

PROHIBIT

BEAST



TECHNICAL DESIGN DOCUMENT

## ChangeLog

2023/06/16

- Suppression de la partie Lighting car obsolète et incohérente depuis longtemps.
- Ajout du Plug-Ins SimpleCombineMesh.
- Mise à jour de la partie UIRoot

2023/06/13

- Ajout des parties HDRP Decal Projector et Decal avec un FBX.
- Ajout des Plug-Ins PolyBrush et ProBuilder.

2023/06/08

- Mise à jour des Répertoires (ajout d'une Végétation dans OBJ) corrections de certaines fautes de frappe

2023/06/06

- Mise à jour des softwares, SpeedTree, corrections de certaines fautes de frappe

2023/06/02

- Mise à jour du chapitre "Outline"

2023/06/01

- ajout du sous chapitre "Cinemachine Path" dans le chapitre "Cinemachine"

2023/05/30

- ajout chapitre cinemachine

2023/04/21 :

- Ajout du chapitre "Le Sang" dans le chapitre "PNJ Ennemi"

2023/04/19 :

- Ajout du chapitre "Autres types de HUD" dans le chapitre "HUD"

2023/04/17 :

- Ajout du chapitre "L'alarme" dans le chapitre "PNJ Ennemi"

2023/04/14 :

- Ajout du chapitre "Détection" dans le chapitre "PNJ Ennemi"

2023/04/03 :

- Ajout de l'explication du code du HUD

2023/03/21 :

- Modification globale du TDD.

2023/03/17 :

- Ajout de la description de "Les Personnages Jouables (PJ) > L'inventaire".

2023/03/16 :

- Ajout de la description de "Le HUD > L'inventaire".

2023/02/22 :

- Ajout de la description des paramètres d'export du Rigging personnage.

2023/02/20 :

- Mise à jour de la description du processus de lightning/Reflection/VFX:PostProcess.

2023/02/17 :

- Ajout de la description du processus de lightning/Reflection/VFX/PostProcess.

2023/02/15 :

- Ajout de la description du processus de modélisation étape 2

2023/02/02 :

- Ajout de la description du processus de modélisation étape 1

2023/01/27 :

- Ajout du chapitre sur les curseurs
- Ajout du chapitre Personnages jouables (PC)

2023/01/26 :

- Ajout de la nomenclature des template de FSM dans le chapitre "Coding standard"

2023/01/25 :

- Ajout du chapitre "création des lignes de dialogues" dans la partie "PNJ Ennemis"

2023/01/19 :

- Ajout du sous chapitre 14.3. Versionning dans le chapitre coding standard.
- Remise en page de tout le dossier.
- MàJ du sommaire.

2023/01/16 :

- Ajout du chapitre 18 Intérieures

2023/01/13 :

- Modification du chapitre 15. PathFinding

2023/01/12 :

- Ajout du chapitre 16. Créer/Ajouter Template FSM.
- Ajout du chapitre 17. PNJ Ennemis.
- Modification du tableau des variables.
- Ajout de Layer.

2023/01/12:

- Insertion du tableau des props et de screens

2023/01/05-06 :

- Insertion de la partie graphiste, ajout du titre

2022/06/07/12 :

- Ajout de la structure et du sommaire, et rédaction des parties connues (features, moteur, softwares...)

2022/05/12 :

- Création du TDD

# Sommaire

<b>1. Cahier des charges</b>	<b>6</b>
<b>2. Features</b>	<b>6</b>
<b>3. Choix du moteur</b>	<b>6</b>
<b>4. Les Plug-Ins</b>	<b>7</b>
<b>5. Core Game Engine</b>	<b>7</b>
<b>6. Limite hardware</b>	<b>8</b>
<b>7. L'Équipe</b>	<b>8</b>
<b>8. Répertoire et répartition</b>	<b>9</b>
<b>9. Nomenclature des variables [WIP]</b>	<b>11</b>
<b>10. Les différents soft utilisés</b>	<b>12</b>
<b>11. Les paramètres graphiques</b>	<b>13</b>
11.1. Nomenclature et Technical Requirement des assets	13
11.2. Sources des fichiers	14
11.3. Paramètres d'import	27
11.4. Shaders et FX	2529
11.5. Lights et Shadows	28
11.6. Lighting	29
11.7. Génération les lights maps	29
11.8. Reflection Probes	30
11.9. Light Probe Group	31
11.10. Post Process	31
11.11. Les particules VFX	32
11.12. Cloth	32
11.13. Paramètres d'export Modèles 3D	32
11.14. Paramètres d'export Rigg	33
11.15. La caméra	35
<b>12. Scene Management</b>	<b>42</b>
12.1. Les différents layers utilisés	42
12.2. Les différents tags utilisés	42
12.3. Collision des layers	43
<b>13. Coding Standard</b>	<b>44</b>
13.1. Nomenclature	44
13.2. Code couleurs	44
13.3. Versionning	45
<b>14. Pathfinding</b>	<b>45</b>
<b>15. Créer/Ajouter un Template FSM :</b>	<b>46</b>
<b>16. Les Personnages Jouables (PJ)</b>	<b>47</b>
16.1. Setup	47
16.1.1. Ajouter des Statistiques	48
16.2. Déplacements	48
16.2.1. Indiquer la destination :	48
16.2.2. Calculer le chemin :	48
16.2.3. Gestion des Mouvements	49

16.3. Path-Object	49
16.3.1. Les échelles	49
16.4. Sélection	49
16.4.1. Sélection manuelle	50
16.4.2. Sélection de groupé	50
16.5. Ajouter une compétence	50
16.6. Dialogues	51
16.6.1. Game object et composants	51
16.6.2. Code	51
16.7. Interactions	52
Port de corps	52
16.8. Compétences	54
16.8.1. Compétence de Corps à Corps (CàC)	55
16.8.2. Compétences de Tir	55
16.8.3. Compétences de Classe	56
16.8.3.1. Heal	56
16.8.3.2. Détection	57
16.8.3.3. Distraction	57
16.9. L'inventaire	57
<b>17. PNJ Ennemi :</b>	<b>60</b>
17.1. Création d'une ronde :	60
17.2. Détection des joueurs :	61
17.2.1. La vision	61
17.2.1.1. Général	61
17.2.1.2. FSM-Detectable-View	62
17.2.1.3. FSM-InFov	62
17.2.1.4. FSM-View	62
17.2.1.5. FSM-View-Helper	63
17.2.1.6. FSM-Alerte-View	63
17.2.1.7. FSM-Focus	63
17.2.1.8. FSM-Movement-ViewCone	64
17.2.2. L'odorat	64
17.2.2.1. Général	64
17.2.2.2. FSM-Detectable-Smell	65
17.2.2.3. FSM-Smell	65
17.2.2.4. FSM-Smell-Helper	65
17.2.2.5. FSM-Alerte-Smell	65
17.2.3. L'ouïe	66
17.3. Les feedback visuels	66
17.4. L'attaque	66
17.5. L'alarme	67
17.6. Le Sang	67
<b>18. See Through / Fade</b>	<b>68</b>
18.1. See Through :	68
18.2. Fade :	69

<b>Animations</b>	<b>69</b>
Système de compétences	69
<b>19. Le HUD</b>	<b>71</b>
19.1. Général	71
19.1.1. La Character List	72
19.1.2. La grande Icône	72
19.1.3. Les Compétences	72
19.1.4. L'inventaire	73
19.1.5. La Mini-Map	73
19.1.6. La Barre de Menu	73
19.1.7. La Barre des Features	74
19.2. L'UI Root	74
19.2.1. FSM-CharList	74
19.2.2. FSM-Color	74
19.2.3. FSM-GetSelf	75
19.2.4. FSM-HealthBars	75
19.2.5. FSM-Inventory	75
19.2.6. FSM-Inventory-Display	75
19.2.7. FSM-MissionFailed	75
19.2.8. FSM-SkillBar	76
19.2.9. FSM-Outline-Button	76
19.2.10. FSM-ChangeButton	76
19.2.11. FSM-Camera-Buttons	77
19.2.12. FSM-Journal-Button	77
19.2.13. FSM-Journal-Window	77
19.2.14. FSM-Pause-Menu	77
19.2.15. FSM-Time-Button	78
19.2.16. FSM-Crouch-Button	78
19.2.17. FSM-Options-Menu	78
19.2.18. FSM-ChangeButton-Eager	78
19.2.19. FSM-Action-Button	79
19.2.20. FSM-Loading-Screen	79
19.3. Autre types de HUD	79
19.3.1. Les barres de vie	79
19.3.2. Les icônes de cachette	80
19.3.3. Les icône d'ennemis	80
<b>20. Highlight</b>	<b>80</b>
<b>21. Outline</b>	<b>82</b>
Explication du Package	82
Outliner	83
Outlinable	83
Intégration	84
Intégration Outline d'un PC	85
Couleurs Outline des PCs	86
Intégration Outline d'un PNJ	86

Couleurs Outline des PNJs	88
Intégration Outline d'un PROPS	88
Le root des PROPS doit posséder le FSM nommé "FSM-Outline" avec le template "FSM-Outline-PROPS".	88
<b>22. Curseurs</b>	<b>89</b>
<b>23. Cinemachine</b>	<b>93</b>
Intégration des éléments	93
Explication des différents éléments	96
Virtual camera	96
Cinemachine Path	98
<b>24. PROPS</b>	<b>99</b>
Traps	99
FSM-Interactive	99
Guide FSM à coder	99
Items	100
<b>25. Model 3D</b>	<b>101</b>
25.1. Personnages	101
Personnages Principaux Gentils	101
Eliot Seaness	102
Malone B. Clonan	103
Frank E. Eager	104
UV's	105
Texture	106
Ennemis	107
Etape 1 : Blender	107
Vulture	108
Wolf	108
Bear	108
Etape 2 : Zbrush	109
Etape 3 : Low Poly	110
Etape 4 : Texture	111
Etape 5 : Rigging/skinning	112
25.2. Props	113
Specifics transports	113
25.3. OBJ	114
Urban Elements	114
25.4. Levels	115
Building:	115
Modulaire	116
Chaque module est unique et vient se connecter parfaitement avec les autres. Pour cela on utilise un processus particulier.	116
Texture	117
Chaque texture tiles de gauche à droite et de haut en bas.	117
25.5. GUI	118
<b>26. Jeu Similaire</b>	<b>120</b>

26.1. Desperados III :	120
26.2. Empire of sin :	121

## 1. Cahier des charges

Type de jeu : **Real Time Strategy, Stealth Tactics.**

Plateforme principale : **PC.**

Public cible : **Hardcore gamers 18+.**

## 2. Features

- 3D ;
- Caméra en plongée ;
- Beaucoup de personnages jouables ;
- Plusieurs dizaines de personnages non jouables dans un niveau ;
- Grands décors remplis de détails et d'objets ;
- Des jeux de lumières clairs-obscur très contrastés ;
- Cutscenes ;
- Doublages ;
- VFX ;
- Décors interactifs ;
- Beaucoup d'états à faire subir aux ennemis ;
- Un HUD interactif ;
- Système de sauvegarde (et sauvegarde rapide) ;
- Système de chargement rapide ;
- Pathfinding.

## 3. Choix du moteur

La plateforme principale étant le PC, le choix du moteur se porte sur Unity pour ses plug-ins PlayMaker et PoolKit.

Unity permet de gérer plus simplement les animations, le rendering, le moteur PhysX et le système audio.

La version doit être récente mais terminée (2021 ou antérieure) et en Long Term Support (LTS).

La version choisie est la **2021.3.16 LTS**

## 4. Les Plug-Ins

**Playmaker 1.9.6**

Outil de codage visuel.

**Poolkit**

Outil de gestion de Spawn et de Pool.

**NGUI Next-Gen UI**

Outil de création de HUD.

**NGUI Custom Playmaker Actions + Playmaker NGUI Scripts**

Ajoutent des actions playmaker pour NGUI.

**(Advanced PlayerPrefs Window)**

Outil pour gérer les playerprefs (sauvegarde).

**Easy Performant Outline**

Plug-In permettant de créer des Outlines.

**KWS Water System HDRP**

Effet de simulation d'eau.

**See-Through Shader**

Plug-In permettant la disparition des bâtiments.

**PolyBrush**

Plug-In permettant de peindre multiples textures (4) sur un même Material.

**ProBuilder**

Plug-In permettant de s'informer en temps réel de la mesure précise des Assets.

**SimpleMeshCombine**

Plug-In permettant de combiner un groupe d'assets pour "faker" un rendu post-build et alléger la charge de travail du moteur.

## 5. Core Game Engine

Projet Unity : HDRP

Mode de rendu : Différé

Frame size : Full HD (1920x1080)

Cascade Shadow : 2

Skinning : 4 bones

Vertex Index : 16 bits

## 6. Limite hardware

### **Configuration Minimale**

- OS : Windows 7
- CPU : Intel Core I5
- GPU : Intel Pentium 7th Gen
- RAM : 4 Gb
- Espace de stockage demandé : ~20 Gb (basé sur Desperados 3)

### **Configuration Recommandée**

- OS : Windows 10
- CPU : Intel Core I5
- GPU : Nvidia GTX 1080
- RAM : 4 Gb
- Espace de stockage : ~20 Gb

## 7. L'Équipe

### **Cheffe de projet :**

Thea Cornil

### **Graphistes :**

Ambre Perrichon

Bastien Chambat

Clément Boyer

Enola Sonzogni

Juliette Male

Kiryan Rodrigues

Nicolas Grosjean

Nicolas Pourny

Quentin Lagniez

Théo Darnois

Théo Mathon Protat

Thomas Hoffmann

Victor Balestrat

### Concepteurs :

Adrien Lafourcade  
 Alexis Picq  
 Clément Pretot  
 Fabien Lagrange  
 Homan Cornet  
 Samuel Grosjean  
 Tanguy Rolland  
 Tomy Tournier

## 8. Répertoire et répartition

Assets	Directory	Sub-directory	Users
/_ANIMATIONS	/Characters		Adrien
	/Props		
/_MODELS	/Characters	/Accessories	
		/Civilians	
		/Enemies	Enola, Juliette, Nicolas G.
		/PC	Ambre, Clément B., Théo M.
		/Building	Ambre, Clément B., Théo M.
		/Decals	
		/Floor	
		/Obstacles	
		/SpecificsElements	
		/Structure	
		/Vegetation	
	/OBJ	/ConstructionElements	
		/IndustrialElements	
		/SpecificsTransports	
		/Universals	

		/UrbanElements	
		/Vehicles	
	/Props	/HidingPlace	Nicolas G.,
		/PathObjects	
		/SpecificsElements	
		/SpecificsTransport s	Nicolas P.,
		/Trap	
		/Universals	
		/Vehicles	Quentin, Victor
<b>/Resources</b>	/Art	/Characters	Tomy
		/Levels	Alexis, Homan
		/OBJ	Alexis, Homan
		/Props	Alexis, Homan
		/VFX	Alexis, Homan
		/Game	
		/Manager	Tomy, Adrien
		/Characters	Tomy, Adrien, Fabien, Clement P, Tanguy, Samuel
		/Levels	Alexis, Homan
		/OBJ	Tomy, Adrien, Fabien, Clement P, Tanguy, Alexis, Homan
		/Props	Tomy, Adrien, Fabien, Clement P, Tanguy, Alexis, Homan
		/UI	Samuel, Tanguy
		/VFX	
	<b>/_SCENES</b>	/Levels	Alexis, Homan
		/Menus	Samuel
<b>/_SOUNDS</b>	/Musics		Alexis
	/SFX		Alexis
<b>/_UI</b>	/DeathScreen		Samuel, Théo D
	/Fonts		Samuel, Théo D
	/InGameUI		Samuel, Théo D
	/LoadingScreen s		Samuel, Théo D
	/MainMenu		Samuel, Théo D

/PauseMenu	Samuel, Théo D
/SplashScreen	Samuel, Théo D
















[TDD-Prohibeabst-Repatriation](#)

## 9. Nomenclature des variables

La convention de nommage des variables est le camelCase.

Les variables correspondantes aux inputs (clavier) doivent se nommer comme suit : "buttonX". X étant la touche correspondante à l'input.

## 10. Les différents soft utilisés

-  Blender 3.3.1
-  Unity 2021.3.16
-  ZBrush 2020.1.1
-  Substance Painter 2022
-  Photoshop 2019
-  FreeFileSync
-  Google Drive
-  Google Docs
-  Google Slide
-  Google Sheets
-  Trello
-  Inkscape
-  PureRef
-  Midjourney
-  SpeedTree v8.4.2

## 11. Les paramètres graphiques

### 11.1. Nomenclature et Technical Requirement des assets

Nous avons choisi de suivre le modèle suivant pour la nomenclature :

>> VOIR ASSETS LIST DRIVE << : [AssetsList-Prohibeast](#)

- **Model 3D** : **.blend + .fbx**

Characters : **[Repertory]-[Name]**

*Exemples* : PC-BClonan ; Enemies-Wolf ; Civilians-Rabbit

GUI : **[Repertory]-[Name]**

*Exemple* : Cursors-Base

Autres : **[AssetType]-[Name]**

*Exemples* : PROPS-Trashcan ; OBJ-Bench ; LEVELS-SideWalk

> Pour les **.spp + .psd** : **[Name]** et si plusieurs objets sur la même texture : **Set-[Name]**

> Pour les **.unitypackage** : **[AssetType]-[Name]** ou **[AssetType]-[Nom Logique de l'Ensemble]**

- **Material** : **[Name]-MAT**

*Exemple* : Trashcan-MAT

- Si 2 materials sur 1 même Objet : PhoneBooth-Body-MAT + PhoneBooth-Glass-MAT

- Si plusieurs objets sur 1 même material : **Set-[Nom Logique de l'Ensemble]-MAT**

- **Dummy** : **Dummy-[Name]**

*Exemples* : Dummy-BClonan ; Dummy-RightHand ...

- **Rigg/Animation** : **ANIM/RIGG-[AssetType]-[Name]**

*Exemple* : RIGG-BClonan

- **Textures** : **[Name]-[TextureType]**

*Exemples* : Trashcan-BaseColor ; Trashcan-NormalMap ; Trashcan-MaskMap ;

Trashcan-Emissive ; Trashcan-AmbientOcclusion ; Trashcan-Metalness

- Si plusieurs objets sur 1 même planche d'UVs/ 1 même texture : **Set-[Nom Logique de l'Ensemble]-[TextureType]** → [Nom Logique...] = même que celui du -MAT

- **Sprites** : **HUD-[AssetType]-[Name]**

*Exemple* : HUD-BigIcon-BClonan

## 11.2. Sources des fichiers

Il est important que toute l'équipe suive les sources de leurs fichiers afin d'éviter des problèmes dans les paramètres d'import.

### Source des Objets 3D :

- Zbrush 2020.1.1 ;
- Blender 3.3.1.
- SpeedTree 8.4.2.

### Source des Textures :

- Substance 2022.
- Photoshop 2019.

### Sources du HUD :

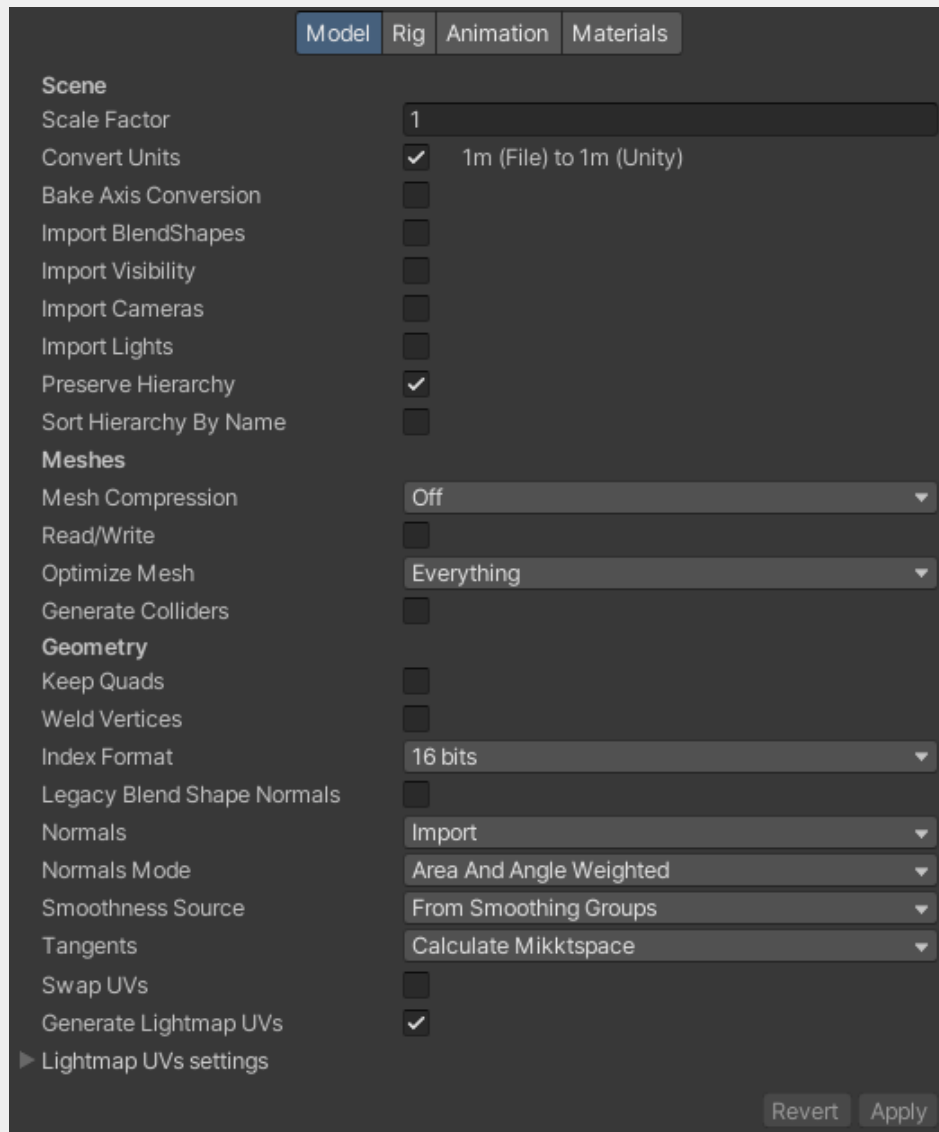
- Photoshop 2019.

