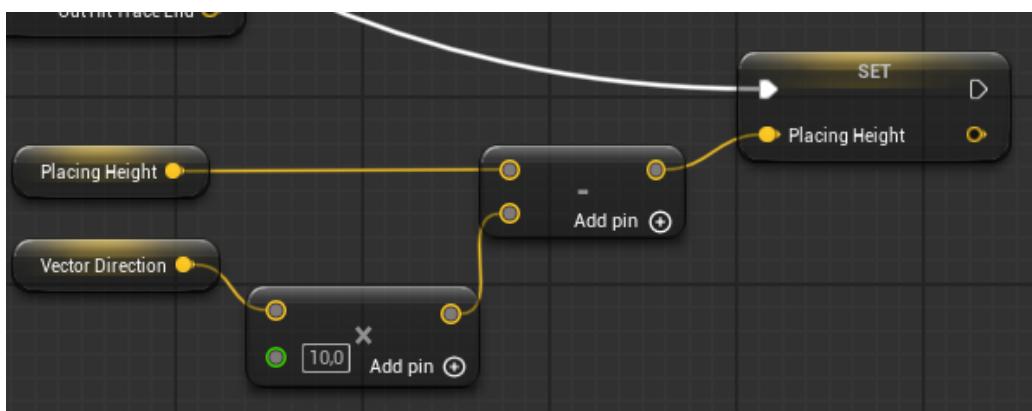


FIGURE 83 – Bouton A - Mode Placement

- Mode Lumière : Diminuer l'intensité de la lumière pointée

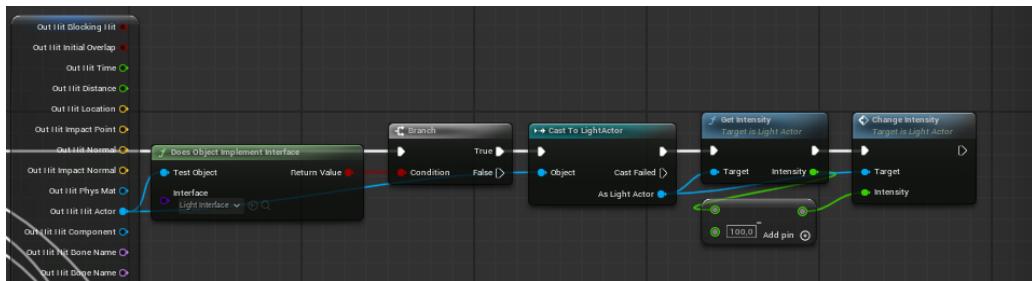


FIGURE 84 – Bouton A - Mode Lumière

- Mode Normal : Interagir avec les objets. Par exemple, ouvrir/fermer une porte ou allumer/éteindre une lampe

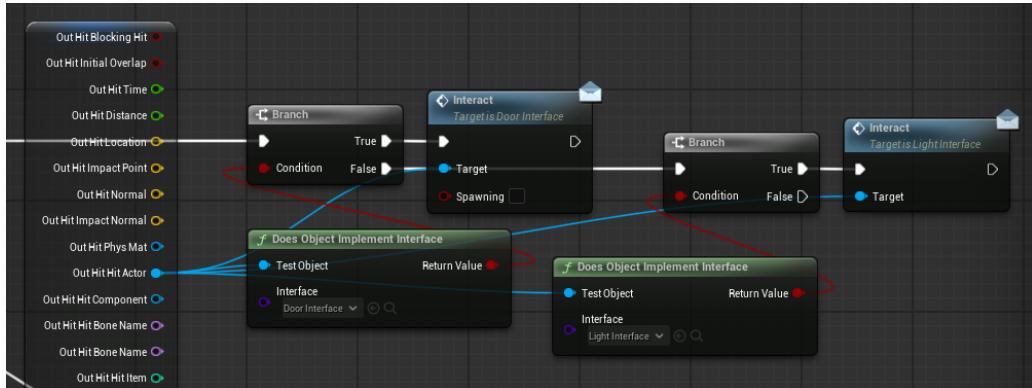


FIGURE 85 – Bouton A - Mode Normal

- Mode Déplacement : Selon la préférence choisie, Rapprocher l'objet vers soi ou le déplacer selon la grille