## 11.3. Paramètres d'import

### **Objets 3D**

PROPS/ OBJ/ LEVELS

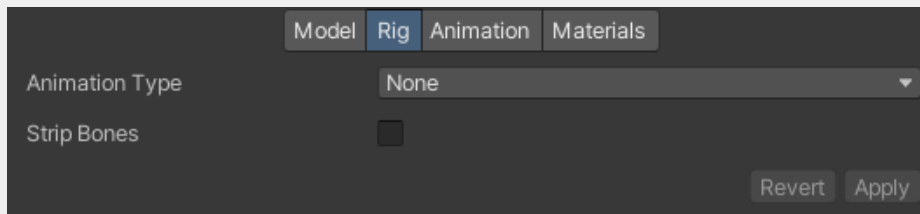
**MODEL**



- > **Bake Axis Conversion** : à décocher ! Mettre les axes du gizmo de la même orientation que ceux dans Unity
- > **Preserve Hierarchy** : à cocher !
- > **Sort Hierarchy By Name** : coché ou décoché
- > **Weld Vertices** : à décocher !
- > **Index Format** : "16 bits"
- > **Normals Mode** : "Area And Angle Weighted" par défaut, SI Midpoly peut-être mettre "Unweighted" selon l'aspect rendu dans Unity
- > **Smoothness Source** : "From Smoothing Groups"
- > **Generate Lightmap UVs** : à cocher par défaut SAUF si l'asset est amené à bouger dans la scène

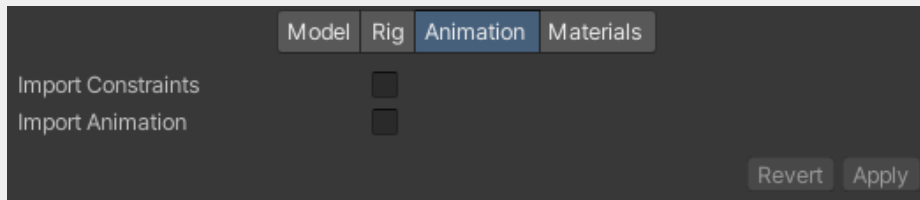
## RIGG

A décocher s'il n'y en a pas !

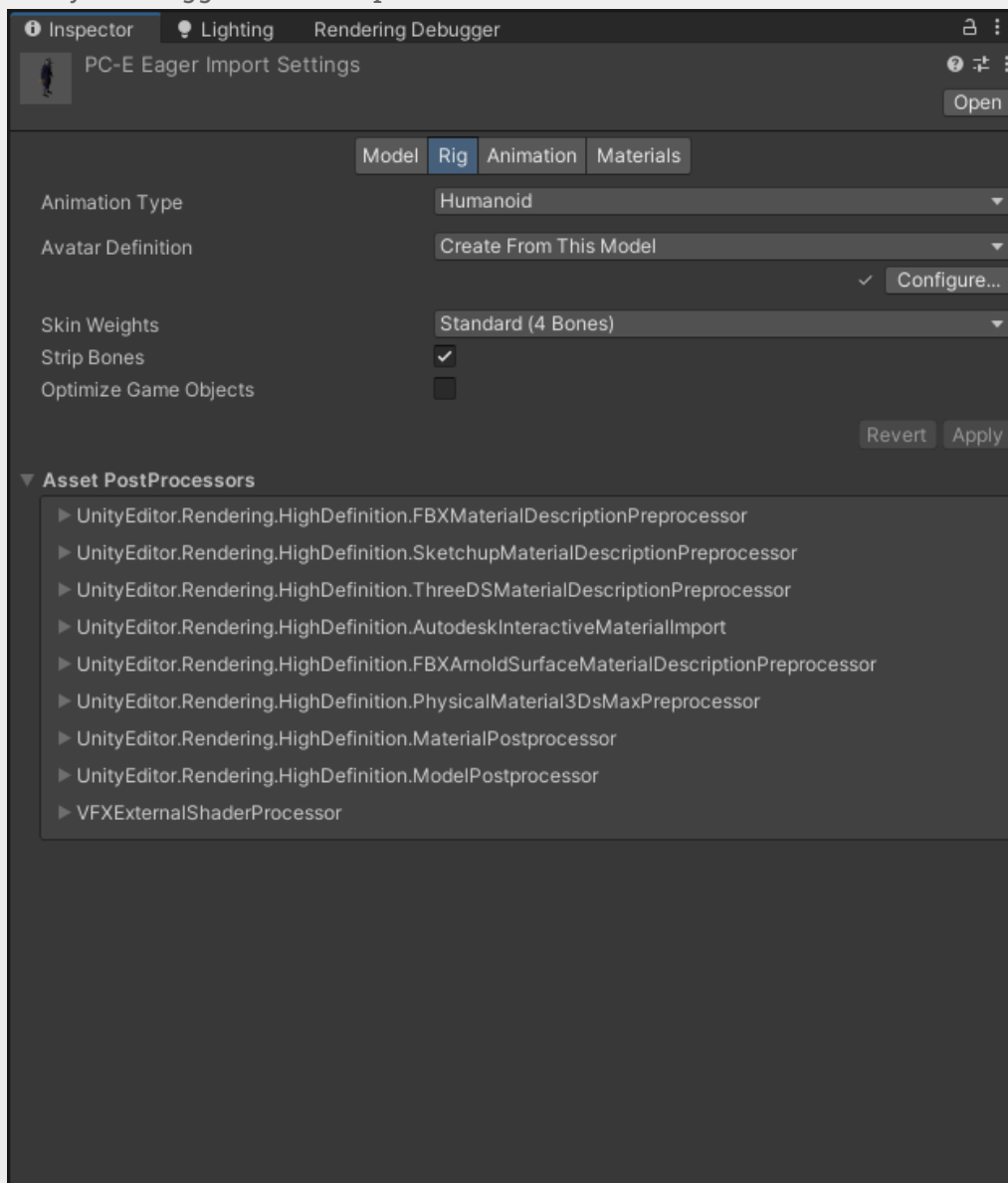


## Animation

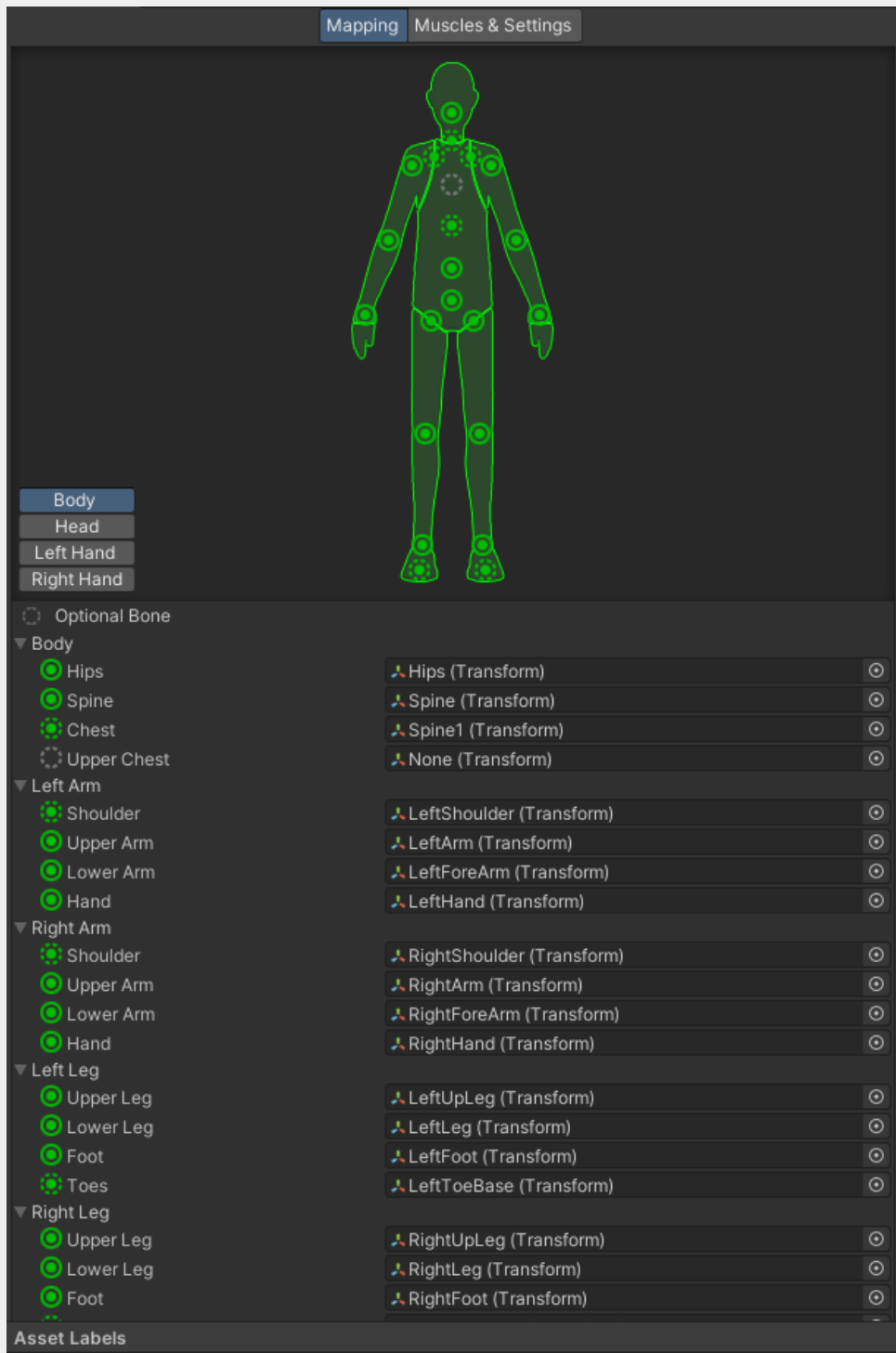
A décocher s'il n'y en a pas !



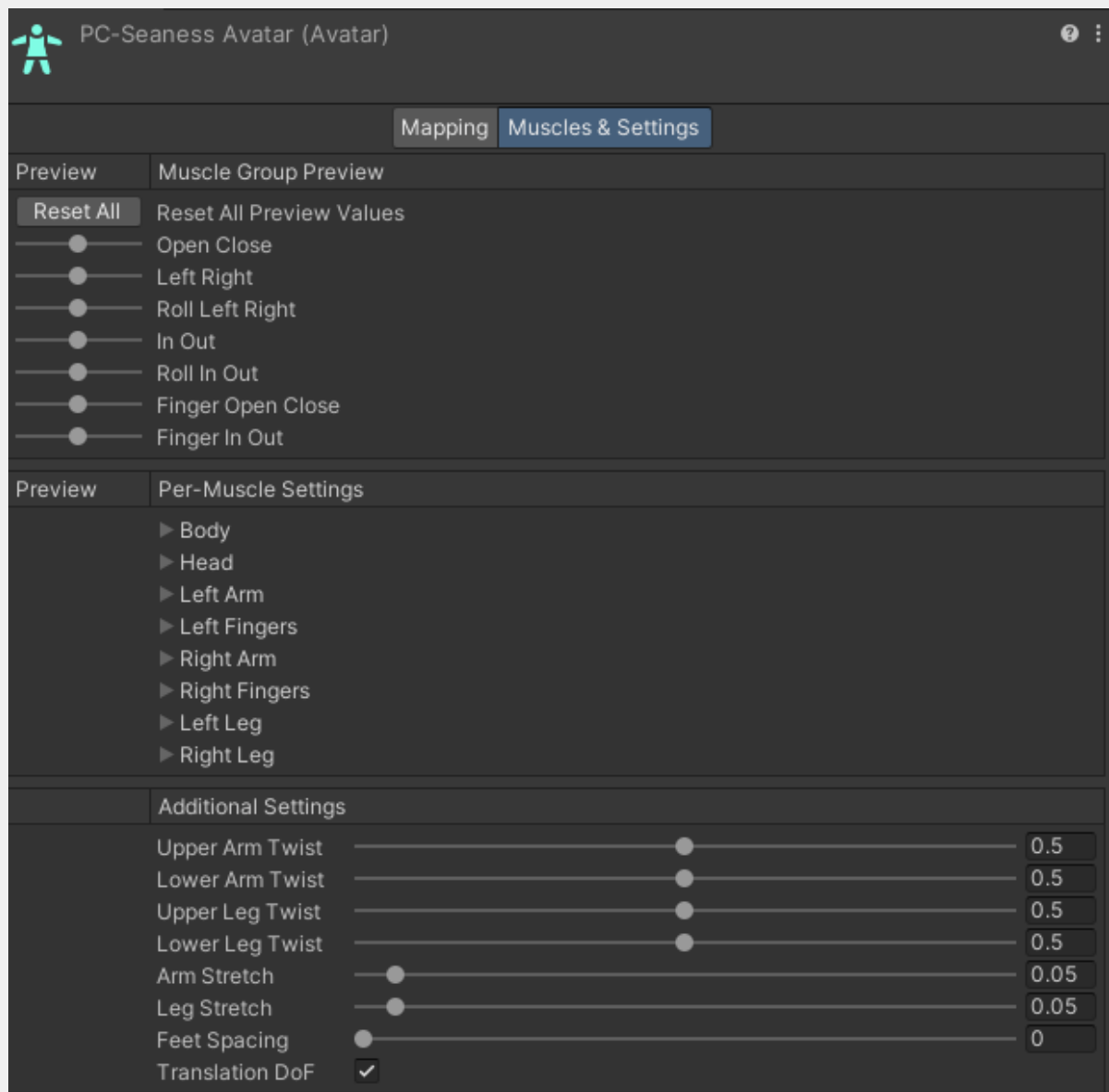
Si il y a un rigg : suivre les paramètres ci-dessous



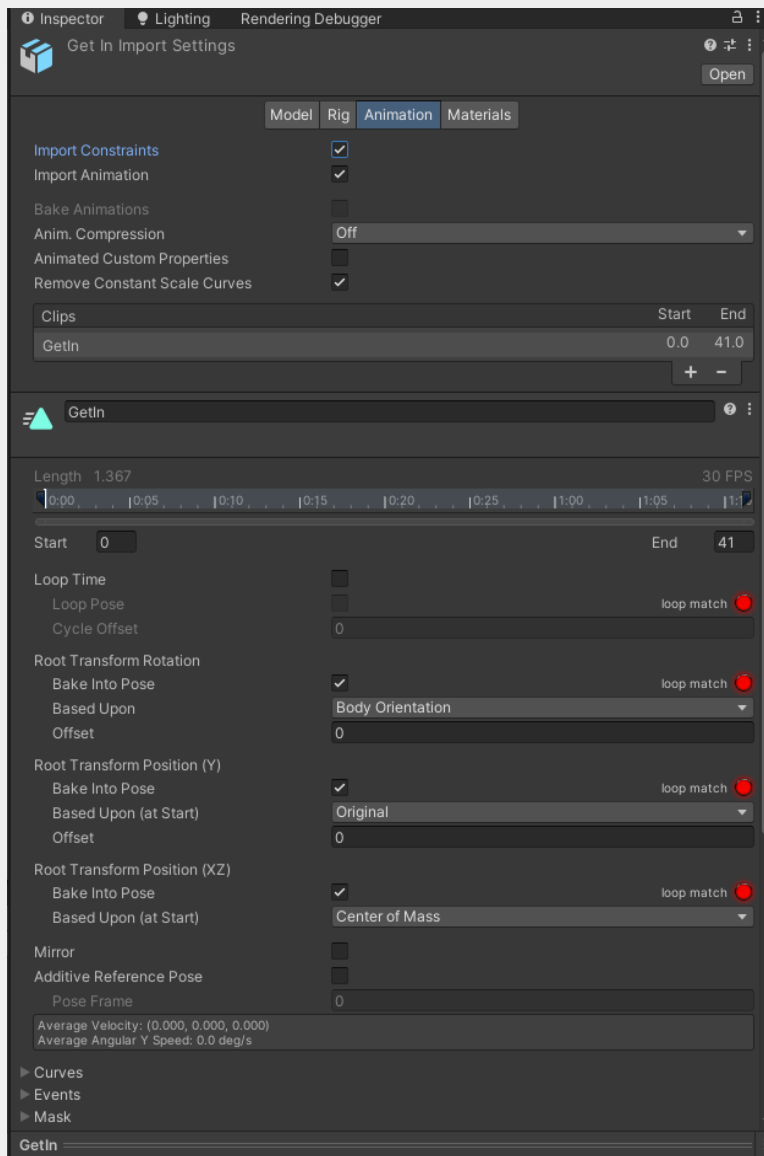
Dans l'onglet configuration vérifier qu'il n'y a pas d'erreurs et que les bones sont bien attribués



Dans l'onglet Muscles et Settings, cocher Translation DOF si le personnage ne possède que 2 spine.

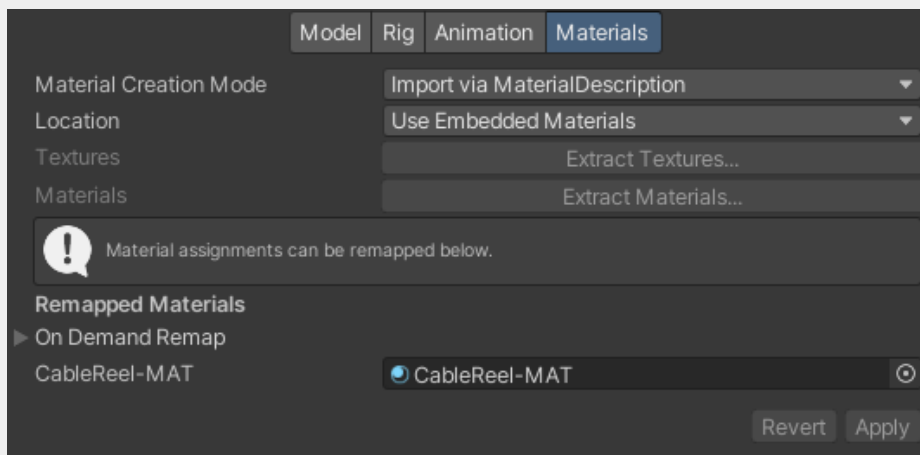


Pour les animations suivre les instructions ci-dessous



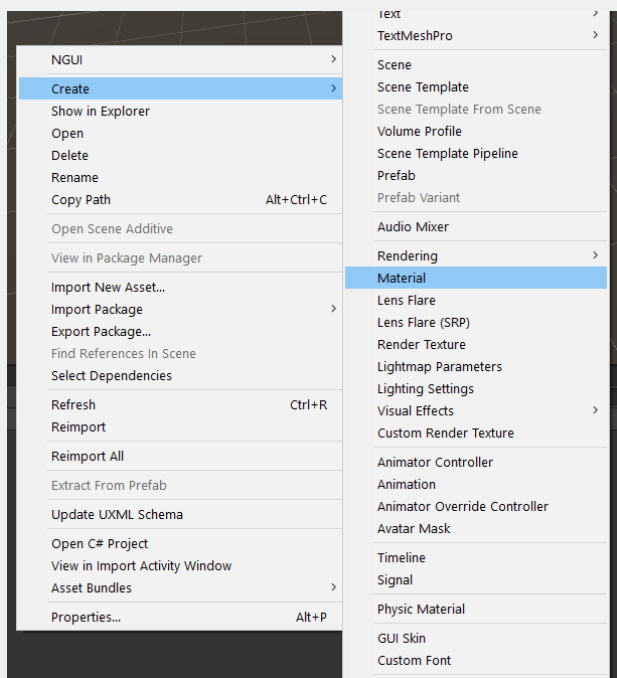
## Materials

Attention aux nomenclatures



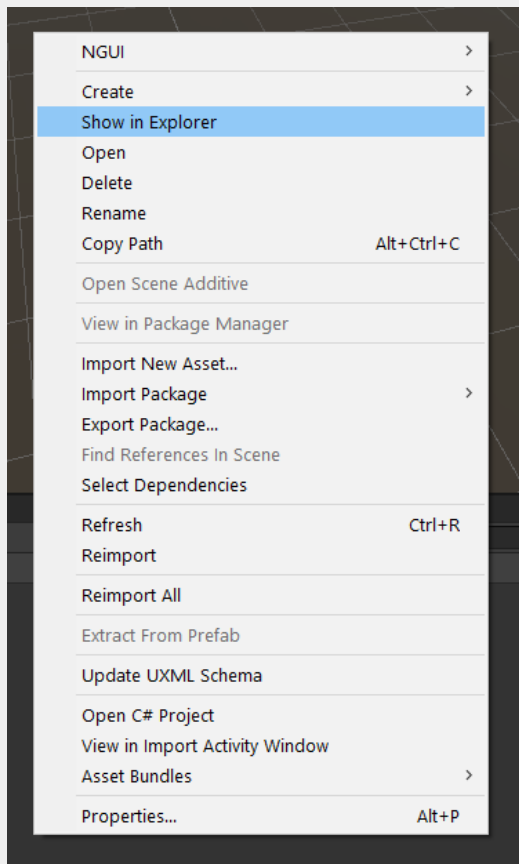
Il est désormais préférable de créer un nouveau Material (en remettant la bonne nomenclature !) que de faire “Extracts Materials” (pour éviter notamment qu’il y est une autre couleur sur la colorimétrie de la Base Map)

*Clic droit > Create > Material > A renommer et à attribuer*



## Mise à jour sur Unity d'assets déjà importés et paramétrés

*Clic droit > Show in explorer > Ctrl + C (Fichiers du pc) > Ctrl + V (Fichiers Unity) > “Remplacer le(s) fichier(s) dans la destination”*



## CHARACTERS

- Sur Blender AE (Anthropomorphic extension) concerne tous les bones qui sortent de la hiérarchie humaine.
- 3 Armatures : Les grands ont le skin RIGG-TC (Tall character), RIGG-MC (Mid-Character) pour les persos moyens et RIGG-WC (Women Character) pour la femme.

Import BlendShapes	<input type="checkbox"/>
Import Visibility	<input type="checkbox"/>
Import Cameras	<input type="checkbox"/>
Import Lights	<input type="checkbox"/>
Preserve Hierarchy	<input checked="" type="checkbox"/>
Sort Hierarchy By Name	<input checked="" type="checkbox"/>
<b>Meshes</b>	
Mesh Compression	Off
Read/Write	<input checked="" type="checkbox"/>
Optimize Mesh	Everything
Generate Colliders	<input type="checkbox"/>
<b>Geometry</b>	
Keep Quads	<input type="checkbox"/>
Weld Vertices	<input checked="" type="checkbox"/>
Index Format	Auto
Legacy Blend Shape Normals	<input type="checkbox"/>
Normals	Calculate
Normals Mode	Unweighted (Legacy)
Smoothness Source	From Smoothing Groups

<div>Model <b>Rig</b> Animation Materials</div>	
Animation Type	Humanoid
Avatar Definition	Create From This Model
	✓ <a href="#">Configure...</a>
Skin Weights	Standard (4 Bones)
Strip Bones	<input checked="" type="checkbox"/>
Optimize Game Objects	<input type="checkbox"/>
<div>Revert Apply</div>	

▼ Asset PostProcessors

## Textures

Texture Type	Default																		
Texture Shape	2D																		
sRGB (Color Texture)	<input checked="" type="checkbox"/>																		
Alpha Source	Input Texture Alpha																		
Alpha Is Transparency	<input type="checkbox"/>																		
Ignore PNG file gamma	<input type="checkbox"/>																		
<b>Advanced</b>																			
Non-Power of 2	ToNearest																		
Read/Write	<input type="checkbox"/>																		
Streaming Mipmaps	<input type="checkbox"/>																		
Virtual Texture Only	<input type="checkbox"/>																		
Generate Mip Maps	<input checked="" type="checkbox"/>																		
Border Mip Maps	<input checked="" type="checkbox"/>																		
Mip Map Filtering	Kaiser																		
Mip Maps Preserve Coverage	<input type="checkbox"/>																		
Fadeout Mip Maps	<input type="checkbox"/>																		
Wrap Mode	Repeat																		
Filter Mode	Trilinear																		
Aniso Level	1																		
<table border="1"> <thead> <tr> <th>Default</th> <th>Monitor</th> <th>Mobile</th> </tr> </thead> <tbody> <tr> <td>Max Size</td> <td>1024</td> <td></td> </tr> <tr> <td>Resize Algorithm</td> <td>Mitchell</td> <td></td> </tr> <tr> <td>Format</td> <td>Automatic</td> <td></td> </tr> <tr> <td>Compression</td> <td>High Quality</td> <td></td> </tr> <tr> <td>Use Crunch Compression</td> <td><input type="checkbox"/></td> <td></td> </tr> </tbody> </table>		Default	Monitor	Mobile	Max Size	1024		Resize Algorithm	Mitchell		Format	Automatic		Compression	High Quality		Use Crunch Compression	<input type="checkbox"/>	
Default	Monitor	Mobile																	
Max Size	1024																		
Resize Algorithm	Mitchell																		
Format	Automatic																		
Compression	High Quality																		
Use Crunch Compression	<input type="checkbox"/>																		

- > **Texture Type**: "Default" SAUF pour les NormalMap (= paramètre à son nom)
- > **Alpha Is Transparency**: à cocher seulement s'il y a de l'alpha
- > **Generate Mip Maps**: à cocher !
- > **Border Mips Maps**: à cocher SAUF pour les textures qui tiles
- > **Mip Map Filtering**: "Kaiser"
- > **Filter Mode**: "Trilinear"
- > **Max Size**: à voir selon asset

### **HDRP Decal Projector**

Même [paramètres d'imports](#) concernant les textures mise à part un détail. Il faut cocher "**Alpha Is Transparency**" s'il y a du jeu d'Opacity.

Shader du Material en **HDRP/Decal**.

**ATTENTION** à bien régler le slider de **Global Opacity** et de cocher/décocher les **Surfaces Options** en fonction des textures disponibles.

### **Decals avec un FBX**

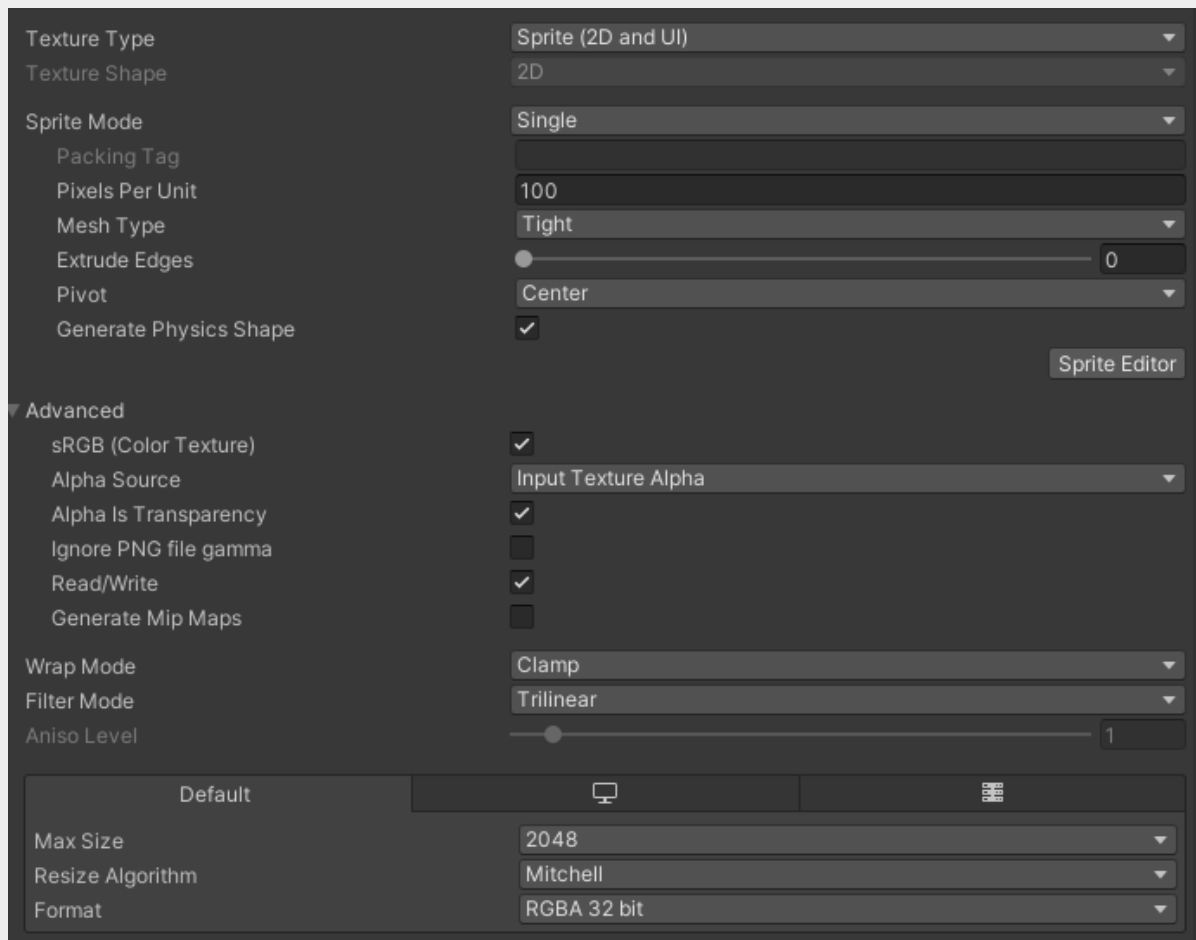
Même [paramètres d'imports](#) concernant le FBX (Plane).

Même [paramètres d'imports](#) concernant les textures mise à part un détail. Il faut cocher "**Alpha Is Transparency**" s'il y a du jeu d'Opacity.

Shader du Material en **HDRP/Lit** (Shader standard).

**ATTENTION** à bien cocher la case "**Alpha Clipping**" sur le Material s'il y a du jeu d'Opacity.

## HUD et Sprites

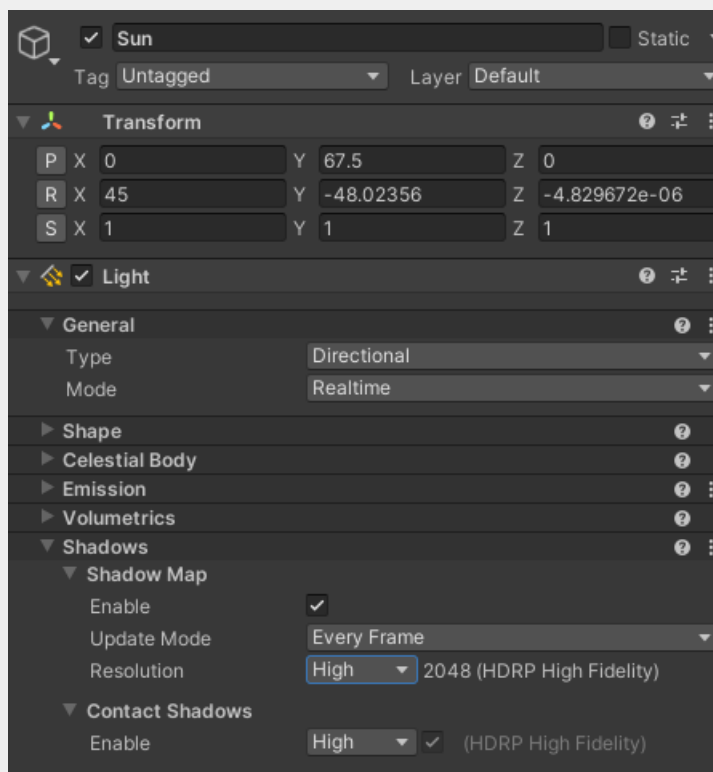


## 11.4. Shaders et FX

Lumière de jour :

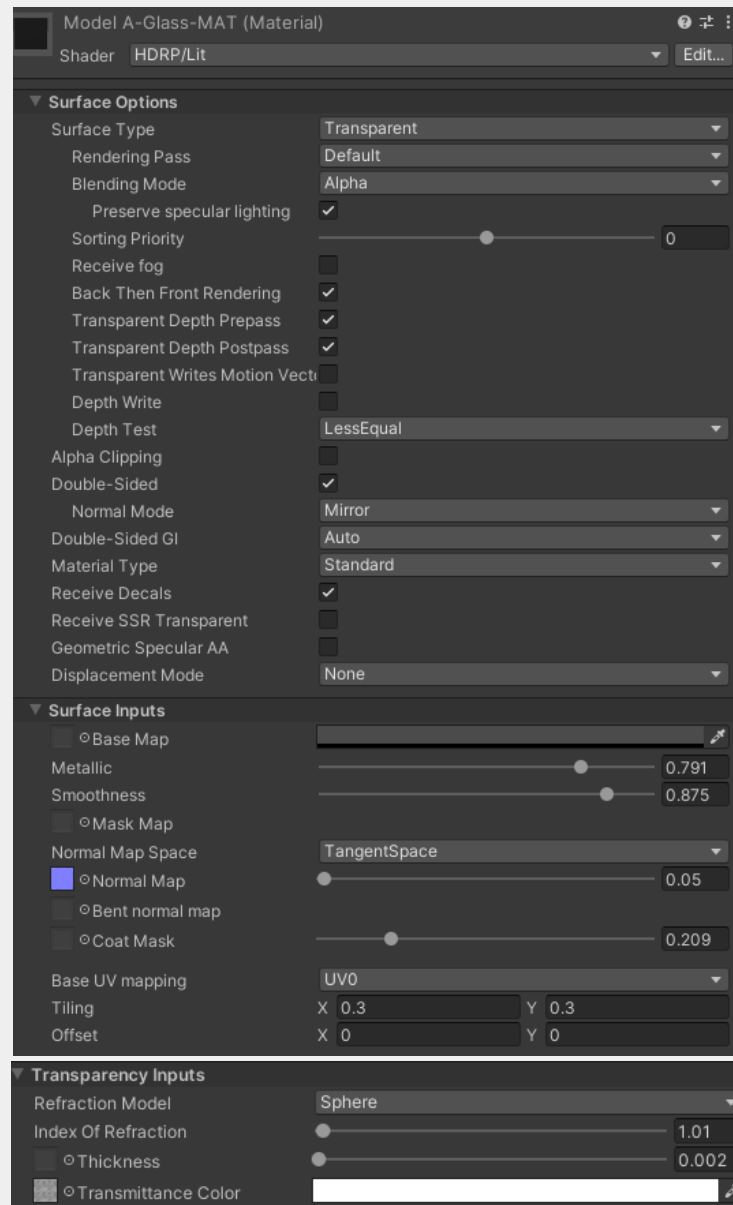
La lumière de jour provient principalement d'une **directional light** orientée de manière à faire des **ombres** visibles de haut.

Les **ombres** doivent être assez détaillées et smooth. La qualité des ombres par défaut est donc sur "High", bien que cela puisse être réglable dans les options par le joueur.

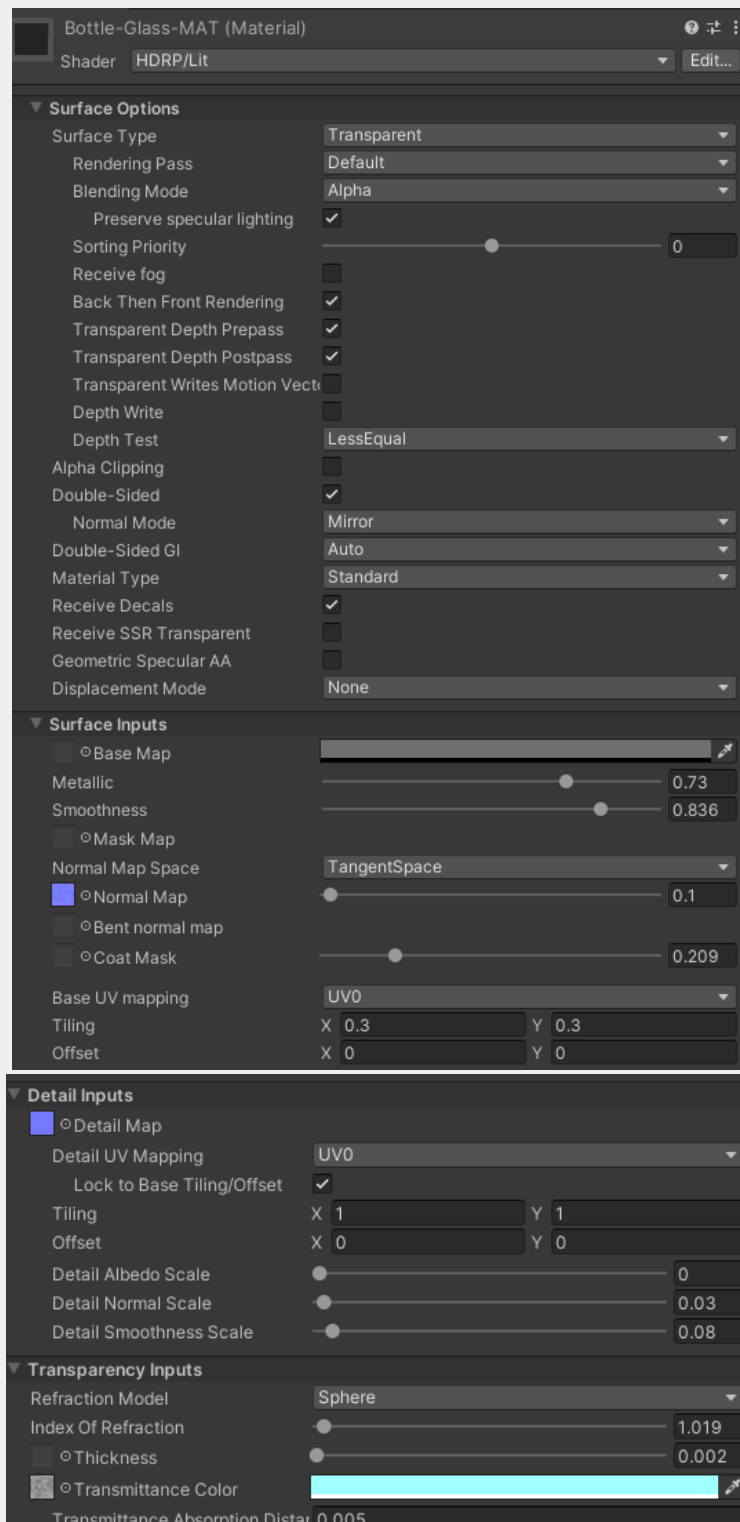


Le Shader Glass :

## Car-SharderGlass



## Bottle-SharderGlass



## 11.5. Lights et Shadows

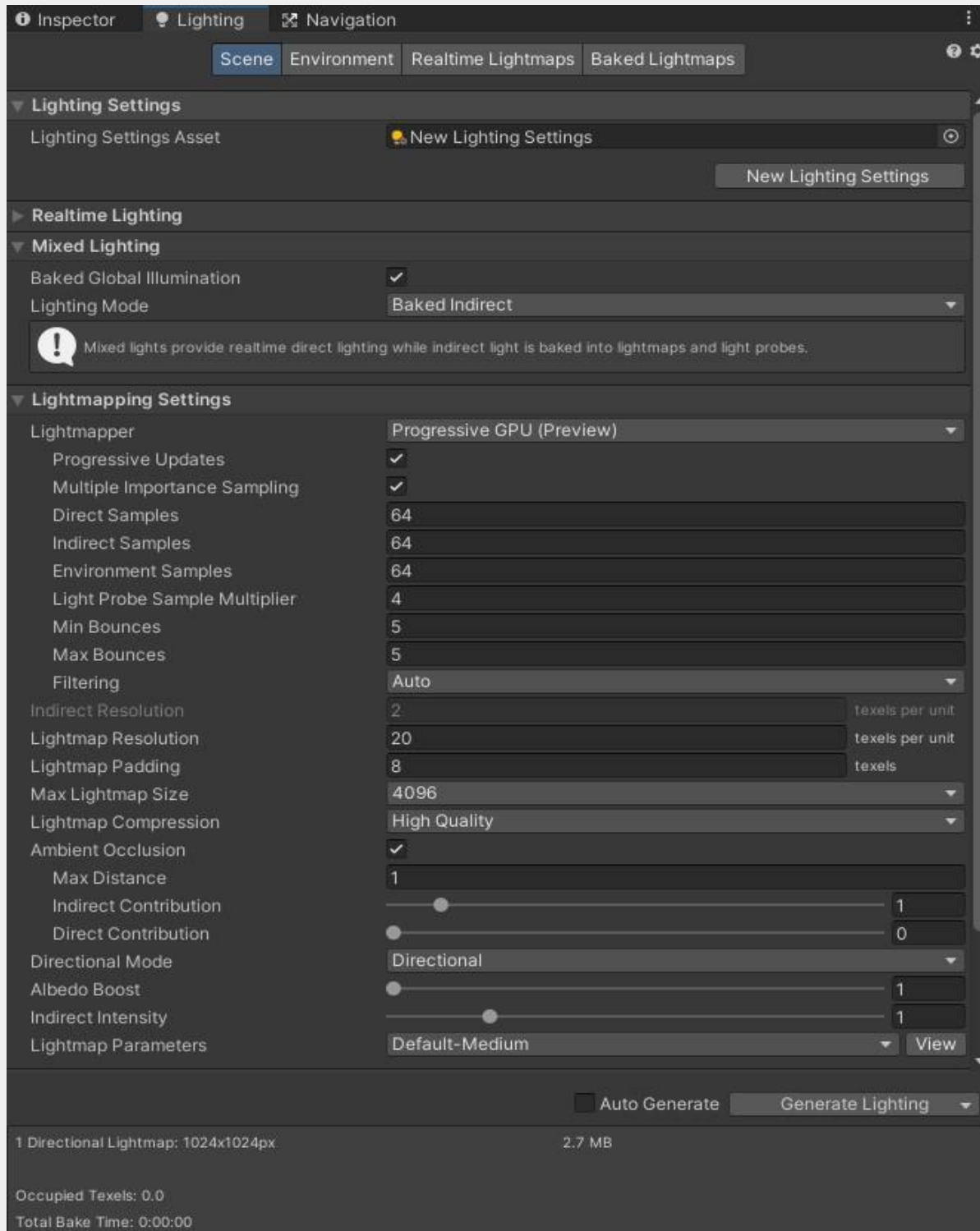
En HDRP, la **cascade des ombres (Shadow Cascade)** est gérée par un Volume Principal.

Les **cascades** sont au nombre de 2 avec une large bordure pour faire une transition fluide quand la caméra zoome et dézoome.

## 11.6. Lighting

## 11.7. Génération des lights maps

Window > Rendering > Lighting (**A REVOIR**)

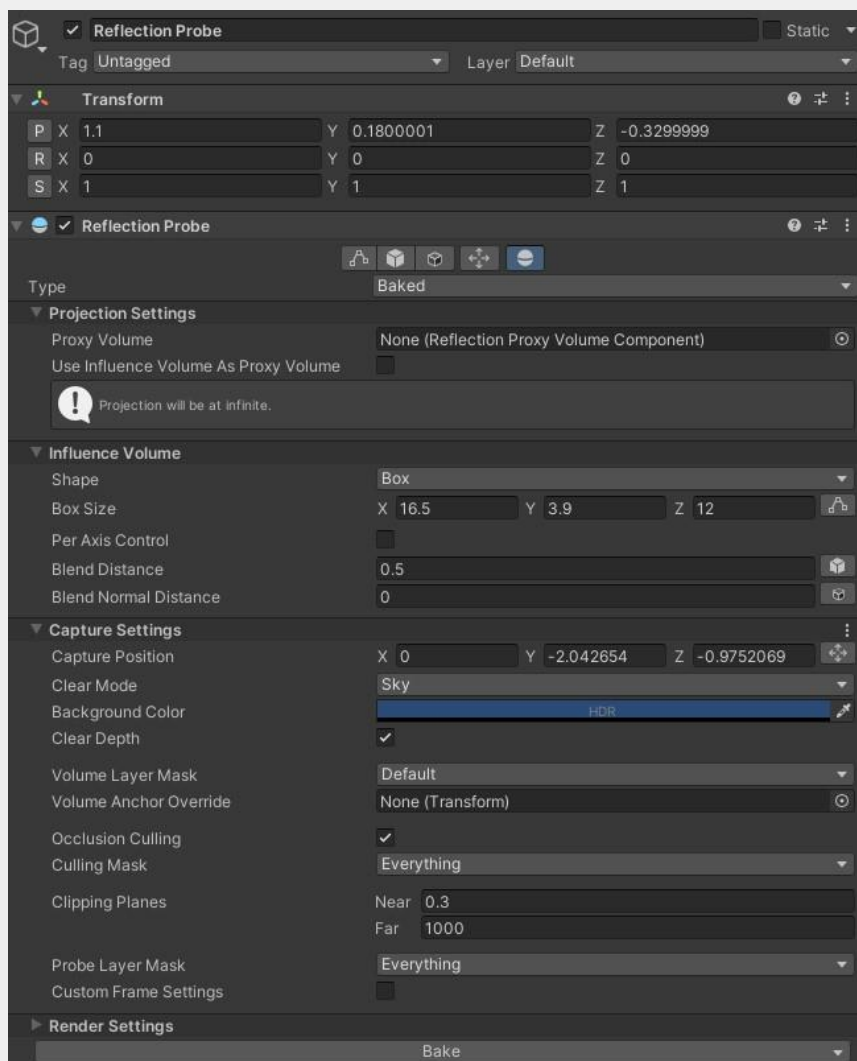


## 11.8. Reflection Probes

*Clic droit > Light > Reflection Probe*

Les **Reflections Probes** sont seulement actives sur les materials qui possèdent de la **metalness**.