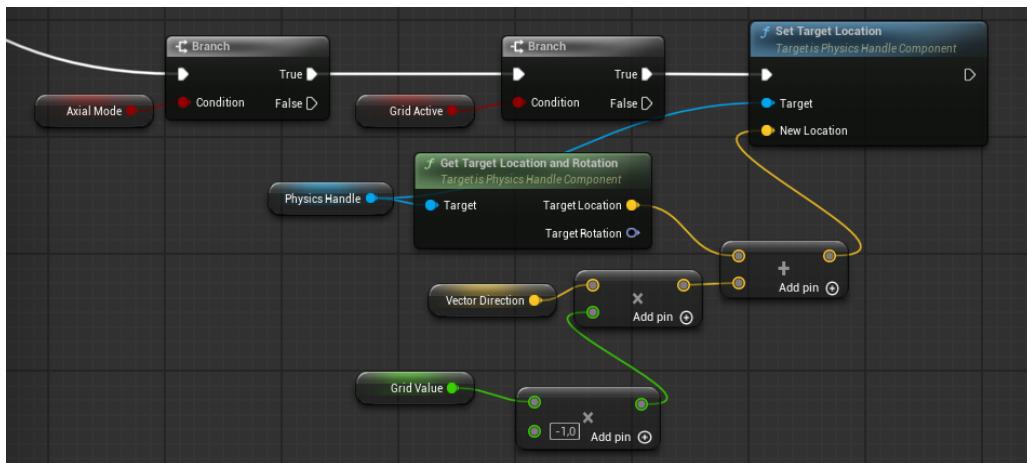
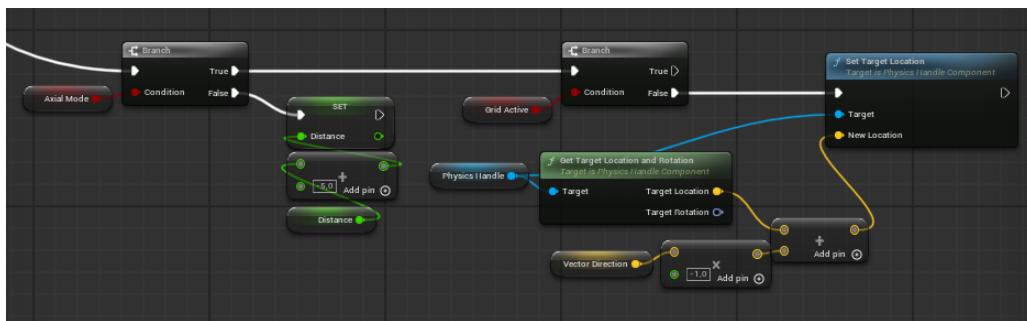


FIGURE 86 – Bouton A - Mode Déplacement

— Mode Rotation : Tourner l'objet selon l'axe Z (vertical) dans le sens trigonométrique

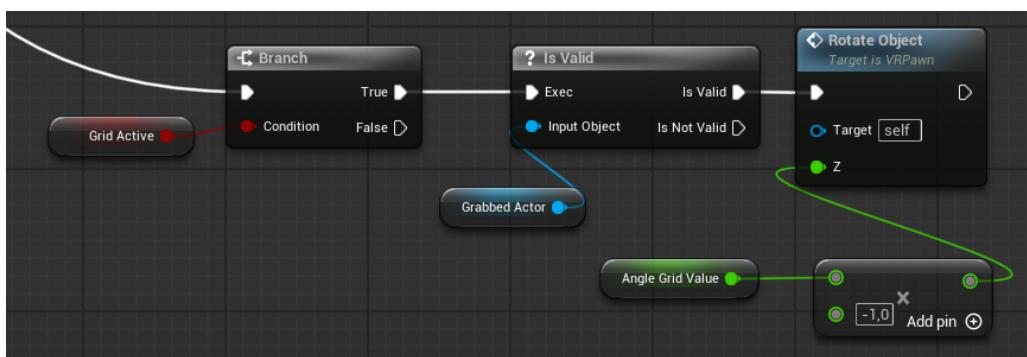
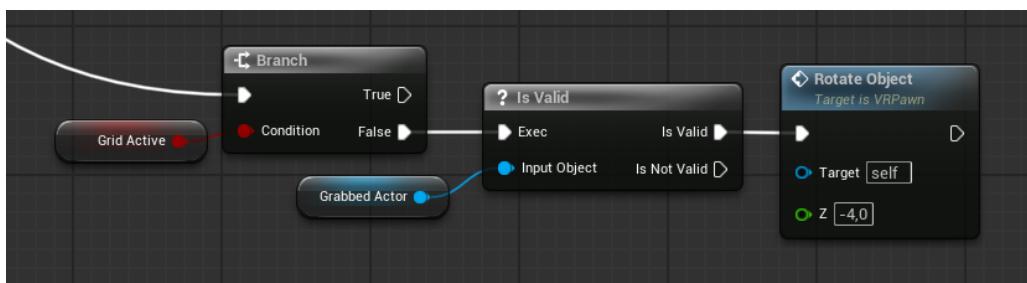


FIGURE 87 – Bouton A - Mode Rotation

— Mode Mise à l'échelle : Rétrécir l'objet

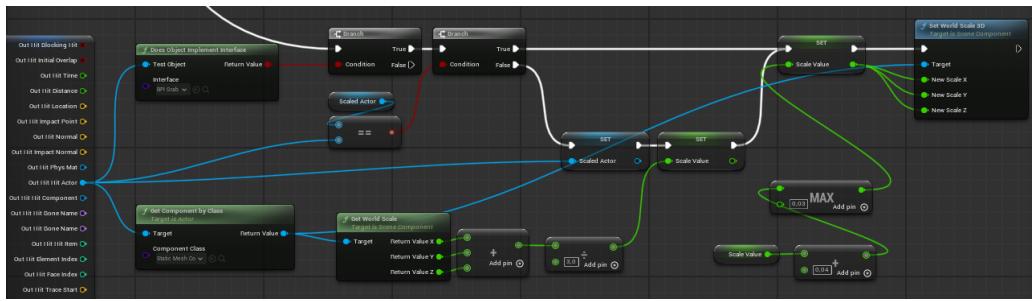


FIGURE 88 – Bouton A - Mode Mise a l'échelle

Bouton B :

- Mode Placement : Augmenter la valeur de la position de l'objet prévisualisé selon l'axe sélectionné. La logique est similaire à celle du bouton A (Figure 83).
- Mode Lumière : Augmenter l'intensité de la lumière pointée. La logique est similaire à celle du bouton A (Figure 84).
- Mode Déplacement : Selon la préférence choisie, éloigner l'objet de soi ou le déplacer selon la grille. La logique est similaire à celle du bouton A (Figure 86).
- Mode Rotation : Tourner l'objet selon l'axe Z (vertical) dans le sens anti-trigonométrique. La logique est similaire à celle du bouton A (Figure 87).
- Mode Mise à l'échelle : Agrandir l'objet. La logique est similaire à celle du bouton A (Figure 88).