A chaque ajout de Probes, régénérer les Lightning Map.



## 11.9. Light Probe Group

*Clic droit > Light > Light Probe Group*

Toujours en 4 Probes par 4 et présentes sur chaque endroit qui peut-être utilisé par un objet non Static. Les Lights Probes sont placées là où les changements de lumières sont visibles.

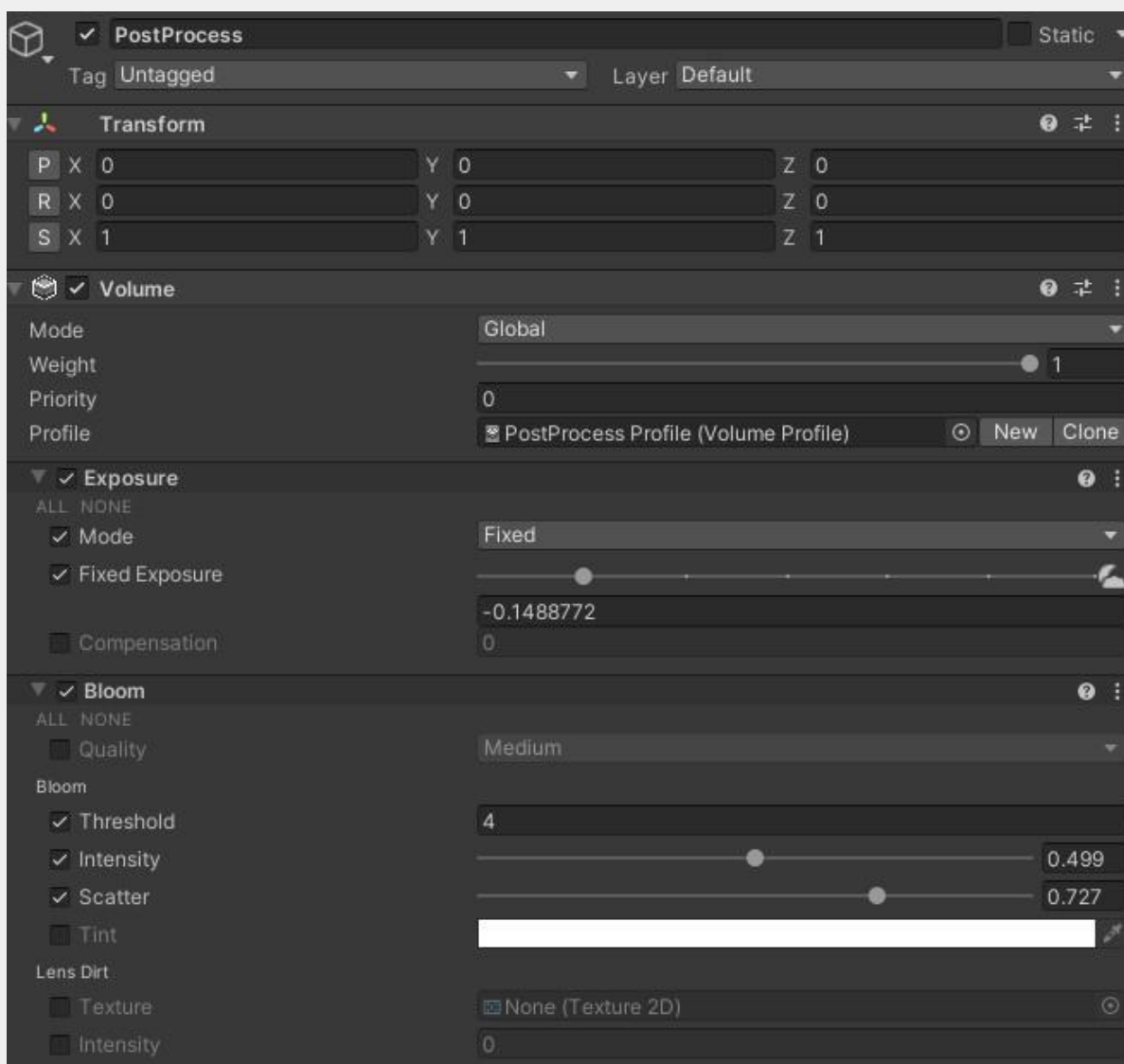
**A chaque ajout de Probes, régénérer les Lightning Map.**

## 11.10. Post Process

L'EXPOSURE et le BLOOM **A REVOIR**

*(Empty Object) > Add Component > Miscellaneous > Volume*

*Add Override > Post-processing > Exposure / Bloom*



### 11.11. Les particules VFX

Pour les VFX, il est nécessaire de mettre un dummy parent pour que les concepteurs ne travaillent pas directement sur le VFX

### 11.12. Cloth

Pour les Objets qui nécessitent un component **cloth** il faut cocher la case Read/Write Enable.

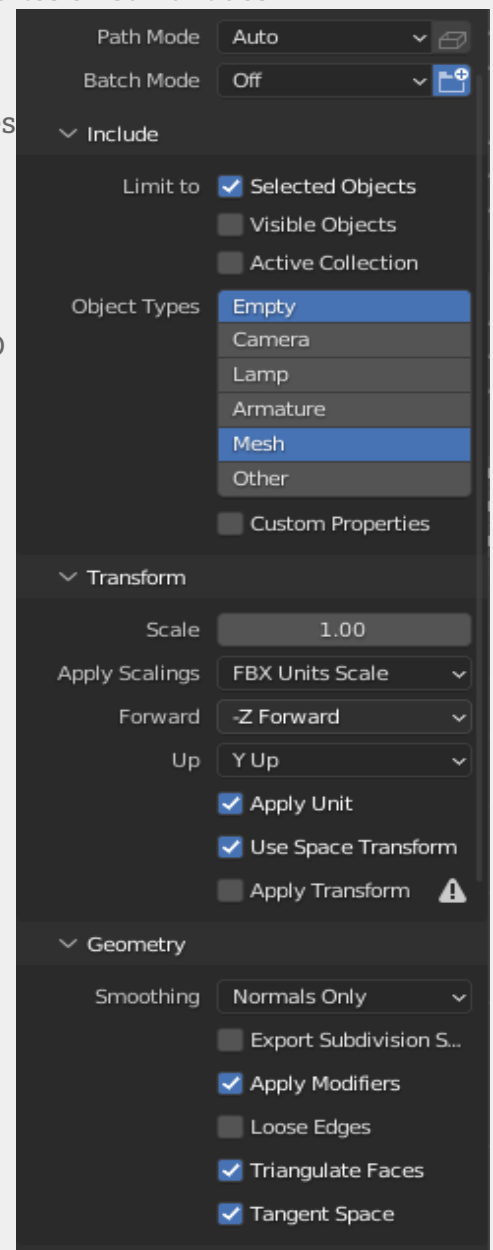


## 11.13. Paramètres d'export Modèles 3D

Il est important que tous les assets graphiques soient exportés en suivant des paramètres bien définis, à l'exception des animations qui doivent être activées et paramétrées dans l'onglet animation afin d'éviter des problèmes dans les paramètres d'import Blender.

Les props, tous les objets de décors non interactifs, doivent être importés selon ces paramètres.

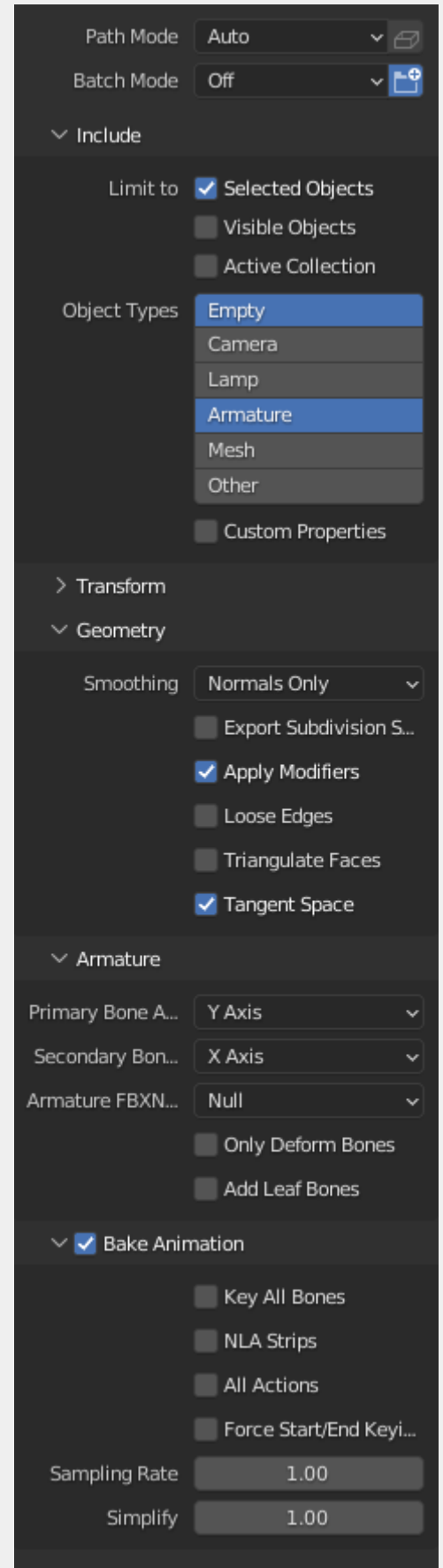
Il faut également noter que lorsqu'on exporte un Model 3D sur lequel on a appliqué un autosmooth, ainsi qu'un modifieur "Weighted Normal", il faut décocher dans le menu "Geometry" -> "Triangulate Faces" afin de ne pas override le shading



## 11.14. Paramètres d'export Rigg

Il est important que tous les Riggs pour les assets graphiques soient exportés en suivant des paramètres bien définis,

-Décocher "add leaf bones": On ne veut pas que blender ajoute un bone end à la fin de chaque extrémités

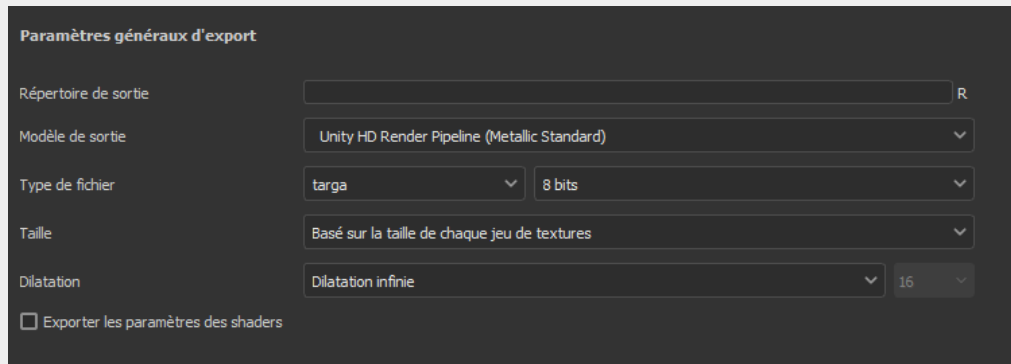


## Photoshop

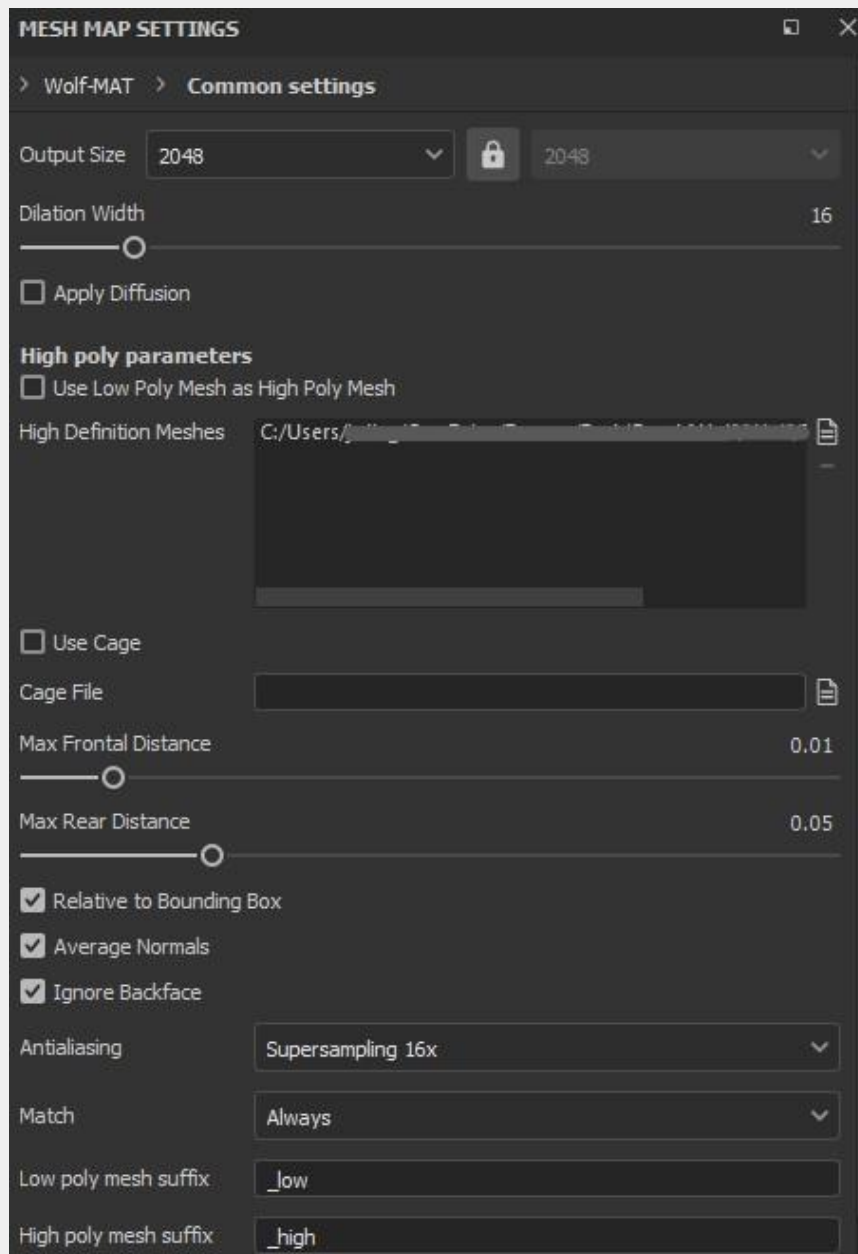
Les éléments de HUD sont réalisés sous Photoshop 2019.  
Les éléments sont exportés au format PNG.

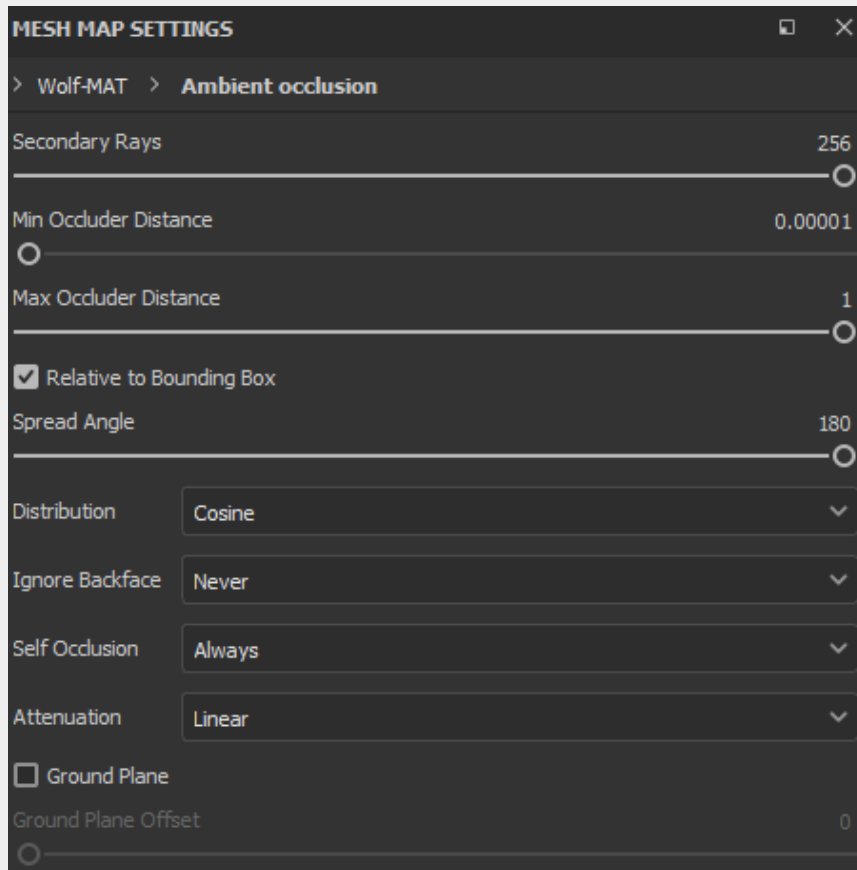
## Substance Painter 3D

Les textures sont réalisées sous Substance Painter 3D.



## Paramètres de Bake :





## 11.15. La caméra

### Hiérarchie de CAMERA-MANAGER :

La **caméra** est le GameObject **Camera**, enfant de **CAMERA-MANAGER**.

La structuration de **CAMERA-MANAGER** se fait comme suit :

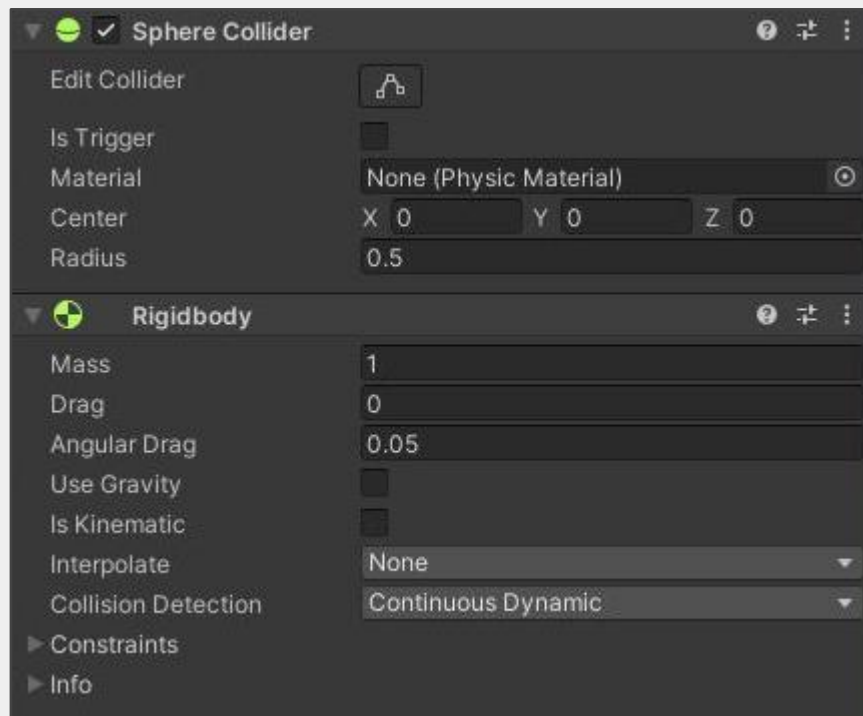
- **CAMERA-MANAGER** (Tag : Untagged ; Layer : Default) ;
  - ◆ **DummyMoveCamera** (Tag : Untagged ; Layer : Camera) ;
  - ◆ **MoveCamera** (Tag : Untagged ; Layer : Camera) ;
    - **Camera** (Tag : MainCamera ; Layer : Camera) ;
      - **BuildingTrigger** (Tag : Untagged ; Layer : Building) ;
    - **DummyCamera** (Tag : Untagged ; Layer : Default).

\* \* \*

### Explications à propos de CAMERA-MANAGER et ses enfants :

- **CAMERA-MANAGER** :  
Contient les FSMs servant à manipuler ses enfants.
- **DummyMoveCamera** :  
Dummy qui gère les déplacements de la caméra sur le plan (X ; Z).

Ce GameObject contient également un Sphere Collider ainsi qu'un Rigidbody, puisque ses déplacements se gèrent avec la physique, et servent également au GameObject de ne pas pouvoir franchir les limites du terrain.



*Configuration de Sphere Collider et Rigidbody dans DummyMoveCamera*

→ **MoveCamera :**

Contient l'enfant **Camera**, et suit **DummyMoveCamera** de façon fluide afin d'avoir dans un premier temps une meilleure expérience de jeu mais aussi pour éviter les saccades ou déplacements par à-coups.