Trigger gauche :

Ce bouton permet d'annuler et revenir dans le mode normal.

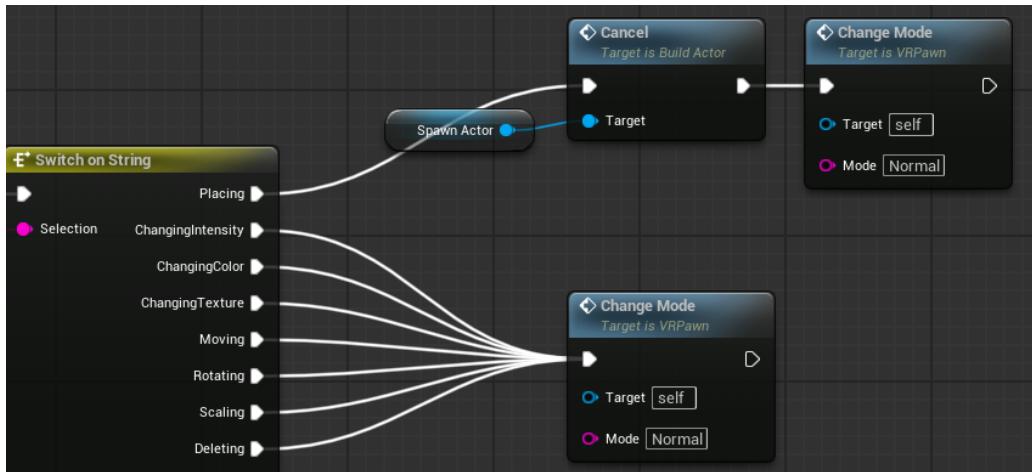


FIGURE 89 – Trigger gauche

Trigger droit :

- Mode Placement : placer l'objet prévisualisé.

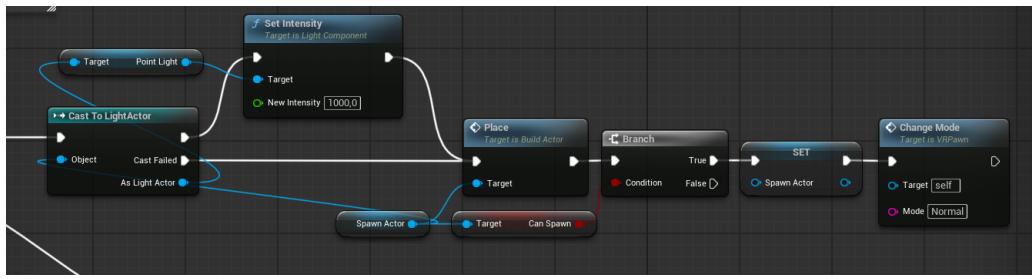


FIGURE 90 – Trigger droit - Mode Placement

— Mode Lumières : Changer la couleur de la lumière selon celle choisie dans le menu.

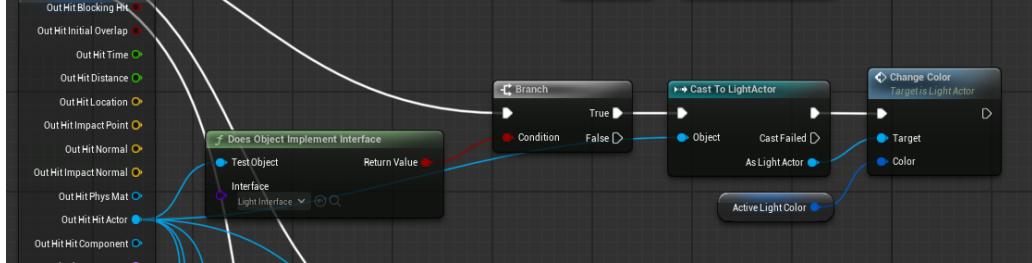


FIGURE 91 – Trigger droit - Mode Lumières

— Mode Suppression : Supprimer l'objet.

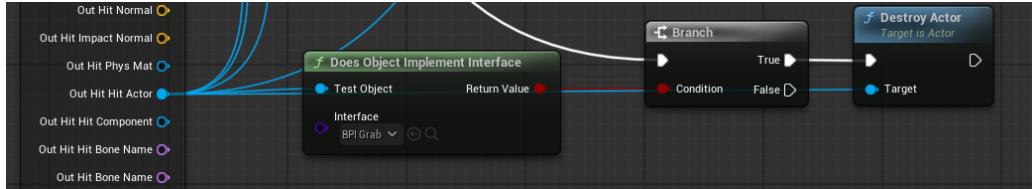


FIGURE 92 – Trigger droit - Mode Suppression

— Changement de texture : Changer la texture de l'objet selon celle choisie dans le menu.