→ **DummyCamera :**

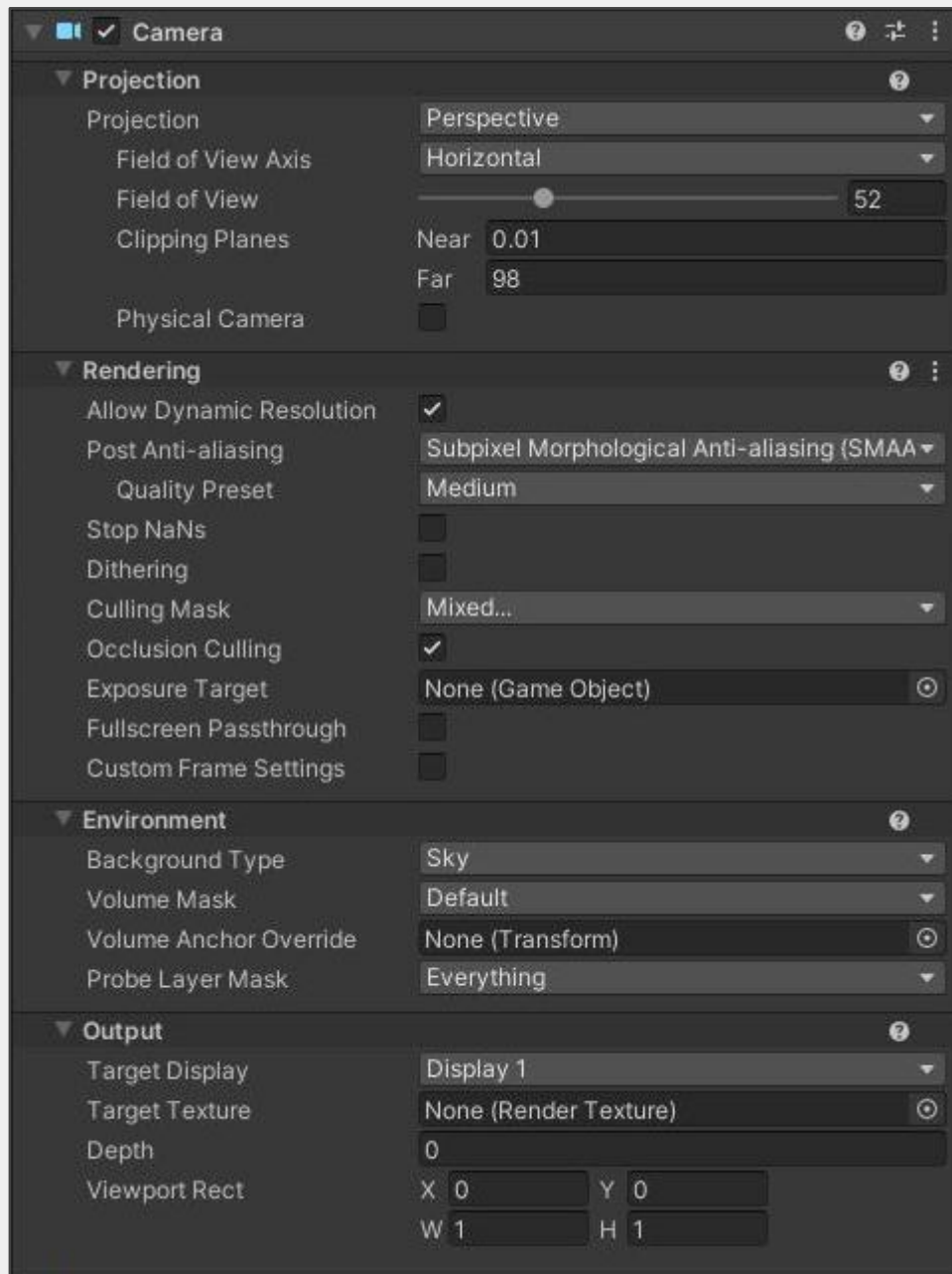
Dummy qui gère le zoom de la caméra par rapport à son axe Z (donc local).

Le zoom max. est de 20m au-dessus du sol.

Le zoom min. est de 45m au-dessus du sol.

→ **Camera :**

Contient le component Camera du jeu, et suit **DummyCamera** pour que le zoom s'effectue de façon lisse.



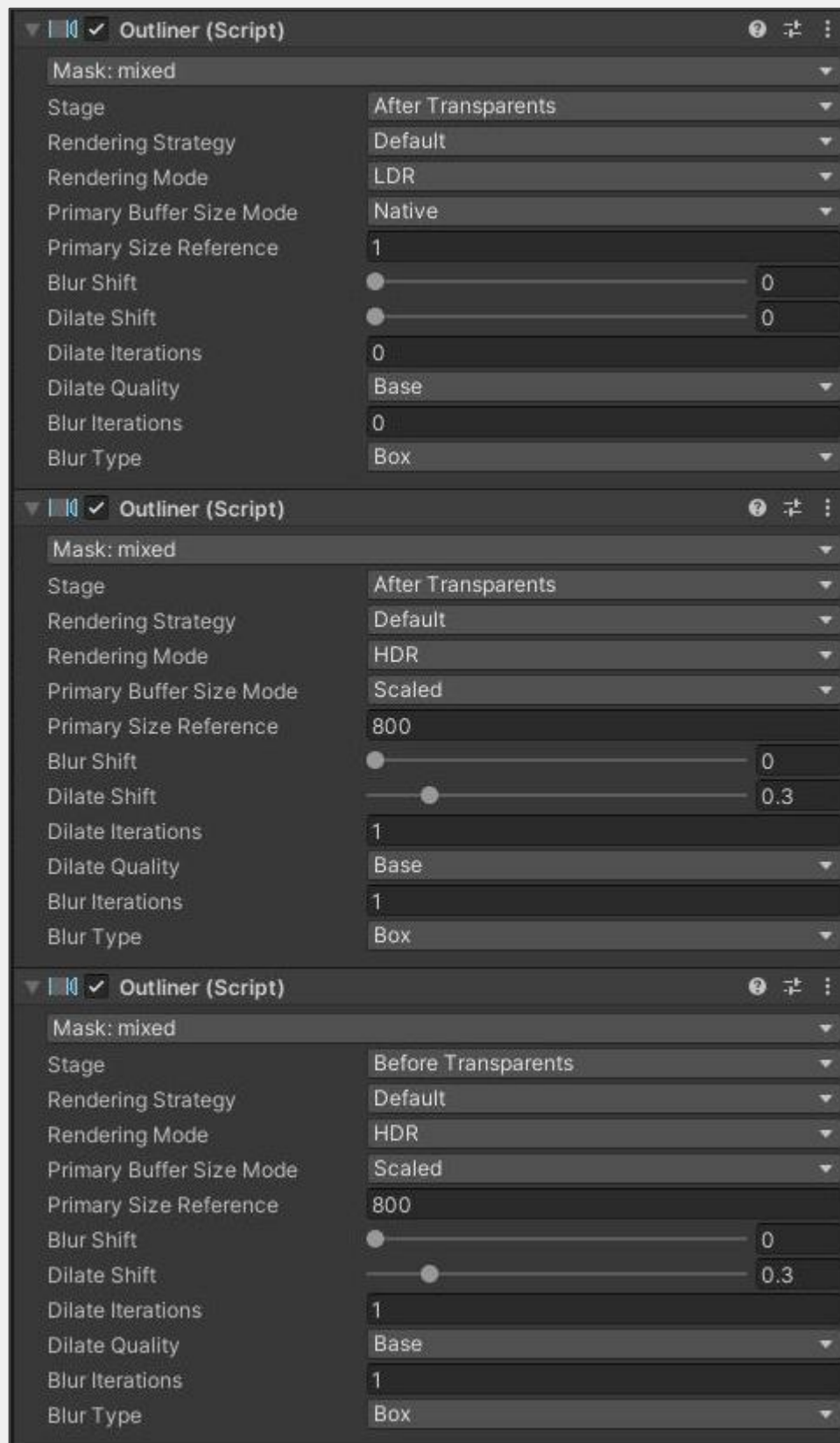
*Configuration de Camera dans Camera*

La projection de la caméra est de type perspective.

La caméra voit son paramètre de Field Of View à 52 afin d'appuyer une perspective avec un champ assez large, mais sans exagération.

**Camera** (ainsi que **DummyCamera**) est, par défaut, placée à 26.5m au-dessus du sol, et est tournée de 55° par rapport à son axe X (donc local) afin de rendre une vue top-down acceptable.

Le GameObject **Camera** contient aussi des composants Outliner, servant à montrer les contours de divers éléments de la scène.



Configuration des Outliner dans Camera

→ **BuildingTrigger :**

Il s'agit d'un Dummy contenant un Rigidbody et un Sphere Collider qui, lorsqu'il touche un building, le rend quasi transparent.

\* \* \*

Explications des FSMs que contient **CAMERA-MANAGER** :

→ **FSM-DetectIfOnScreen :**

Il détecte si le curseur Windows sort de l'écran.

Si tel est le cas, **DummyMoveCamera** se déplace sur le plan (X ; Z).

→ **FSM-FindHimself/BackToMainMenu :**

Sert à intégrer la variable globale **cameraManager** (**CAMERA-MANAGER**).

→ **FSM-FollowDummyMoveCamera :**

C'est grâce à ce FSM que **MoveCamera** suit **DummyMoveCamera** de façon fluide, et permet donc de déplacer la caméra de façon lisse.

→ **FSM-FollowZoom :**

C'est grâce à ce FSM que **Camera** suit **DummyCamera** de façon fluide, et permet donc de lisser l'effet de zoom.

→ **FSM-GetAxis :**

Envoie les valeurs des axis aux FSMs de **CAMERA-MANAGER** qui en ont besoin.

→ **FSM-Movement :**

Permet de déplacer **DummyMoveCamera** selon la méthode de déplacements de caméra utilisée.

→ **FSM-RecenterToPlayer :**

Permet de déplacer **DummyMoveCamera** jusqu'au **PJ** sélectionné.

→ **FSM-Rotate :**

Permet de faire tourner **DummyMoveCamera** et **MoveCamera** à la même vitesse, autour de l'axe Y du centre de l'écran de jeu.

→ **FSM-Zoom :**

Gère le zoom de la caméra en déplaçant **DummyCamera** par rapport à son axe Z (donc local).

\* \* \*

## Comment installer la caméra dans une scène ?

1/ Importer le nouveau Prefab **CAMERA-MANAGER** dans la scène, puis l'installer (le remplacer) dans le dossier Asset > Resources > Game > \_Manager.

2/ Ajouter et vérifier les Tags et Layers associés à **CAMERA-MANAGER** et ses enfants.  
Une liste à ce sujet se trouve au début du chapitre.

Si les layers Camera et Limit n'existent pas, alors ils doivent être créés comme suit :

- Layer 26 : Camera ;
- Layer 28 : Limit.

3/ Modifier la matrice des collisions des Layers (Edit > Project Settings > Physics > Layer Collision Matrix) comme suit :

	Limit	Camera	PathGuide	Smell	Building	SoundDetection	DeadBody	Item	RangeSkill	PathObject	EnemyView	MiniMap	Enemy	DetectionZone	Civil	Cursor	SeeThrough	Obstacle	Bush	Wall	Player	UI	Water	Ground	Ignore Raycast	TransparentFX	Default
Limit	✓																										➤
Camera		✓																									
PathGuide			✓																								
Smell				✓																							
Building					✓																						
SoundDetection						✓																					
DeadBody							✓																				
Item								✓																			
RangeSkill									✓																		
PathObject										✓																	
EnemyView											✓																
MiniMap												✓															
Enemy													✓														
DetectionZone														✓													
Civil															✓												
Cursor																✓											
SeeThrough																	✓										
Obstacle																		✓									
Bush																			✓								
Wall																				✓							
Player																					✓						
UI																						✓					
Water																							✓				
Ground																								✓			
Ignore Raycast																									✓		
TransparentFX																										✓	
Default																											➤

*Layer Collision Matrix*

Attention : des Layers peuvent encore s'ajouter ou être modifiés, c'est pourquoi cette capture peut devenir obsolète rapidement.

De ce fait, le plus important concernant la **caméra** sont les Layers **Default**, **Camera** et **Limit**.

Les cases concernant leurs interactions physiques ne changeront pas.

4/ Ajouter / Remplacer le nouveau FSM FSM-GameCursor/Position dans **SELECTOR-MANAGER**.

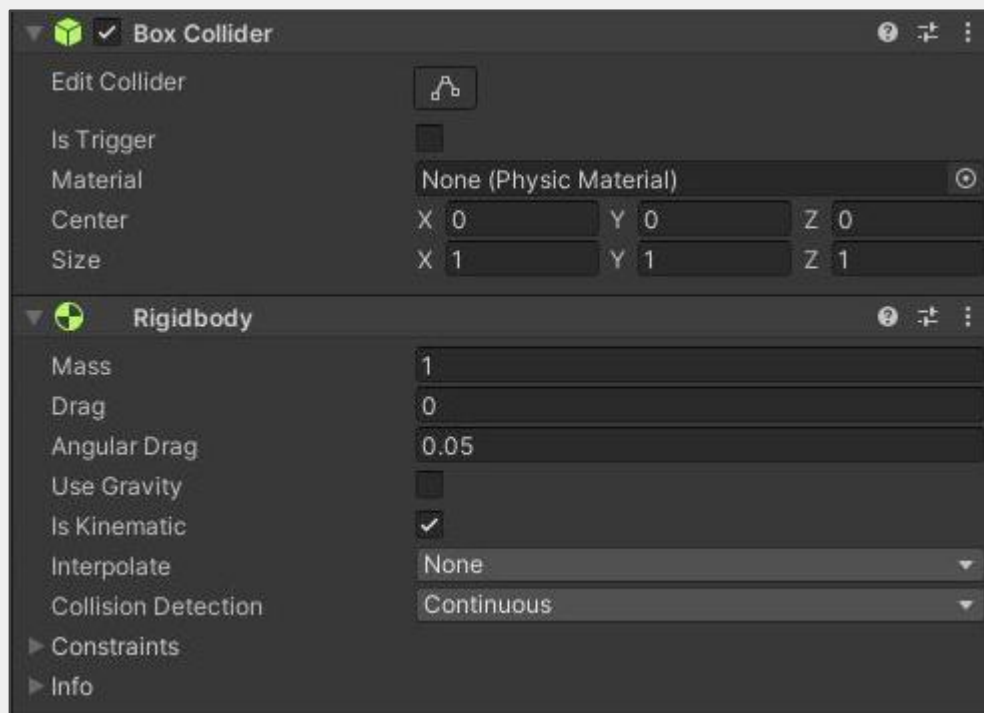
\* \* \*

### Comment installer des limites de terrain dans une scène ?

Un GameObject contenant des limitations de terrain doit être installé dans la scène. Par exemple, un GameObject (un Dummy) ayant pour nom **Limits** doit contenir des enfants qui sont les vrais limites du terrain (ex : **Limit-01** ; **Limit-02**...).

Ces enfants doivent avoir un Tag Limit et un Layer Limit.

Il doivent aussi contenir les composants Collider (Cube Collider, Sphere Collider, Mesh Collider...) et Rigidbody.



*Configuration de Sphere Collider et Rigidbody dans Limit-XX*

## 12. Scene Management

### 12.1. Les différents layers utilisés

- Layer 0 : Default
- Layer 1 : Transparent FX
- Layer 2 : Ignore Raycast (pour les objets qui laisseront passer le clic, comme certain murs ou props)
- Layer 3 : Ground (Sol/Terrain/décors)
- Layer 4 : Water
- Layer 5 : UI
- Layer 6 : Player
- Layer 7 : Wall
- Layer 8 : Bush
- Layer 9 : Obstacle
- Layer 10 : SeeThrough
- Layer 11 : Cursor
- Layer 12 : Civil
- Layer 13 : DetectionZone
- Layer 14 : Enemy
- Layer 15 : MiniMap
- Layer 16 : EnemyView
- Layer 17 : PathObject
- Layer 18 : RangeSkill
- Layer 19 : Item
- Layer 20 : DeadBody
- Layer 21 : SoundDetection
- Layer 22 : Building
- Layer 23 : Smell
- Layer 24 : /
- Layer 25 : Projectile
- Layer 26 : Camera
- Layer 27 : Objectives
- Layer 28 : Limit

### 12.2. Les différents tags utilisés

- Untagged (objets non interactifs)
- Tag 0 : Interactive
- Tag 1 : Sound
- Tag 2 : Enemy
- Tag 3 : View

- Tag 4 : Smell
- Tag 5 : CameraMove
- Tag 6 : WallCamera
- Tag 7 : PlayerInteract
- Tag 8 : Tracked
- Tag 9 : Highlight
- Tag 10 : PlayerAttack
- Tag 11 : EnemyDeadBody
- Tag 12 : Civil
- Tag 13 : Item
- Tag 14 : SmellScale
- Tag 15 : PathObject
- Tag 16 : UI
- Tag 17 : RangeSkill
- Tag 18 : loudNoise
- Tag 19 : Bush
- Tag 20 : Move
- Tag 21 : Blood
- Tag 22 : Guide
- Tag 23 : FootStep
- Tag 24 : Ground
- Tag 25 : ObjectInterest
- Tag 26 : HUD
- Tag 27 : Balise
- Tag 28 : TrapTarget

## 12.3. Collision des layers

	Default	TransparentFX	Ignore Raycast	Ground	Water	UI	Player	Wall	Building	Obstacle	SeeThrough	CursorGround	Civil	DetectionZone	Enemy	MiniMap	EnemyView	PathObject	RangeSkill	Item	DeadBody	SoundDetection
Default	>																					
TransparentFX																						
Ignore Raycast																						
Ground																						
Water																						
UI																						
Player																						
Wall																						
Building																						
Obstacle																						
SeeThrough																						
CursorGround																						
Civil																						
DetectionZone																						
Enemy																						
MiniMap																						
EnemyView																						
PathObject																						
RangeSkill																						
Item																						
DeadBody																						
SoundDetection																						

## 13. Coding Standard

### 13.1. Nomenclature

Les FSM commencent tous par "FSM- " ainsi qu'un état INIT.

Les Template de FSM commencent tous par "FSM-" et finissent tous par "-Template

Les Variables s'écrivent principalement en minuscules, sans majuscules au début, sans caractères spéciaux. Les majuscules doivent être utilisées comme séparateur.

Les Global Event se profilent de cette manière "PLAYER/TAKEDMG".

Les Event sont écrits en majuscules et avec espace pour la visibilité.

Tous les FSM et leurs états doivent être commentés pour faciliter leur compréhension.

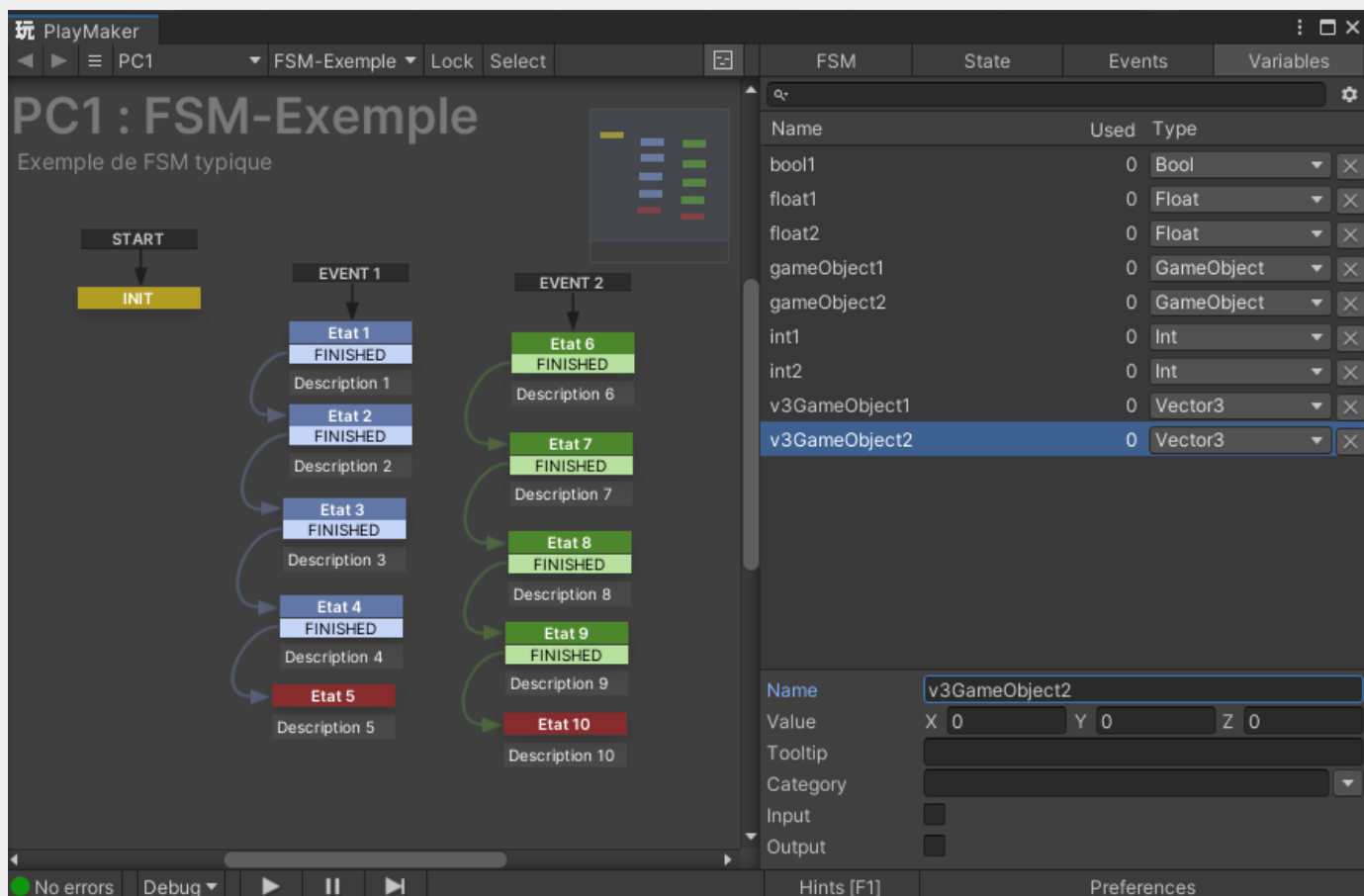
⚠ Faire attention à placer les FSM sur le ROOT des objets nécessitant un script.