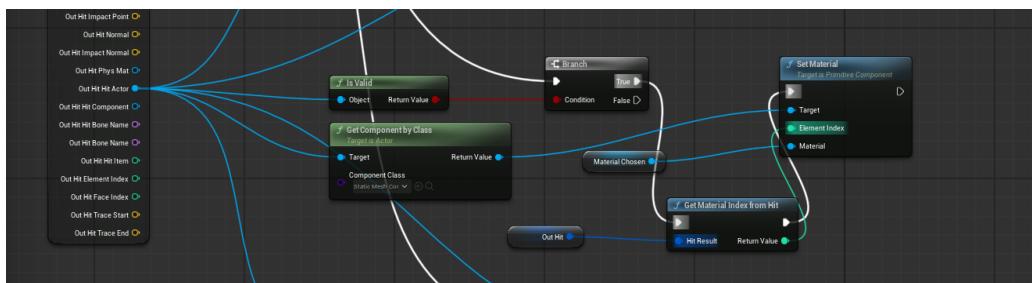


FIGURE 93 – Logique du changement de texture

Grip gauche : Ce bouton permet de changer d'axe en suivant l'ordre X- \downarrow Y- \downarrow Z pour les modes affectés (Placement et Déplacement). Il montre également pendant un court instant la direction de l'axe actif à l'utilisateur.

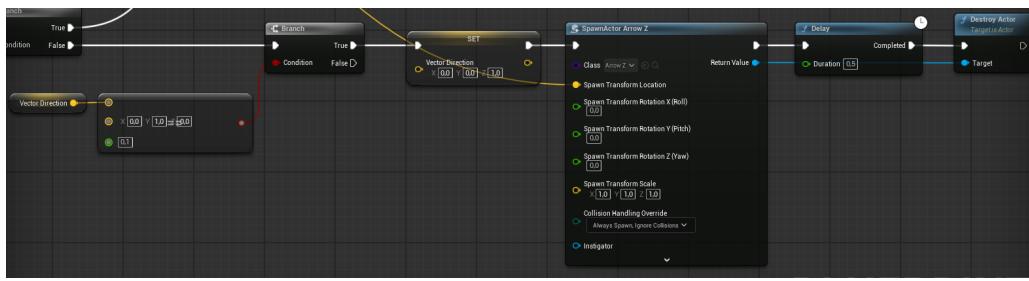


FIGURE 94 – Exemple de la logique de Grip gauche

Grip droit :

Dans le mode Normal, Déplacement, Rotation ou mise à l'échelle, permet d'attraper et de manipuler l'objet.