### 13.2. Code couleurs

Les états INIT sont *Jaunes*

Les boucles sont *Oranges*

Les fins de boucles sont *Rouges*

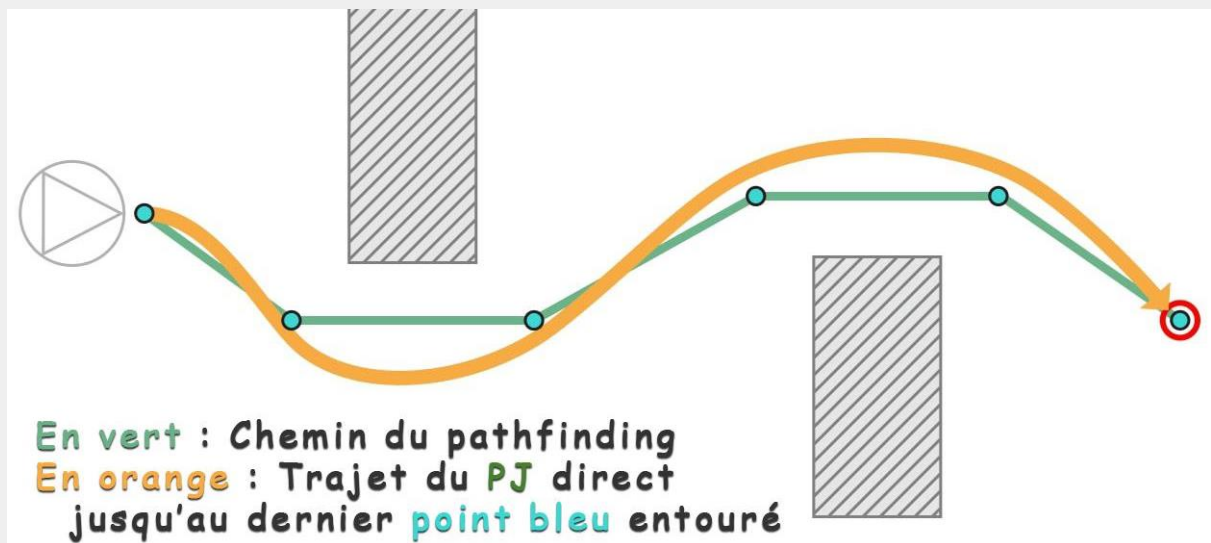


### 13.3. Versionning

Les versions des POC seront répertoriées ainsi : NumeroDuPOC + VerMajeur + VerMineur + Patch.

- Majeur : Si ajout de grosses fonctionnalités, de grosses modifications dans le code ou à l'ajout d'assets graphiques importants.
- Mineur : Modification de fonctionnalités déjà présentes ou à l'ajout d'asset mineur.
- Patch : Correction de bug ou modification mineur du script

## 14. Pathfinding

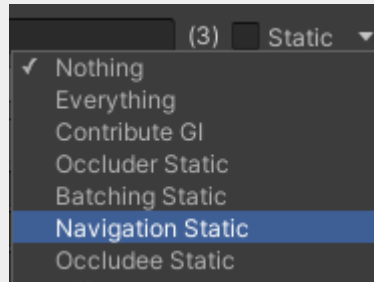


Si l'objectif à atteindre est assez éloigné, le personnage peut utiliser des **éléments interactifs** afin de le rejoindre au plus court, comme des échelles, des portes....

Par contre, si c'est un **groupe de PJs** qui se dirige jusqu'à leur nouvel emplacement, c'est le **PJ** qui est au plus près de l'**élément interactif** qui interagit avec en premier, puis le second, etc....

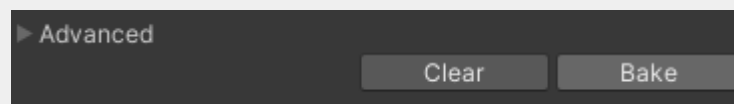
Si le joueur clique ailleurs pendant la course d'un **PJ**, ce dernier se redirigera jusqu'au nouveau point.

Avant tout il faut que la zone cible soit "Navigation Statique".



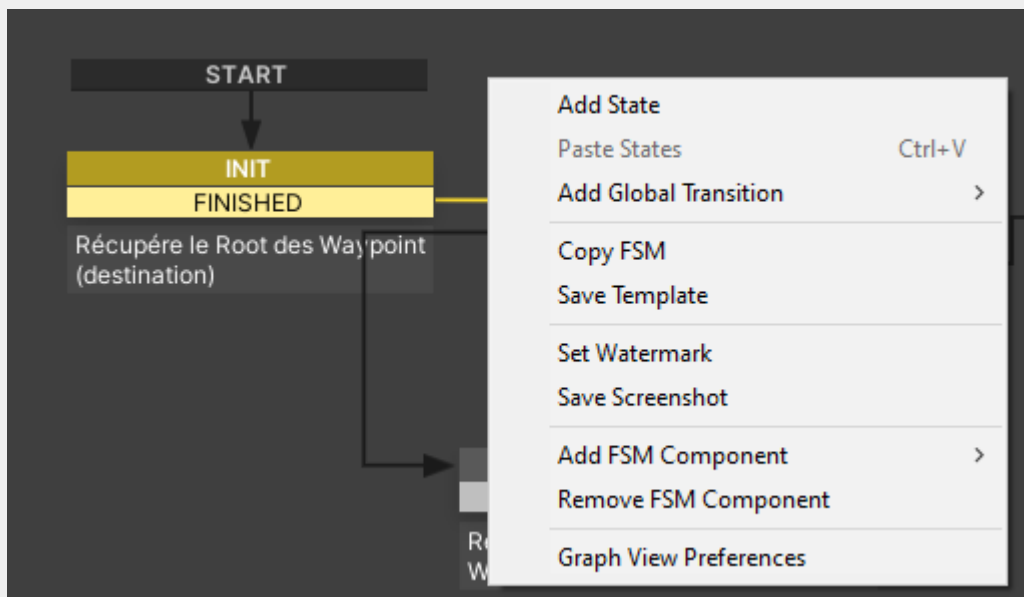
Pour que les PJs puissent se déplacer dans la zone il suffit de **Bake** la zone depuis la fenêtre Navigation d'Unity présente dans : *Window* ⇒ *AI* ⇒ *Navigation*.

Dirigez-vous ensuite sur l'onglet Bake, puis cliquer sur le bouton "Bake"



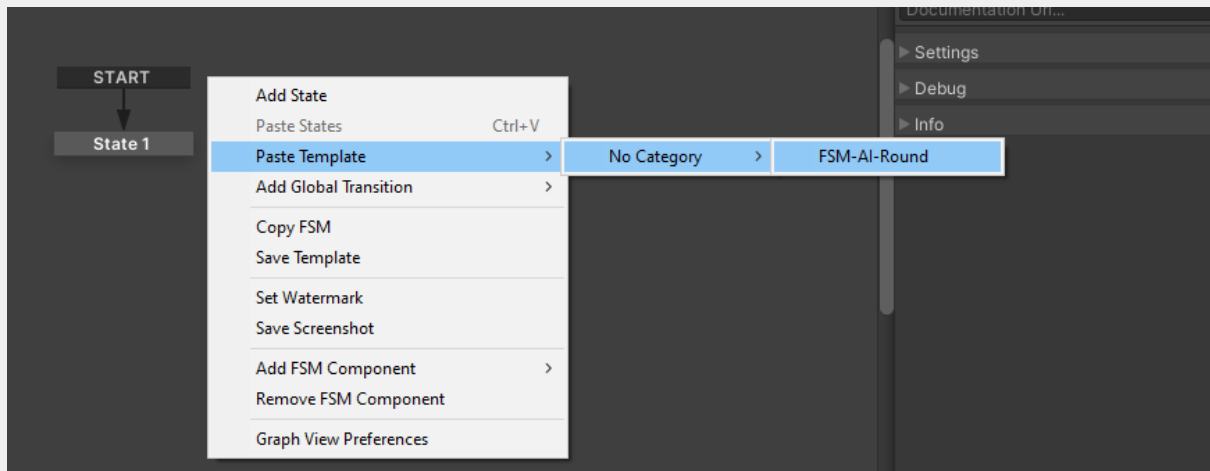
## 15. Créer/Ajouter un Template FSM :

Lorsqu'un de vos scripts doit être utilisé sur plusieurs objets il est nécessaire d'en faire un "Template" pour faciliter la tâche. Pour créer un "Template", il suffit d'ajouter le composant PlayMakerFSM. Éditer le "FSM" créer, faire clic droit + Save Template.



*\*Sauvegarder vos Template dans Le répertoire Assets ⇒ PlayMaker ⇒ Template.*

Pour ajouter un Template à l'un de vos FSM, il suffit d'ajouter le composant PlayMakerFSM. Dans l'édition, faire clic droit + Paste Template + Le Template de votre choix.



## 16. Les Personnages Jouables (PJ)

### 16.1. Setup

Les prefabs de Models de Personnages sont équipés d'un FSM-Statistiques qui permet de set toutes les statistiques propres à chaque personnage, ainsi que la taille de sa capsule collider

Les prefab de Models se trouvent dans le répertoire  
**Asset/\_RESOURCES/Game/Character/PC**

Il y a 4 catégories de variables :

- Les variables de **Movement**
- Les variables de **HUD**
- Les variables du **Cursor**
- Les variables **Other**

Les FSM concernés par ce script sont :

- Pour les **Mouvements** :
  - FSM-Movement [playerRoot]
  - FSM-MovementSound [playerRoot]
- Pour le **HUD** :
  - FSM-Selected [playerRoot]
  - FSM-Dialog [playerRoot]
- Pour le **Cursor** :
- Pour les **Other** :
  - FSM-Outile [playerRoot]
  - FSM-Hp-Manager [playerRoot]

### 16.1.1. Ajouter des Statistiques

Pour ajouter une statistiques dans ce FSM il faut :

1. ajouter la variable désirée en input.....
2. catégoriser de manière appropriée.
3. ajouter l'action playmaker dans la state correspondante à la catégorie.

(Ex : je souhaite ajouter le nom d'une icône,

1. j'ajoute une variable string en input avec le nom désiré
2. je la range dans la catégorie HUD
3. j'ajoute l'action "set FSM String" à la state HUD que j'envoie au "FSM-Selected" sur le Player Root)

## 16.2. Déplacements

Les PJ's se déplacent grâce au système de "PathFinding" (Cf.14.Pathfinding) directement présent dans le moteur Unity.

Tous les "FSM" liés au déplacement sont sur le "PlayerRoot" (trouvable dans le répertoire : (*"Assets⇒Resources⇒Game⇒Character⇒PC"*)).

Les déplacements des PJ's se font en 3 grandes étapes :

- Étape 1 : Indiquer la destination du PJ, avec le **"FSM-PathFind"**.
- Étape 2 : Calculer le chemin du PJ, avec le **"FSM-CalculatePath"**.
- Étape 3 : Set les animations et set les "statistiques de déplacement du joueur, avec le **"FSM-Movement"**.

### 16.2.1. Indiquer la destination :

La destination du PC se fait dans le **"FSM-PathFind"**.

Elle est définie par rapport à la position du "GameCursor" présent dans le "SELECTOR-MANAGER" (trouvable dans le répertoire : (*"Assets⇒Resources⇒Game⇒\_Manager"*)).

Depuis le **"FSM-PathFind"** il est possible de set :

- La destination du PJ lors d'un clique gauche dans le niveau en envoyant l'événement **"PATHFIND/SETPOSITION"**
- La destination du PJ lors de l'utilisation d'un skill en envoyant l'événement **"PATHFIND/SETSKILLPOS"** (Ajouter un délai de 0.5), (La variable "gameObject" de la cible est la suivante **"pnj/Target"**).
- Arrêter le mouvement du PJ en envoyant l'événement **"PATHFIND/STOPMOVE"**.

### 16.2.2. Calculer le chemin :

Le chemin se calcule dans le **"FSM-CalculatePath"**, pour cela il stock toutes les destinations du PJ dans un array **"arrayPathCorners"**.

Ce FSM pose une sphere à chacun des virages présents sur le chemin du PJ, ce qui permet de connaître le prochain angle de rotation du PJ.

Depuis le FSM **"FSM-CalculatePath"** il est possible de set :

- Changer l'array de destination **"arrayPathCorners"**.
- Calculer un nouveau chemin, en envoyant l'événement **"PATHFIND/CALCULATEPATH"**.

### 16.2.3. Gestion des Mouvements

Les animations sont "set" depuis le **"FSM-Movement"** qui selon le mouvement choisit définit les booléens correspondant dans l'"Animator controller" liés qui se trouvent dans le répertoire : (*"Assets⇒\_ANIMATION⇒Character⇒Animators"*).

Depuis le FSM les variables de l'Animator modifiées sont les suivantes :

- Float :
  - yPosition
  - floatLR
  - floatFB
- Bool :
  - startMovingBool
  - endMovingBool

## 16.3. Path-Object

### 16.3.1. Les échelles

Les déplacements path-object sont gérés via le composant **"OffMeshLink"**.

Les scripts qui touchent au path-Object de l'échelle sont les suivants : **"FSM-Movement"** **"FSM-CalculatePath"** **"FSM-PathObjectChecker"** qui sont sur le **"PlayerRoot"** qui se trouve dans le répertoire : (*"Assets⇒Resources⇒Game⇒Character⇒PC"*).

Les Ladders sont disponibles sous plusieurs hauteurs : 3, 4.5 et 6 mètres elles se trouvent dans le répertoire : (*"Assets⇒Resources⇒Game⇒Props⇒PathObjects"*).

## 16.4. Sélection

La sélection des PJ's se fait dans le **"FSM-Selector"** présent sur le **"SELECTOR-MANAGER"** (trouvable dans le répertoire : (*"Assets⇒Resources⇒Game⇒\_Manager"*)).

La désélection des PJ's se fait dans le **"FSM-Unselector"** présent sur le **"SELECTOR-MANAGER"**.

Lorsqu'un / ou plusieurs PJ's sont sélectionnés ils sont stockés dans un array **"playerSelectedArray"** dont sa "length" est stockée dans un int **"playerInArray"** trouvable sur le **"FSM-GetPlayerSelected"** dans le **"SELECTOR-MANAGER"**.

### 16.4.1. Sélection manuelle

La sélection manuelle se fait depuis les "FSM-MousePick-Detection", "FSM-Détection" et le "FSM-Selector".

Elle se définit en 3 étapes clés :

- Étape 1 : Récupérer l'objet touché depuis le "FSM-MousePick-Detection".
- Étape 2 : Si l'objet touché est tagué "Player" Set up la sélection dans le "FSM-Détection"
- Étape 3 : Une fois que le joueur a cliqué. Sélectionne le PJ choisi dans le "FSM-Selector"

#### 16.4.2. Sélection de groupé

La sélection groupée se fait depuis le "FSM-Selector" ainsi que le "FSM-Selected" présent sur le "PlayerRoot" trouvable dans le répertoire :

(*"Assets⇒Resources⇒Game⇒Character⇒PC"*).

Lorsque le joueur clique droit, il crée un carré de sélection. Les PJ vont ensuite vérifier s'il ils sont à l'intérieur de celui-ci ou non depuis leurs "FSM-Selected".

### 16.5. Ajouter une compétence

Chaque character possède en son *FSM-Statistique* des variables aux noms de skillX ainsi que skillXString représentant ses compétences. Lorsqu'ils sont remplis correctement, ils s'ajouteront aux personnages jouables dès leur chargement, car ils sont envoyés au Player-Root dans le FSM-Statistique.

- La variable skillX est à remplir avec le template du FSM de la compétence.
- La variable skillXString est à remplir avec le nom du template du FSM de la compétence.

De cette façon, chaque compétence est travaillée dans un FSM, puis mise dans un template.

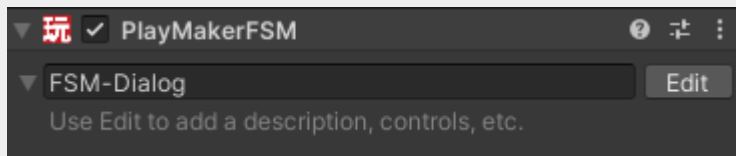
### 16.6. Dialogues

#### 16.6.1. Game object et composants

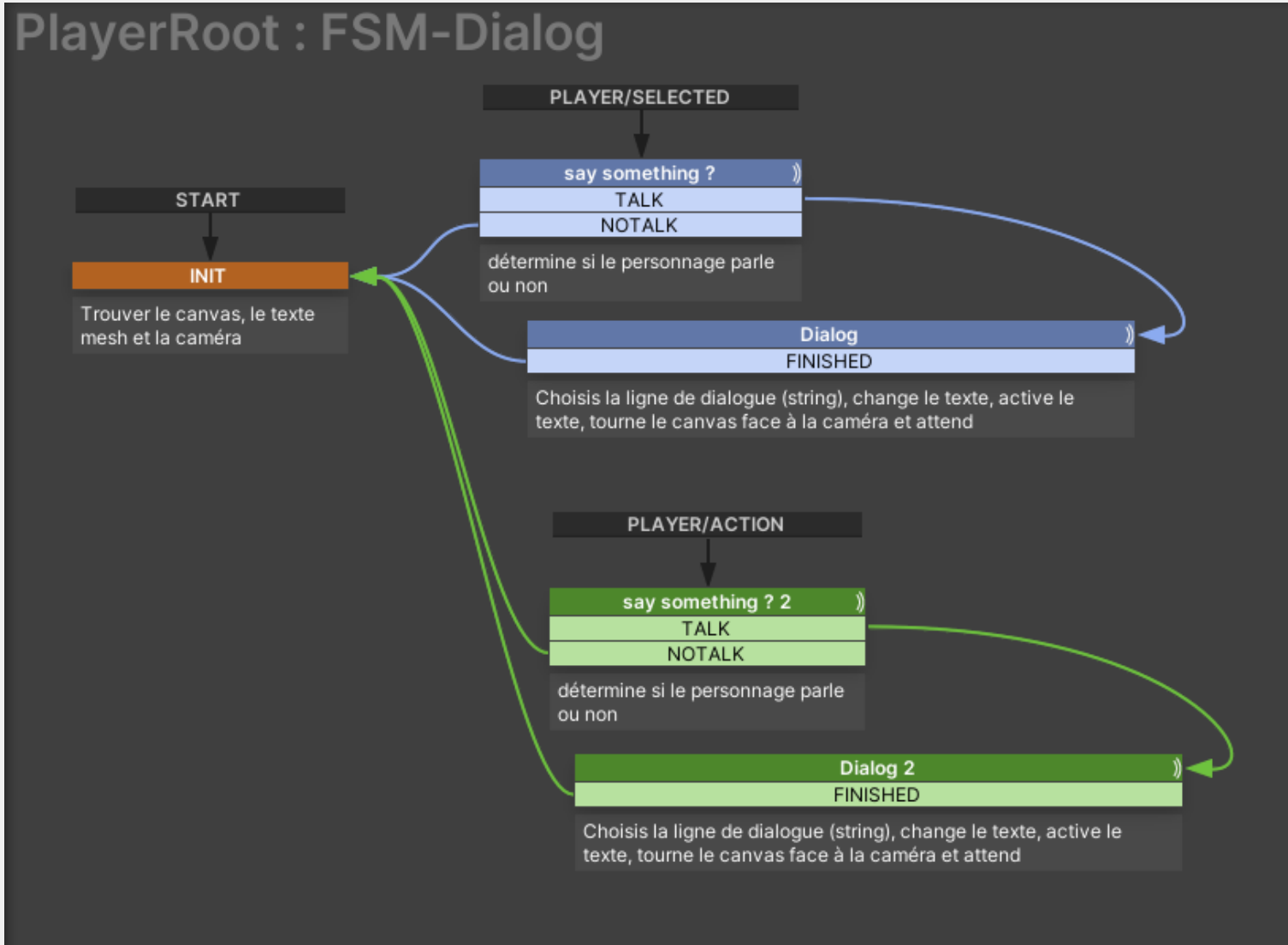
Les Game Object nécessaire pour l'affichage des dialogues sont le prefab Cont-Dialog et l'empty object Dialog-Anchor qui doit se trouver enfant du Dummy Gameplay à une hauteur de 2.

#### 16.6.2. Code

Le FSM doit être ajouté au *Player Root*



Structure :



Ajouter des *Send Event* dans les FSM appropriés (exemple : dans le *FSM-Selection* ajouter un *Send Event : PLAYER/SELECTED* dans les états où le personnage se fait sélectionner)

Pour personnaliser les lignes de dialogues en fonction du personnage, Ajouter des *Array de Strings* dans le *FSM-Statistiques*.

Les Arrays sont ensuite envoyées au *Player Root* au *FSM-Dialog*.

## 16.7. Interactions

## Port de corps

Un **ennemi** est toujours nommé comme suit : **Enemy-EnemyName-ROOT** (ex : *Enemy-Wolf-ROOT*).

Et chacun d'entre eux voyant leurs points de vie chuter à 0 meurt.

Lorsqu'un **ennemi** est tué, il est instantanément remplacé par son cadavre (dead body), et ce à la même position.

Chaque cadavre est nommé comme suit : **EnemyDead-Root-EnemyName** (ex : *EnemyDead-Root-Wolf*).

Avant de s'écrouler au sol, le dead body joue alors une animation de mort : **Enemy-AnimatorController**.

Ce controller (donc l'Animator en question) se trouve dans le modèle de l'ennemi.

Ici, la hiérarchie d'un cadavre (donc d'un **prefab EnemyDead-Root-EnemyName**) : **EnemyDead-Root-EnemyName** (contient les FSMs) ;

- **DummyGameplay** ;
  - ◆ **PlayerPosToGo** ;
  - ◆ **SetUpAnim** ;
  - ◆ **Collider-RayCast** ;
- **DummyGraph** ;
  - ◆ **Enemy-EnemyName Variant** (modèle contenant le controller Enemy-AnimatorController et l'avatar Enemy-EnemyNameAvatar).

◆ ◆ ◆

L'Animator **Enemy-AnimatorController** comporte deux states pour le fonctionnement de l'**animation de mort** et l'**animation de portage par un PJ**.

Pour l'animation de mort, le Blend Tree **DeathTree**.

Et pour l'animation de portage par un PJ : **Carried**.