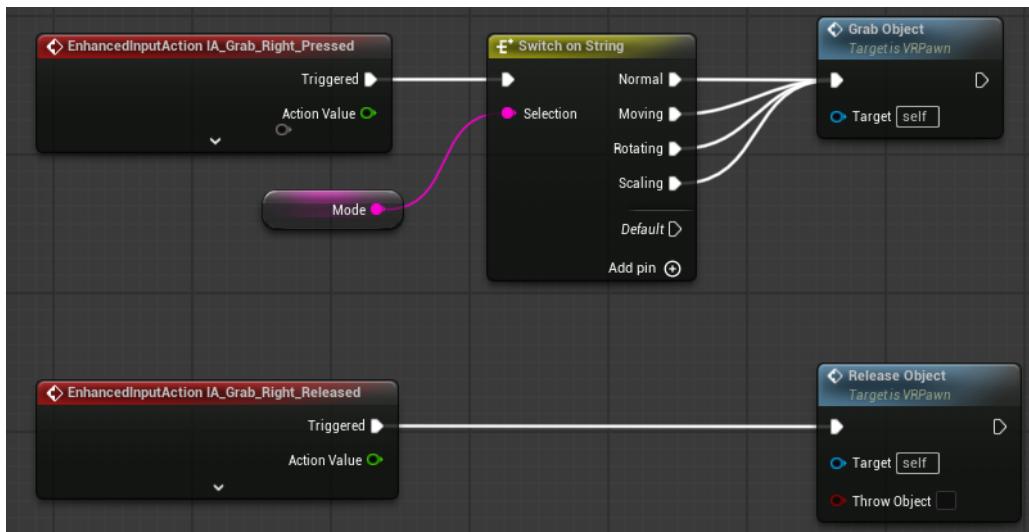


FIGURE 95 – Grip droit mode Normal, Déplacement, Rotation et Mise à l'échelle

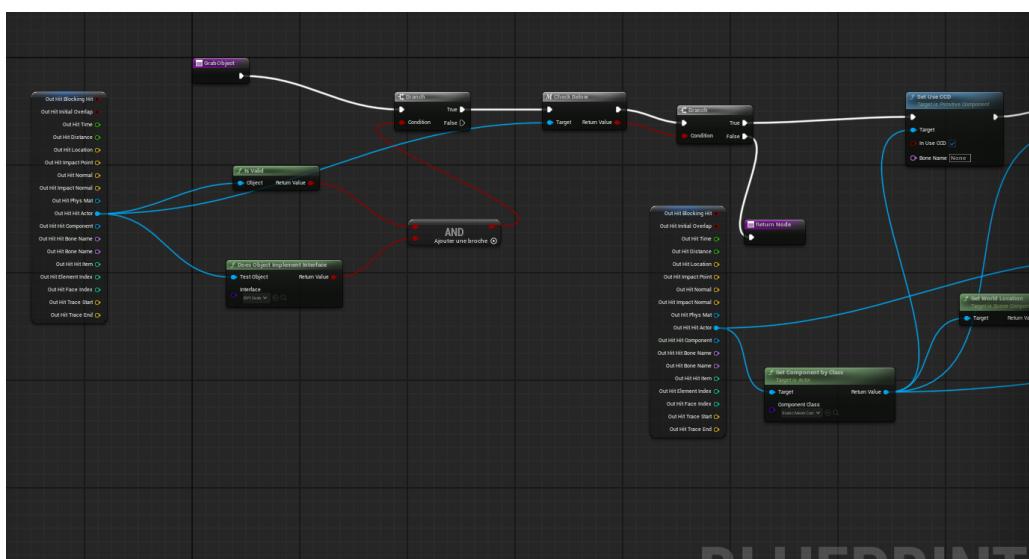


FIGURE 96 – Fonction GrabObject - Partie 1

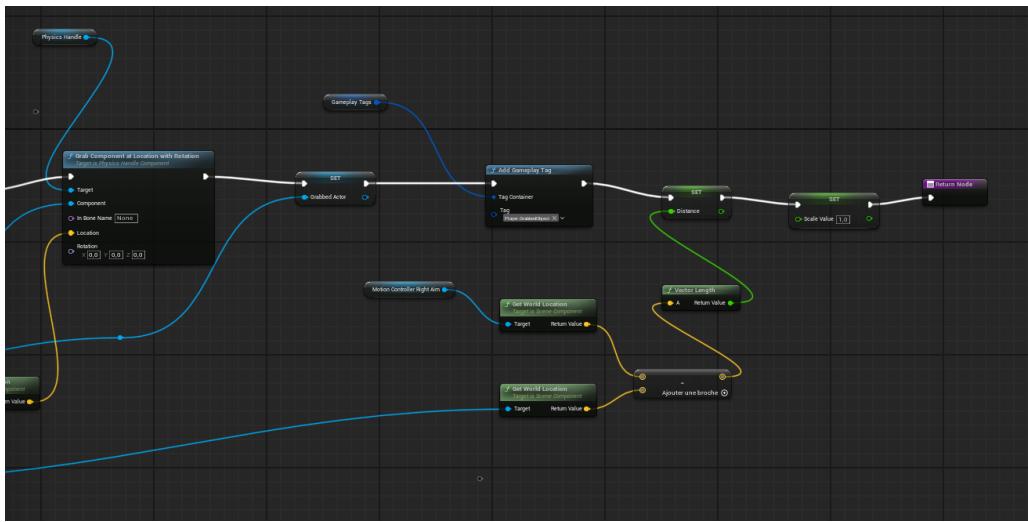


FIGURE 97 – Fonction GrabObject - Partie 2

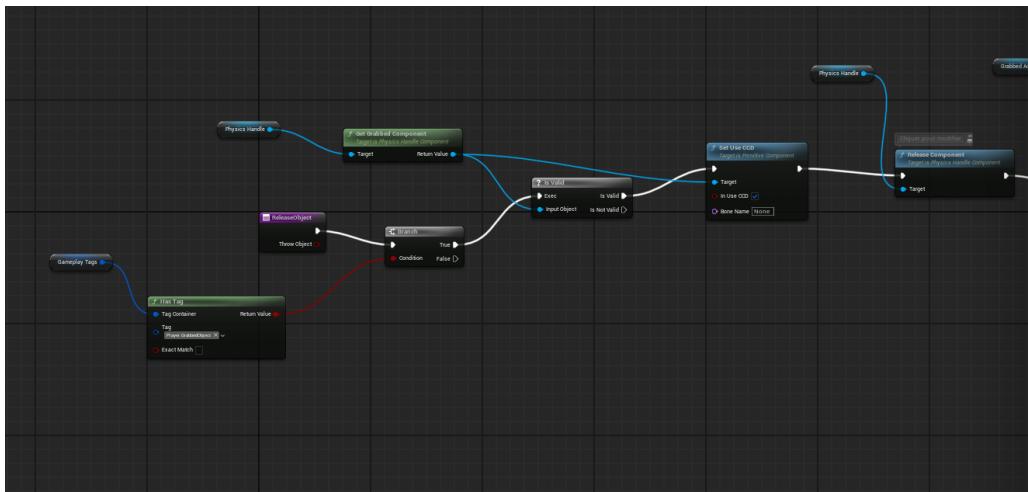


FIGURE 98 – Fonction ReleaseObject - Partie 1

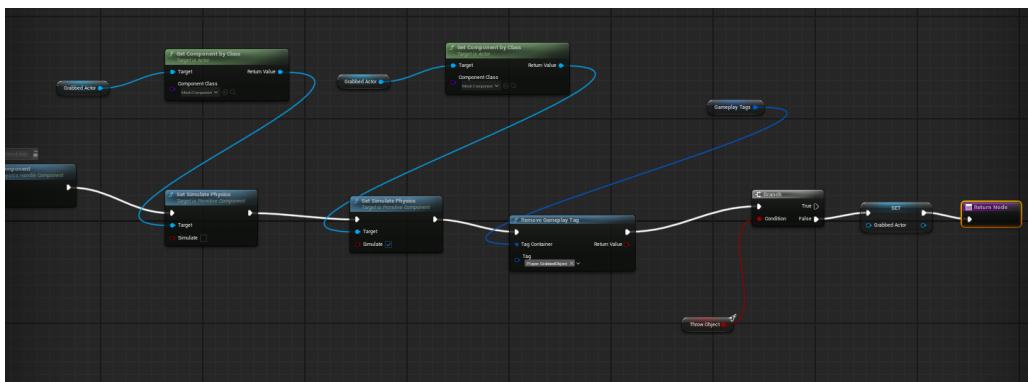


FIGURE 99 – Fonction ReleaseObject - Partie 2

Dans le mode Placement, permet de bloquer l'objet prévisualisé à sa position lors de l'appel.

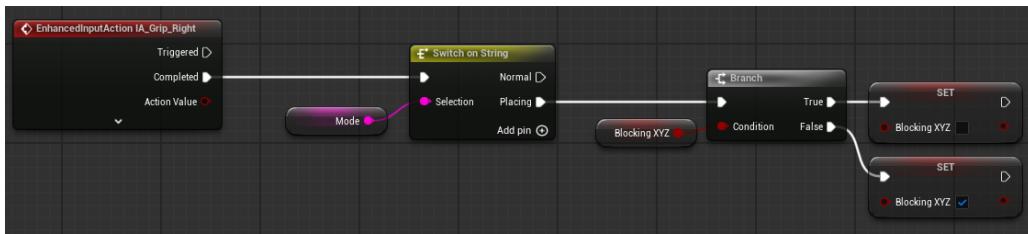


FIGURE 100 – Grip droit - Mode Placement

C Fonctions Blueprint C++

Pour notre projet, nous avons créé plusieurs BluePrints personnalisés.

- GetFBXAndOBJFilesInFolder : cette fonction prend en entrée un chemin d'accès relatif d'un dossier et donne comme sortie tous les fichiers FBX et OBJ présents dans celui-ci

```

1  #include "MyBlueprintFunctionLibrary.h"
2  #include "Components/StaticMeshComponent.h"
3  #include "Engine/StaticMesh.h"
4
5  int32 UMyBlueprintFunctionLibrary::GetMaterialIndexFromHit(const FHitResult& HitResult)
6  {
7      UStaticMeshComponent* MeshComp = Cast<UStaticMeshComponent>(HitResult.GetComponent());
8      if (!MeshComp || !MeshComp->GetStaticMesh())
9      {
10         return -1;
11     }
12
13     // Indice du triangle (primitive) touché
14     int32 TriangleIndex = HitResult.FaceIndex;
15
16     // Récupère le LOD 0
17     const FStaticMeshLODResources& LOD = MeshComp->GetStaticMesh()->GetRenderData()->LODResources[0];
18
19     for (int32 SectionIndex = 0; SectionIndex < LOD.Sections.Num(); ++SectionIndex)
20     {
21         const FStaticMeshSection& Section = LOD.Sections[SectionIndex];
22
23         // Calcul de l'ID du premier triangle de la section
24         int32 StartPrim = Section.FirstIndex / 3;
25         int32 NumPrim = Section.NumTriangles;
26
27         if (TriangleIndex >= StartPrim
28             && TriangleIndex < StartPrim + NumPrim)
29         {
30             return Section.MaterialIndex;
31         }
32     }
33
34     return -1;
35 }
36
37
38

```

FIGURE 101 – Fonction GetFBXAndOBJFilesInFolder

- GetMaterialIndexFromHit : cette fonction prend en entrée une line trace et donne en sortie l'index de l'élément du matériau touché. Elle est utilisée pour pouvoir faire les changements de textures multiples.

```

1  #include "MyBlueprintFunctionLibrary.h"
2  #include "Components/StaticMeshComponent.h"
3  #include "Engine/StaticMesh.h"
4
5  int32 UMyBlueprintFunctionLibrary::GetMaterialIndexFromHit(const FHitResult& HitResult)
6  {
7      UStaticMeshComponent* MeshComp = Cast<UStaticMeshComponent>(HitResult.GetComponent());
8      if (!MeshComp || !MeshComp->GetStaticMesh())
9      {
10         return -1;
11     }
12
13     // Indice du triangle (primitive) touché
14     int32 TriangleIndex = HitResult.FaceIndex;
15
16     // Récupère le LOD 0
17     const FStaticMeshLODResources& LOD = MeshComp->GetStaticMesh()->GetRenderData()->LODResources[0];
18
19     for (int32 SectionIndex = 0; SectionIndex < LOD.Sections.Num(); ++SectionIndex)
20     {
21         const FStaticMeshSection& Section = LOD.Sections[SectionIndex];
22
23         // Calcul de l'ID du premier triangle de la section
24         int32 StartPrim = Section.FirstIndex / 3;
25         int32 NumPrim = Section.NumTriangles;
26
27         if (TriangleIndex >= StartPrim
28             && TriangleIndex < StartPrim + NumPrim)
29         {
30             return Section.MaterialIndex;
31         }
32     }
33
34     return -1;
35 }
36
37
38