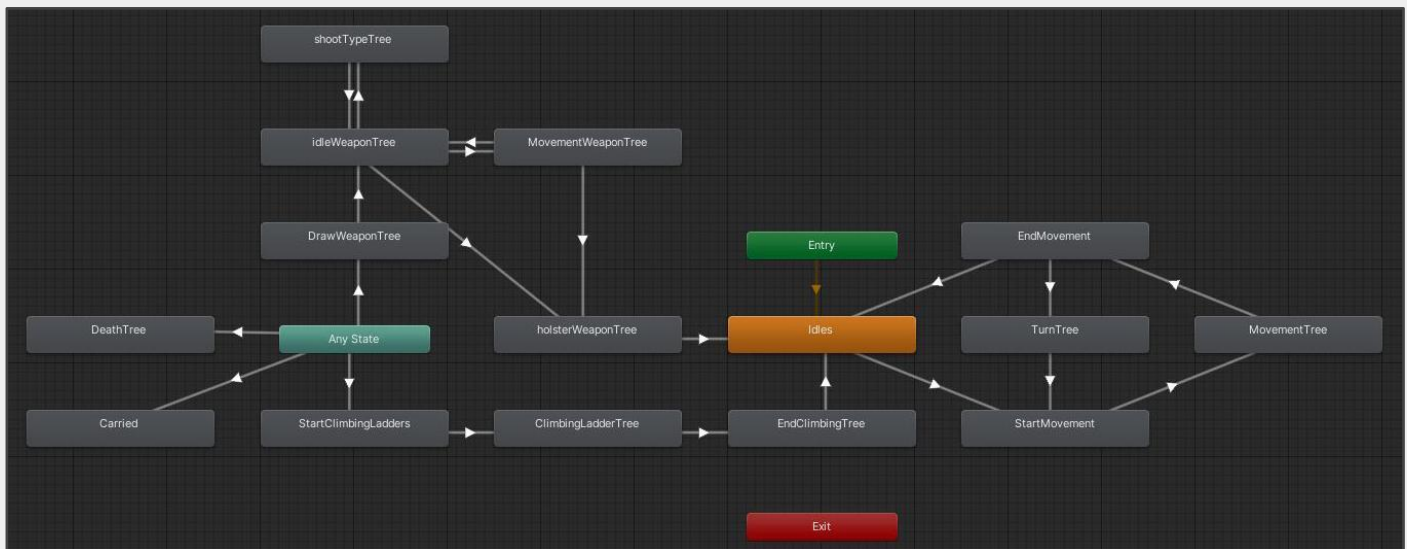
Pour passer du state **Any State** au state **DeathTree** ou **Carried**, une condition de type **trigger** doit être déclenchée.

Any State à DeathTree : le trigger **deathTrigger**.

Any State à Carried : le trigger **carryBody**.

Le trigger **deathTrigger** est déclenché par le **FSM-DeadAnim** de chaque **prefab EnemyDead-Root-EnemyName** : state **DeathAnimation**.

Le trigger **carryBody** est déclenché par le **FSM-CarryBody** du **prefab PlayerRoot** : state **SetAnim**.



Controller *Enemy-AnimatorController*

Les **FSMs** utilisés pour le **port de corps** sont :

- Pour le **prefab PlayerRoot** :
  - ◆ FSM-CarryBody ;
  - ◆ FSM-GetAxisPlayer ;
  - ◆ FSM-Movement ;
  - ◆ FSM-PathFind.
- Pour les **prefabs EnemyDead-Root-EnemyName** :
  - ◆ FSM-CheckPlayerLocatePoint ;
  - ◆ FSM-DeadAnim ;
  - ◆ FSM-Interactive ;
  - ◆ FSM-StopCarryBody.

## Explications des FSMs :

- **FSM-Interactive** : Vérifie si le joueur appuie sur les touches permettant le port de corps ;
- **FSM-CheckPlayerLocatePoint** : Vérifie si le / les **PJ(s)** se dirige(nt) au point point de clic ;
- **FSM-CarryBody** : Sert à ramasser un corps ou à le reposer au sol ;
- **FSM-PathFind** : Sert à diriger ou à arrêter le **PJ** selon le système du pathfinding d'Unity ;
- **FSM-Movement** : Sert à spécifier le type de déplacements du **PJ** (marcher, courir, s'accroupir, prendre une échelle...) ;
- **FSM-DeadAnim** : Permet à l'ennemi de jouer l'animation de mort (Enemy-AnimatorController). Lui permet aussi de jouer l'animation de "porté" (EnemyIsCarried) et "posé au sol" (EnemyDrop) ;
- **FSM-StopCarryBody** : Sert au **PJ** qui porte le corps de le poser au sol puis de se remettre debout (mêmes inputs utilisés que pour l'action de porter un corps) ;

- **FSM-GetAxisPlayer** : Sert à récupérer les inputs utilisés par le joueur et de faire la liaison avec les FSMs FSM-Movement et FSM-PathFind.

Il existe 2 méthodes pour un **PJ** de **poser un corps** :

- Soit en **se baissant** (Crouch) en utilisant l'input concerné.  
Dans ce cas, une fois le corps posé au sol, le **PJ** reste accroupi ;
- Soit en utilisant les mêmes inputs que pour l'**action de porter un corps**, ce qui le fait remettre debout.

## 16.8. Compétences

Les compétences sont réparties en 4 catégories : les compétences de **Corps à Corps**, les compétences de **tir**, les compétences de **classe** et les compétences **signature**. Chaque personnage possède une capacité de chaque catégorie.

Tous les FSM de compétences sont rangés en tant que Template dans le dossier Assets.

Les Models des personnages contiennent tous un **FSM-Skills-MANAGER** qui va attribuer les templates au Player-Root associé. ainsi que les stats spécifiques à chaque personnage ou à leur classe.

La plupart des FSM de compétence contiennent des statistiques similaires :

- **skill/CD** : Cooldown de la compétence (float)
- **skill/Damage** : dégâts infligés à la cible (int)
- **skill/Range** : portée de la compétence (float)
- **skill/RangeNoise** : portée du son produit par la compétence (float)
- **skill/Blood** : détermine si la cible de l'attaque laisse une trace de sang ou non (Bool)

Quand une compétence est activée, cela change le scale du de l'objet **Dummy-Range** en fonction de la statistique **Skill/Range** dans le **Dummy-Gameplay** ce qui permet d'afficher la portée au joueur grâce à un Decal.

Toutes les compétences sont associées à un FSM-CooldownSkillX (X étant le numéro de la compétence déterminé par le type : **CàC** = 1; **Tir** = 2; **Classe** = 3; **Signature** = 4)

Ces FSM permettent de déterminer à tout moment le temps restant au cooldown du skill associé.

### 16.8.1. Compétence de Corps à Corps (CàC)

La compétence de CàC est commune à tous les personnages. Seul change la durée de l'attaque en fonction de la classe du personnage.

Structure :

1. Le joueur active la compétence, la portée s'affiche autour du personnage sélectionné.
2. Quand le joueur passe la souris sur une cible (civile ou ennemi), un cercle apparaît autour de la cible pour indiquer la zone de son qui sera produite durant l'attaque.
3. Quand le joueur clique gauche sur la cible, le personnage se dirige vers la cible et déclenche l'attaque au contact du Dummy-Range avec la cible.
4. Au contact, les personnages sont ensuite alignés en fonction de l'angle par lequel arrive le personnage jouable pour rendre l'animation cohérente. (+ de 90° par rapport à l'avant de l'ennemi = backstab, l'ennemi est mis dos au personnage jouable)
5. L'animation et le cooldown sont déclenchés, les personnages sont immobilisés durant leur animation, la zone de son est produite.

Statistiques :

- **skill/CD** : 6s
- **skill/Damage** : 4 (kill garanti)
- **skill/Range** : 3
- **skill/RangeNoise** : 5
- **skill/Blood** : True

### 16.8.2. Compétences de Tir

Les compétences de tir peuvent varier en fonction du personnage jouable, mais la plupart utilisant une arme de poing, la capacité sera basé sur le même template.

Les attaques de Tir sont limitées en utilisation par les munitions disponibles indiquées au-dessus de la compétence. Quand celle-ci arrive à 0, la compétence est rendue inactive tant que des munitions supplémentaires n'ont pas été ramassées. Les munitions ne peuvent dépasser le maximum déterminé par la variable **ammoMax** (int)

Structure :

1. le joueur active la compétence, la portée s'affiche autour du personnage sélectionné ainsi qu'un trait entre l'arme du personnage et la souris pour indiquer s'il y a des obstacles entre le personnage et l'endroit visé, un cercle

apparaît autour du personnage jouable pour indiquer la zone de son qui sera produite durant l'attaque.

2. Quand le joueur clique gauche sur la cible, le personnage se dirige vers la cible de manière à n'avoir aucun obstacle entre eux et déclenche l'attaque.
3. Une fois ces conditions remplies, l'attaque est déclenchée, les animations et le cooldown sont démarrés et la réserve de munitions diminue de 1

Statistiques :

- **skill/CD** : 8s
- **skill/Damage** : 4 (kill garanti)
- **skill/Range** : 8
- **skill/RangeNoise** : 6
- **skill/Blood** : False

### 16.8.3. Compétences de Classe

#### 16.8.3.1. Heal

Classe : Medic

Le Heal permet à un personnage jouable de soigner un allié ou lui-même d'un point de vie.

1. Le joueur active la compétence, la portée s'affiche autour du personnage sélectionné.
2. Quand le joueur clique gauche sur la cible, le personnage se dirige vers la cible et déclenche l'attaque au contact du Dummy-Range avec la cible. si le joueur décide que le personnage se soigne lui-même la compétence est immédiatement déclenchée (/!\ si la vie de la cible est déjà au maximum, le personnage sélectionné refuse de bouger et prononce une phrase indiquant que cette action est impossible/!\)
3. Au contact, le personnage visé se fait soigner d'un PV, le cooldown est déclenché.

Statistiques :

- **skill/CD** : 15s
- **skill/Heal** : -1
- **skill/Range** : 3
- **skill/RangeNoise** : 0

#### 16.8.3.2. Détection

Classe : Leader

La détection permet à un personnage d'activer les Outlines de tous les objets interactifs (ennemis compris) dans une zone autour de lui.

La compétence et le cooldown se déclenchent immédiatement après activation.

Statistiques :

- **skill/CD** : 10s
- **skill/Range** : 20

#### 16.8.3.3. Distraction

Classe : Espion

La distraction permet de détourner temporairement l'attention des ennemis.

## 16.9. L'inventaire

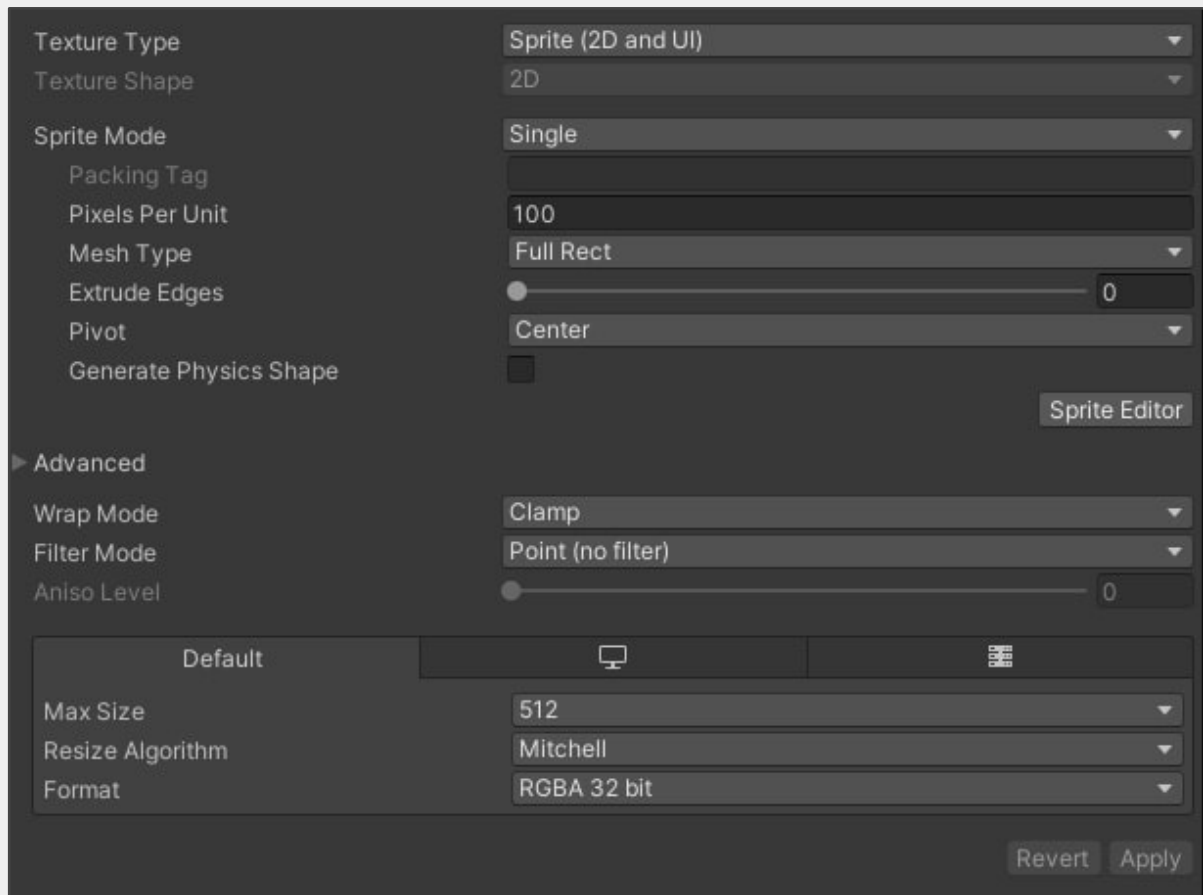
Pour **intégrer de nouveaux items** dans une scène Unity liée au projet Prohibeabest, il faut intégrer leurs **Sprites** et leurs **Prefabs**.

Les **Sprites** de l'inventaire, donc les éléments du HUD, doivent se placer dans le dossier "Assets > \_UI > InGameUI > Inventory".

**Il est TRÈS IMPORTANT de les nommer comme suit : "Inventory-NomDeLItem".**

Exemples : Inventory-Ammo, Inventory-Bottle....

Ces éléments, qui sont en réalité des images PNG, doivent être configurés comme montré sur l'image ci-dessous.



Quant aux **Prefabs**, ils doivent être placés dans le dossier "Assets > Resources > Game > Props > Items".

**Il est TRÈS IMPORTANT de les nommer simplement comme suit : "NomDeLItem".**

Exemples : Ammo, Bottle....

Ils doivent avoir comme **Tag** : **Item** et comme **Layer** : **Item**.

Aussi, ils doivent contenir :

- Un component **Box Collider** (Is Trigger : On) :  
Sert à être détecté par le curseur ;
- Le component **Script Outlinable** :  
Sert à afficher un outline blanc autour d'eux ;
- 3 **FSM** (FSM récupérables dans d'autres items) :
  - ◆ **FSM-CheckPlayerLocatePoint** :  
Sert à connaître le **PJ** qui va ramasser l'item ;
  - ◆ **FSM-Outline** :  
Sert à activer / désactiver le component **Script Outlinable** ;
  - ◆ **FSM-Interactive** :  
Sert à savoir si le joueur effectue Ctrl + Clic Gauche.

Chaque **Prefab** doit contenir deux enfants :

- **DummyGraph** (Tag : Item ; Layer : Ignore Raycast) :

C'est le modèle de l'item ;

→ **PlayerPosToGo** (Tag : Item ; Layer : Ignore Raycast) :

C'est une délimitation item-player.

Ce GameObject doit contenir un component **Box Collider** (IS Trigger : On) forcément un peu plus large que celui du root afin que le **PJ** qui doit ramasser l'item s'arrête dès qu'il entre en collision avec lui.

Cela ajoute du réalisme lors de l'animation de Pick-Up puisque le **PJ** ramasse l'item devant lui et non à ses pieds.

Une fois les **Sprites** et les **Prefabs** installés correctement avec tous leurs composants, il faut les **instancier grâce à PoolKit**.

Pour cela, il faut se diriger dans l'inspector de GLOBAL-POOL et ajouter le **Prefab** (l'item donc).

Puis, aller dans POOL-MANAGER, et créer un nouveau GameObject qui fera office de **spawner** grâce au component Spawner (Script).

Le GameObject doit être enfant de POOL-MANAGER et doit se nommer comme suit : "PoolKitSpawnerNomDeLItem".

Exemple : PoolKitSpawnerAmmo, PoolKitSpawnerBottle....

♦ ♦ ♦

Afin qu'un **PJ** puisse **ramasser puis utiliser un item**, il dispose de 4 FSM (**PlayerRoot**) :

→ **FSM-InteractionItem** :

Sert à confirmer que le **PJ** ramasse bien l'item que le joueur a sélectionné.

Ici aussi se joue l'animation de pick-up.

Il confirme également au FSM FSM-PlayerInventory qu'un item a été ramassé.

→ **FSM-PlayerInventory** :

Sert à stocker dans un tableau les items que possède le **PJ**, et envoie les données du tableau à l'UI.

→ **FSM-Selected** :

Sert à envoyer des données essentielles aux FSM de l'inventaire et de l'UI (et d'autres) au sujet de la sélection d'un PJ (ou de sa désélection).

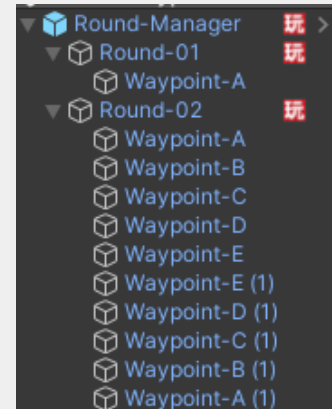
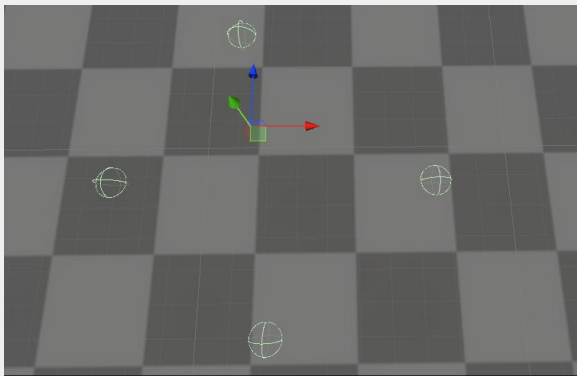
→ **FSM-UseItem** :

Vérifie quel item est sélectionné ou non.

## 17. PNJ Ennemi :

### 17.1. Création d'une ronde :

Pour ajouter une ronde à un PNJ il faut d'abord créer la route en plaçant les différents "Waypoint" de sorte à faire la ronde voulue, vous nommerez les "Points" de cette manière : Point-A, Point-B, etc.... Les rondes doivent **impérativement** être placées au sol. Les points doivent être orientés dans la direction où l'on veut que l'ennemi regarde.



Tous les Waypoints posés devront être enfant d'un empty object appelé : "Round-Num". Ces même empty doivent être enfant du Round-Manager

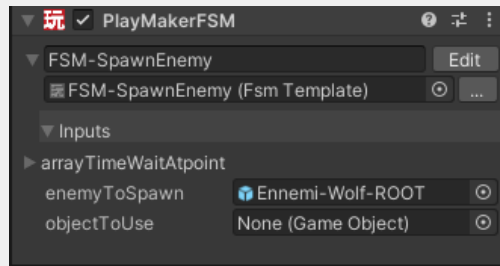
Ensuite, placer sur les empty "Round-Num" le FSM : FSM-SpawnEnemy. Ensuite, mettre dans la variable "enemyToSpawn" le prefab de l'ennemi que l'on veut créer. Afin de configurer la ronde de l'ennemi,

l' "arrayTimeWaitAtPoint" détermine le temps en seconde que passera le PNJ à l'arrêt sur un WayPoint.

**⚠ Cet array doit posséder autant d'entrées que de WayPoints ⚠**

S'ils ont qu'un seul point dans une ronde, les ennemis peuvent utiliser le mobilier. Pour utiliser du mobilier (chaise/banc), il faut qu'il soit placé à la bonne portée du point de la ronde et doit être orienté dans le même sens que lui.

Une fois le mobilier en place, il suffit de la drag&drop dans la variable "objectToUse".



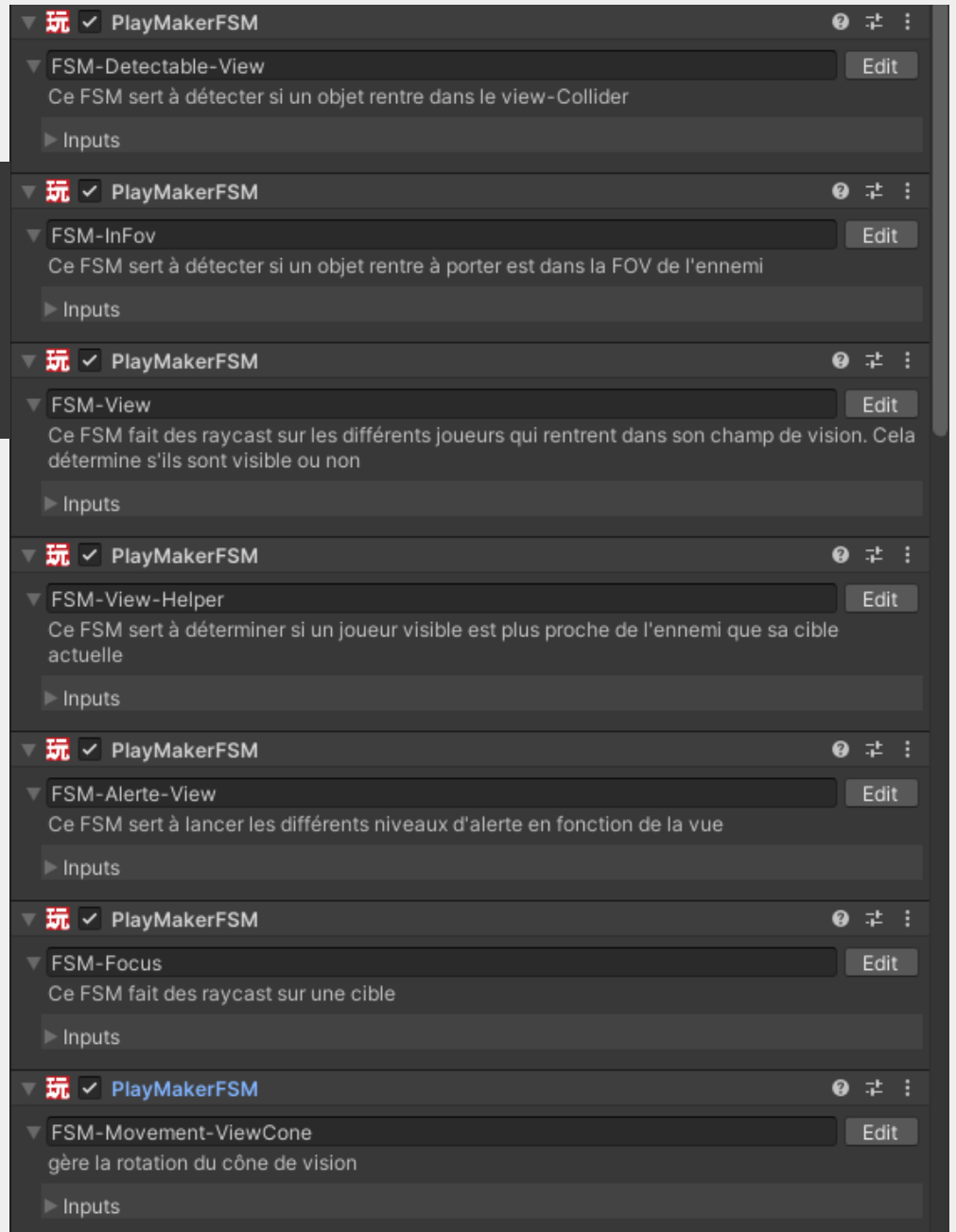
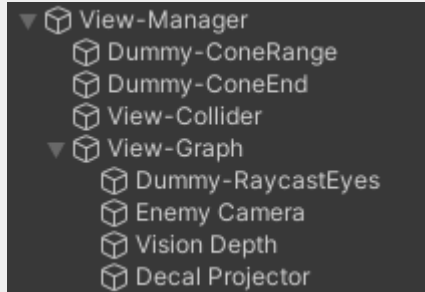
## 17.2. Détection des joueurs :

Les ennemis disposent tous d'un dummy appelé "**Detection-Manager**" qui est parent de tous les Unity objects nécessaires au fonctionnement des FSM de détections.