```

FIGURE 102 – Fonction GetMaterialIndexFromHit

- ImportScene : cette fonction est nativement présente dans le plugin AssimpLibrary mais nous l'avons modifié pour pouvoir importer en une fois des objets contenant plusieurs meshes. Pour pouvoir faire ça, on prend les positions relatives de chaque mesh et on les assemble, ce qui nous permet de reconstruire l'objet en un seul mesh.

```

360     UAScene* UASceneFunctionLibrary::ImportScene(FString FileName, UObject* WorldContextObject, int Flags, bool DisableAutoSpaceChange)
361     {
362         Assimp::DefaultLogger::set(new EAassimlStream());
363
364         aiProcess_Triangulate
365         | aiProcess_GenSmoothNormals
366         | aiProcess_OptimizeMeshes
367         | aiProcess_PreserveVertexOrder
368         | aiProcess_OptimizeMeshes
369         | aiProcess_OptimizeGraph;
370
371         if (!DisableAutoSpaceChange)
372         {
373             Flags |= aiProcess_MakeLeftHanded;
374         }
375
376         const struct aiScene* Scene = aiImportFile(TCHAR_TO_UTF8(*FileName), (unsigned int)Flags);
377
378         if (!Scene)
379         {
380             UE_LOG(LogAssimp, Error, TEXT("Error importing scene: %s", *FString(UTF8_TO_TCHAR(aiGetLastErrorString()))));
381             return NULL;
382         }
383
384         // Construction de la scène (attention, ne pas encore remplir OwnerMeshes)
385         UAScene* NewScene = NewObject<UAScene>();
386         NewScene->FullFilePath = FileName;
387
388         // Fusion manuelle des meshes
389         int TotalVertices = 0;
390         int TotalFaces = 0;
391         for (unsigned int i = 0; i < Scene->mNumMeshes; i++)
392         {
393             TotalVertices += Scene->mMeshes[i]->mNumVertices;
394             TotalFaces += Scene->mMeshes[i]->mNumFaces;
395         }
396
397         // Création d'un mesh fusionné
398         UAScene* CombinedMesh = new UAScene();
399         CombinedMesh->mNumVertices = TotalVertices;
400         CombinedMesh->mVertices = new FVector[TotalVertices];
401         CombinedMesh->mNormals = new FVector[TotalVertices];
402         CombinedMesh->mTextureCoordinates = new FVector[TotalVertices];
403         CombinedMesh->mTextureCoordinates[0] = 2;
404
405         CombinedMesh->mFaces = TotalFaces;
406         unsigned int VertexOffset = 0;
407         unsigned int FaceOffset = 0;
408
409         for (unsigned int i = 0; i < Scene->mNumMeshes; i++)
410         {
411             aiMesh* SrcMesh = Scene->mMeshes[i];
412
413             // Copy vertices
414             for (unsigned int v = 0; v < SrcMesh->mNumVertices; v++)
415             {
416                 CombinedMesh->mVertices[VertexOffset + v] = SrcMesh->mVertices[v];
417                 if (SrcMesh->mNormals)
418                     CombinedMesh->mNormals[VertexOffset + v] = SrcMesh->mNormals[v];
419                 if (SrcMesh->mTextureCoordinates)
420                     CombinedMesh->mTextureCoordinates[0][VertexOffset + v] = SrcMesh->mTextureCoordinates[0][v];
421             }
422
423             // Copy faces (with vertex index offset)
424             for (unsigned int f = 0; f < SrcMesh->mNumFaces; f++)
425             {
426                 aiFace* SrcFace = SrcMesh->mFaces[f];
427                 aiFace* DstFace = CombinedMesh->mFaces[FaceOffset + f];
428
429                 DstFace.mNumIndices = SrcFace.mNumIndices;
430                 DstFace.mIndices = new unsigned int[DstFace.mNumIndices];
431
432                 for (unsigned int idx = 0; idx < DstFace.mNumIndices; idx++)
433                 {
434                     DstFace.mIndices[idx] = SrcFace.mIndices[idx] + VertexOffset;
435                 }
436
437                 VertexOffset += SrcMesh->mNumVertices;
438                 FaceOffset += SrcMesh->mNumFaces;
439             }
440
441             // Crédit du propre UAScene
442             UAScene* UnifiedMesh = NewObject<UAScene>();
443             UnifiedMesh->Mesh = CombinedMesh;
444             UnifiedMesh->StaticMesh = NULL;
445             NewScene->OwnerMeshes.Add(UnifiedMesh);
446
447         }
448
449         return NewScene;
450     }

```

FIGURE 103 – Fonction ImportScene

- GetStaticMesh : cette fonction est également nativement présente dans le plugin AssimpLibrary. Nous l'avons modifié pour pouvoir implémenter une collision simple. En effet, il y a 2 types de collisions sur Unreal engine, les collisions complexes et les collisions simples. Lorsque nous importons un objet, il a nativement une collision complexe mais pas de collision simple. Le problème est de la collision complexe est qu'elle ne permet pas de simuler la physique, ce qui fait qu'une fois qu'un objet est posé, il est totalement statique. Pour palier à ce problème, nous rajoutons manuellement une collision simple .

```

132     UStaticMesh* UStaticMesh::GetStaticMesh()
133     {
134         if (StaticMesh)
135         {
136             return StaticMesh;
137         }
138
139         // --- Cr茅ation du MeshDescription
140         MeshDescription = UStaticMesh::CreateStaticMeshDescription(this);
141
142         MeshDescriptionBuilder.MeshDescriptionBuilder();
143         MeshDescriptionBuilder.SetMeshDescription(MeshDescription->GetMeshDescription());
144         MeshDescriptionBuilder.EnablePolyGroups();
145         MeshDescriptionBuilder.SetGridLayout(1);
146
147         TArray<VertexInstancesID> VertexInstances;
148         VertexInstances.Add(InitializedMesh->Vertices);
149
150         for (unsigned int Index = 0; Index < Mesh->nVertices; Index++)
151         {
152             const FVector2D VertexID = MeshDescriptionBuilder.AppendVertices(allVector3DToVector(Mesh->nVertices[Index]));
153             const FVector2D VertexInstanceID = MeshDescriptionBuilder.AppendInstance(VertexInstances[Index]);
154             VertexInstances[Index] = Instance;
155
156             if (Mesh->HasNormals())
157             {
158                 MeshDescriptionBuilder.SetInstanceNormal(Instance, allVector3DToVector(Mesh->nNormals[Index]));
159             }
160             else
161             {
162                 UE_LOG(LogKismet, Warning, TEXT("Normals not found, consider generating them with assimp"));
163             }
164
165             if (Mesh->HasTextureCoords(0))
166             {
167                 MeshDescriptionBuilder.SetInstanceUV(
168                     Instance,
169                     FVector2D(Mesh->nTextureCoords[0][Index].x, Mesh->nTextureCoords[0][Index].y),
170                     0
171                 );
172             }
173
174             const FPolygonGroupID PolygonGroup = MeshDescriptionBuilder.AppendPolygonGroup();
175
176             for (unsigned int i = 0; i < Mesh->nFaces; i++)
177             {
178                 const auto& Face = Mesh->nFaces[i];
179                 if (Face.nIndices > 2)
180                 {
181                     MeshDescriptionBuilder.AppendTriangle(
182                         VertexInstances[Face.nIndices[0]],
183                         VertexInstances[Face.nIndices[1]],
184                         VertexInstances[Face.nIndices[2]],
185                         PolygonGroup
186                     );
187                 }
188             }
189
190             // --- Construction du StaticMesh
191             StaticMesh = NewObject(this);
192             StaticMesh->GetStaticMaterials().Add(FStaticMaterial());
193
194             USceneObject::FBuildMeshDescriptionsParams MeshDescriptionsParam;
195             MeshDescriptionsParam.bBuildCollision = false; // on g茅e la collision manuellement
196             MeshDescriptionsParam.bBuild = true;
197
198             TArray<UStaticMesh*> MeshDescriptions;
199             MeshDescriptions.Replace(MeshDescription->GetMeshDescription());
200             StaticMesh->BuildFromMeshes(MeshDescriptions, MeshDescriptionsParam);
201
202             // --- Cr茅ation du BodySetup pour la collision
203             StaticMesh->CreateBodySetup();
204             BodySetup = StaticMesh->GetBodySetup();
205
206             if (BodySetup)
207             {
208                 // Vide toute collision simple pr茅existante
209                 BodySetup->AggGeom.ConvexElms.Empty();
210                 BodySetup->AggGeom.Softbodies.Empty();
211                 BodySetup->AggGeom.SphereElms.Empty();
212                 BodySetup->AggGeom.SpryElms.Empty();
213
214                 // --- G茅n茅ration d'un convex hull 脿 partir de tous les sommets
215                 FMeshDescription* MeshDesc = MeshDescription->GetMeshDescription();
216                 TArray<Vector> Convverts;
217                 Convverts.Reserve(MeshDesc->Vertices.Num());
218
219                 for (const FVector2D VID : MeshDesc->Vertices().GetElementIDs())
220                 {
221                     // GetVertexPosition retourne un FVector3f et on convertit en FVector
222                     const FVector3f Pos3f = MeshDesc->GetVertexPosition(VID);
223                     const FVector Pos1f(Pos3f);
224                     Convverts.Add(Pos1f);
225
226                     if (Convverts.Num() == 3)
227                     {
228                         FConvexElm ConvexElm;
229                         ConvexElm.VertexData = Convverts;
230                         Convverts.UpdateLastBox();
231                         BodySetup->AggGeom.ConvexElms.Add(ConvexElm);
232                     }
233
234                     // Utiliser les collisions simples par d茅faut (auto-g茅n茅r茅e convex + traces per poly)
235                     BodySetup->CollisionTraceFlag = ECollisionTraceFlag::CTF_UseSimpleAndComplex;
236
237                     // Reconstruire les donn茅es physiques
238                     BodySetup->IsValidatePhysicsData();
239                     BodySetup->CreatePhysicsMeshes();
240
241                 }
242
243                 // --- Marquer le mesh comme modifi茅 pour que le moteur prenne en compte la nouvelle collision
244                 StaticMesh->Modify();
245                 if (WITH_EDITOR)
246                 {
247                     StaticMesh->PostEditChange();
248                 }
249             }
250
251             return StaticMesh;
252         }
253     }

```

FIGURE 104 – Fonction GetStaticMesh