## 17.2.1. La vision

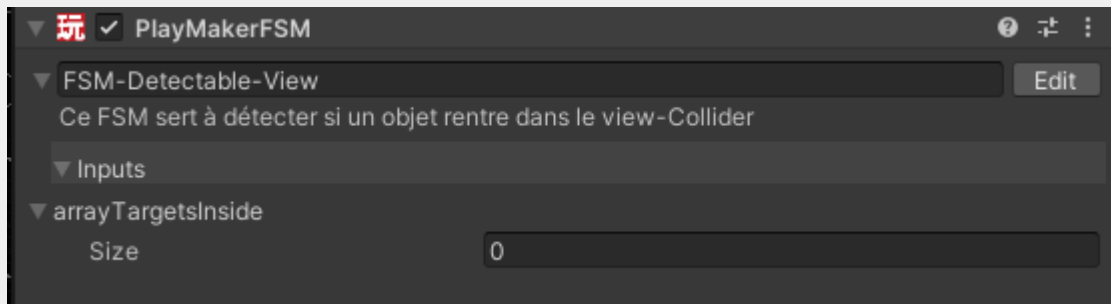
### 17.2.1.1. Général

La vue fonctionne grâce à 6 scripts et 8 Unity objects tous taggés "View".



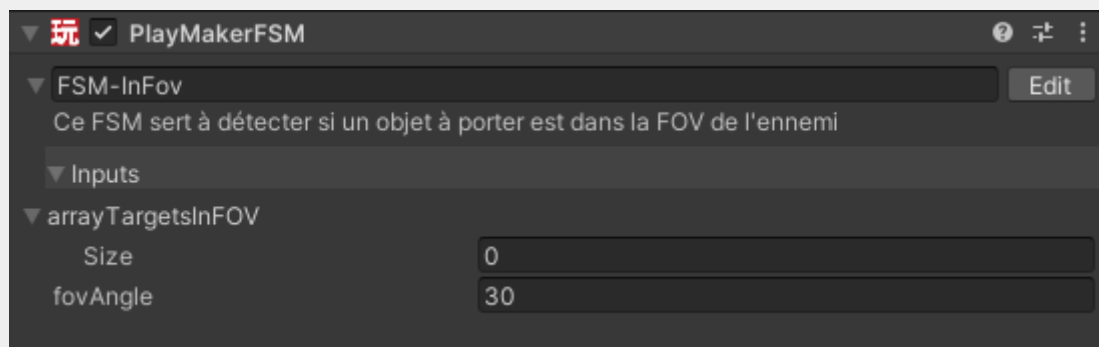
#### 17.2.1.2. FSM-Detectable-View

Crée un array contenant tous les objets d'intérêt présent dans le view-Collider.  
Le view-Collider représente la distance maximale de la vue de l'ennemi.



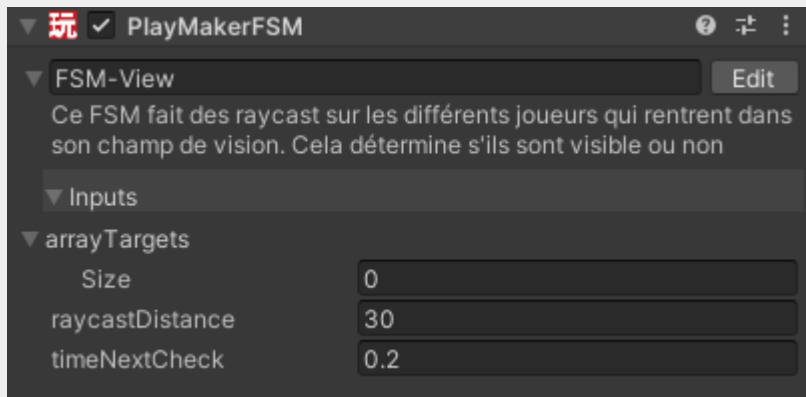
#### 17.2.1.3. FSM-InFov

Check à intervalle court et régulier si les différents objets présents dans l' "arrayTargetsInside" sont dans la Fov de l'ennemi. La Fov est calculée par rapport à l'orientation du View-Manager.



#### 17.2.1.4. FSM-View

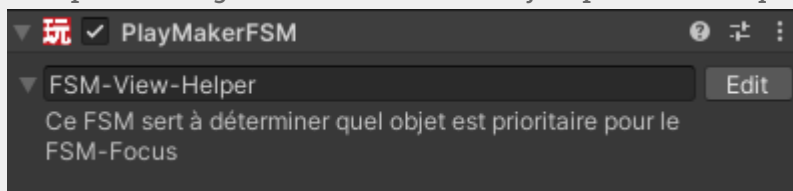
Récupère les objets de l'array "arrayTargetInsideFOV" et vérifie si les objets ne sont pas cachés par un obstacle.



L'endroit où le raycast est envoyé dépend du type de l'objet (humanoid, objet, ...).

## 17.2.1.5. FSM-View-Helper

Compare les tags et la distance des objets pour établir quel objet doit être focus.



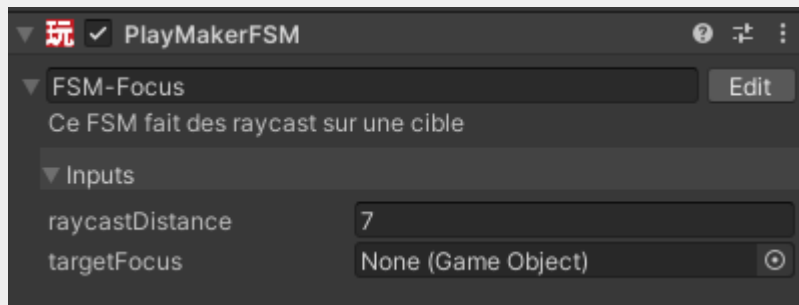
## 17.2.1.6. FSM-Alerte-View

Va déterminer le niveau d'alerte en fonction de l'état du cône de vision. Il détermine notamment le remplissage de la jauge ainsi que sa couleur et déclenche des changement d'état (recherche ou alerte)



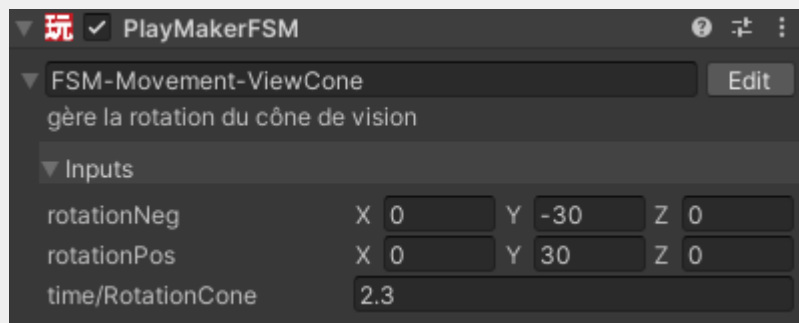
## 17.2.1.7. FSM-Focus

Traque la cible déterminée par FSM-View-Helper et vérifie qu'elle soit visible à chaque frame.



## 17.2.1.8. FSM-Movement-ViewCone

Détermine les mouvements du cône de vision



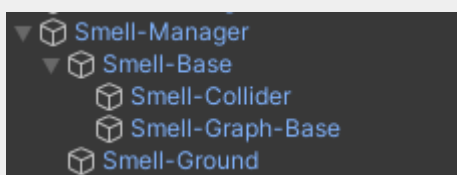
## 17.2.2. L'odorat

### 17.2.2.1. Général

L'odorat n'est porté que par les loups et les reptiles. Il est géré par 4 FSM différents et suit globalement le même fonctionnement que la vue à la différence que l'odorat peut sentir un objet d'intérêt caché derrière un obstacle.

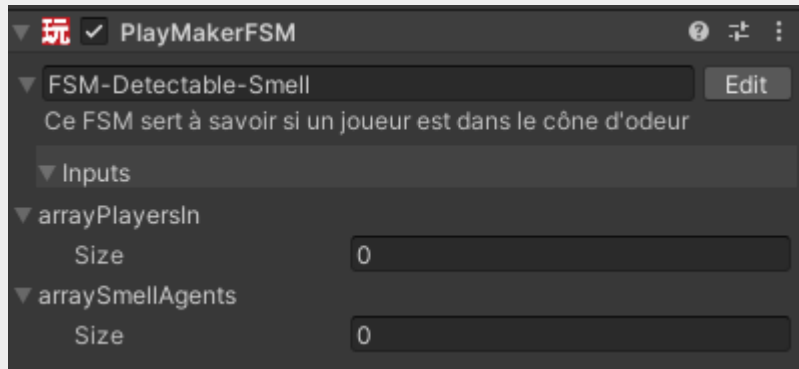
Pour cela, le FSM-Smell crée un SmellAgent invisible capable de se déplacer dans la zone d'odorat en suivant un pathfind jusqu'à la cible.

Ces FSM fonctionnent à l'aide de 4 objets enfant du root taggés "Smell"



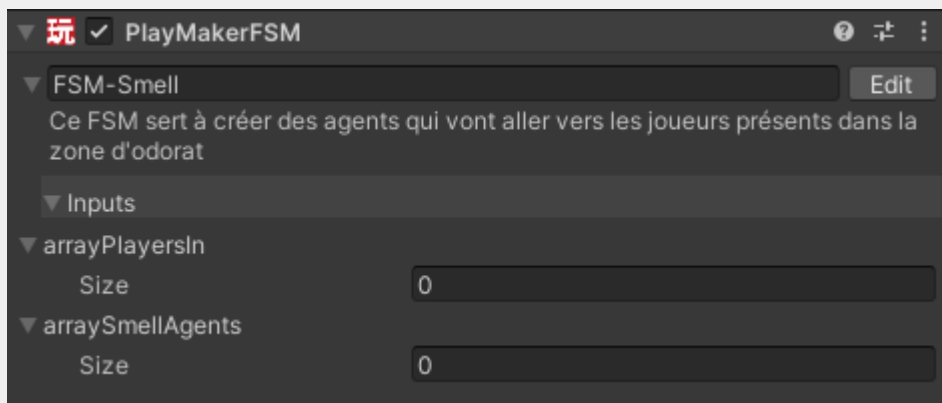
#### 17.2.2.2. FSM-Detectable-Smell

Créer un array contenant tous les objets d'intérêts présents dans la zone d'odorat.



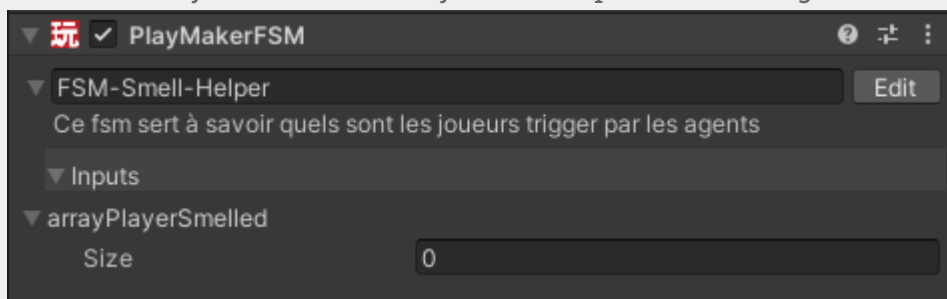
#### 3. FSM-Smell

Créer des SmellAgents pour chaque objet présent dans la zone d'odorat



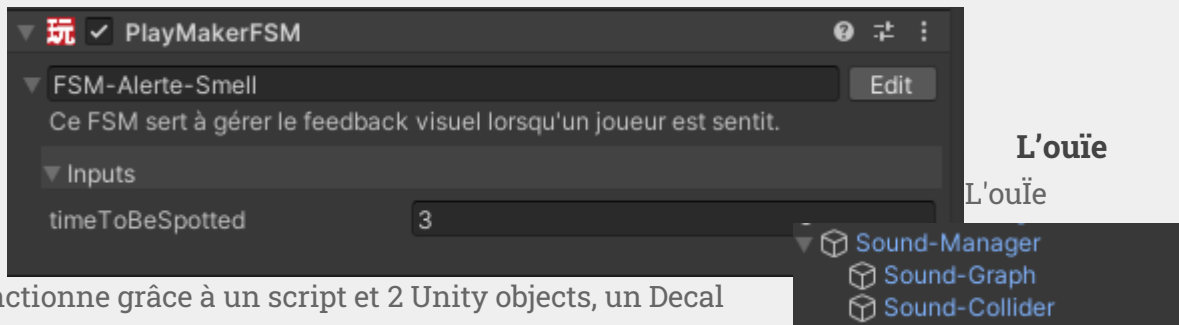
#### 17.2.2.4. FSM-Smell-Helper

Créer un array contenant les objets atteint par les SmellAgents

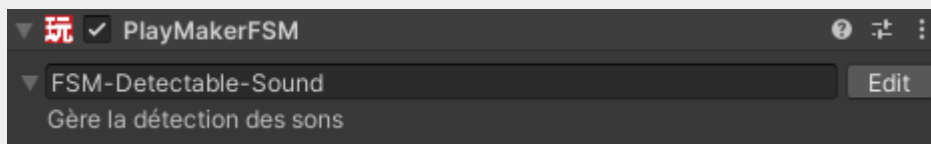


#### 17.2.2.5. FSM-Alerte-Smell

Gérer le feedback visuel de la zone d'odorat (de vert à rouge ou de rouge à vert si aucun objet n'est détecté) et déclenche les réaction du PNJ



fonctionne grâce à un script et 2 Unity objects, un Decal pour le visuel et un collider pour le gameplay taggés "Sound". (NOTE : Seul les grands et petits mammifères disposent du Decal)

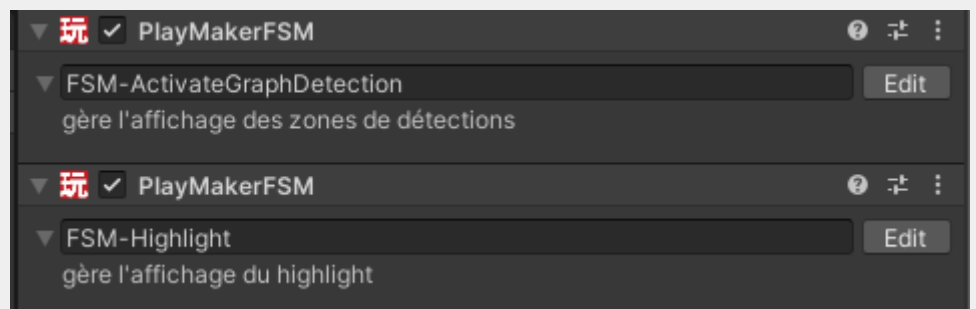


Ce FSM se contente de vérifier les collisions des objets sons créés par les joueurs ou les objets interactifs et envoie un signal au FSM-Alerte pour que le PNJ réagisse aux sons.

### 17.3. Les feedback visuels

Les feedback visuels sont gérés par deux FSM :

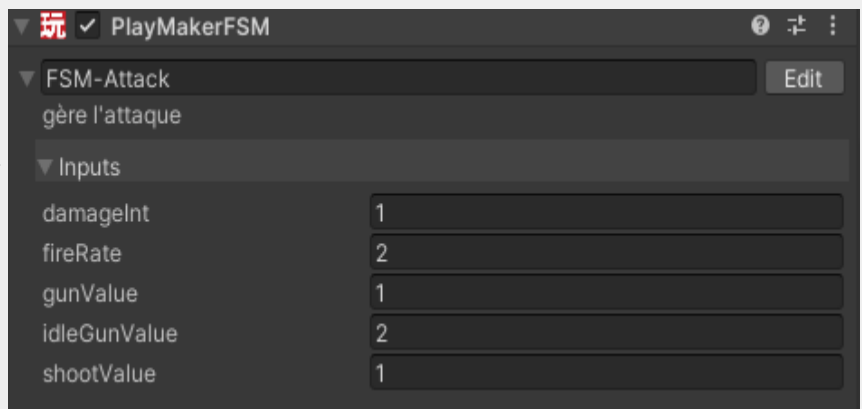
- Le FSM-ActivateGraphDetection qui s'occupe d'afficher les zones de détection
- Le FSM-Outline s'occupe d'afficher l'outline de l'ennemi quand le signal lui est envoyé



## 17.4. L'attaque

Le FSM-Attack se contente de recevoir le signal ENEMY/ATTACK/PLAYER du FSM-Movement-Alerte quand un personnage jouable est visible pendant qu'un ennemi est en alerte. Le PNJ va tirer régulièrement en fonction de la valeur fireRate.

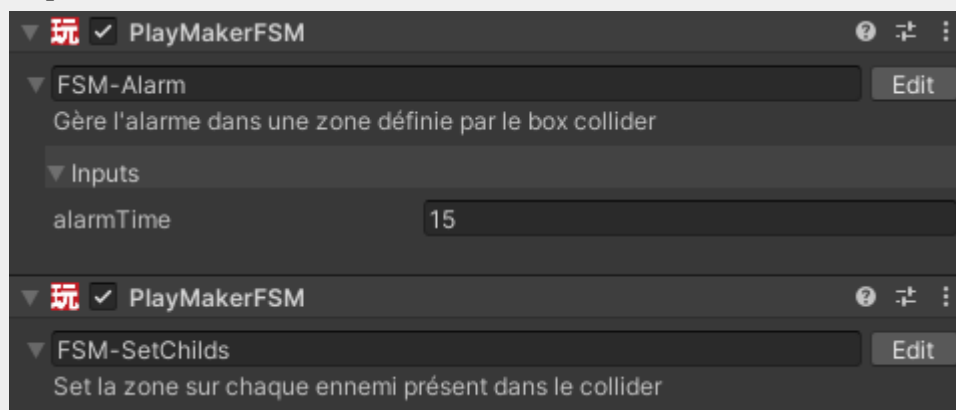
Après chaque tir, le PNJ va vérifier les PV de la cible et s'arrêter quand ceux-ci arrivent à 0.



## 17.5. L'alarme

Le système d'alarme est géré par un prefab appelé **Zone-XX** (XX étant le numéro de la zone 01, 02, 03, etc...)

Le prefab contient une Box collider et deux FSMs :



Le FSM-Alarme va déclencher le mode alerte sur tous les ennemis présents dans le collider quand l'un d'entre eux rentre en mode alerte. La variable alarmTime permet de régler le temps que dure l'alarme dans la zone.

Le FSM-SetChilds permet de régler la variable CurrentZone dans le FSM-Alerte des ennemis

## 17.6. Le Sang

A leur mort, en fonction du type de l'attaque qui les a tué, les ennemis laissent une tâche de sang au sol. Cette tâche est un objet pouvant être repéré par les ennemis et les poussent à chercher la zone dans laquelle elle a été vue. De plus, quand un

personnage jouable marche dans cette tâche, il laisse des traces de pas sur son passage. Ces traces peuvent également être vues par les ennemis et les poussent à suivre le chemin tracé.

Les Tâches de sang sont des préfabs appelé **Blood-Pool** qui possède une sphere collider et un FSM appelé **FSM-Blood** qui va gérer les traces de pas produites par un personnage jouable en les rangeant dans une array qui permet aux ennemis de savoir quel chemin suivre quand ils repèrent les traces.

Les traces possèdent un **FSM-Fade** qui les fait disparaître après quelques secondes et qui permettent de se retirer de l'array situé dans **FSM-Blood**.

## 18. See Through / Fade

### 18.1. See Through :

1. Pour que le Shader fonctionne dans la scène cible il est d'abord primordial de setup celle-ci de cette façon;

Créez un Empty que vous nommerez *"See-Through Shader Manager"*.

Il faudra y ajouter les component suivant :

- *"STS-Global Shader Replacement"* que vous paramétré comme ceci ;



*"Vous trouverez le Shader "SeeThrough" dans le répertoire suivant Materials ⇒ Shader-SeeThrough".*

- *"STS-Player Position Manager"* dans lequel il faudra ajouter les GameObject suivant de cette façon ;

2. Mise en place d'un Bâtiment bénéficiant d'un intérieur. Il est nécessaire que les "Mesh" de la modélisation soit séparé de cette de cette manière :

- DummyMeshStayVisible. comportant les Mesh qui resteront visibles (exemple : les murs intérieurs, le mobilier etc...).
- DummyMeshSeeThrough. Comportant les Mesh qui seront invisibles (exemple : les toits.)



Placer ensuite le Component *"STS-Group Shader Replacement"* sur le *"DummyMeshSeeThrough"*

De cette façon :

*"En cours.."*

## 18.2. Fade :

Le "Fade" des bâtiments est géré par le FSM Template **"FSM-Fade"** trouvable dans le répertoire : ( *"Assets⇒Playmaker⇒Template"* ). Le FSM doit être posé sur le "root" du bâtiment ciblé.

Pour "SetUp" le "Fade" il suffit de mettre les "Components" box collider ainsi que "Nav Mesh Obstacle" sur le root du bâtiment. (Ses deux "component" doivent faire la taille du bâtiment ciblé).

Le bâtiment et ses "child" doivent être sur le layer "Building".

## Animations

### Float :

yPosition : détermine si un personnage est accroupi

xSpeed : détermine la vitesse du personnage

rotationLR : détermine si le personnage tourne à droite ou à gauche

rotationFB : détermine si le personnage va vers l'avant ou fait demi-tour.

### Trigger :

startMovement : détermine si un personnage se met à se déplacer

endMovement : détermine si un personnage s'arrête

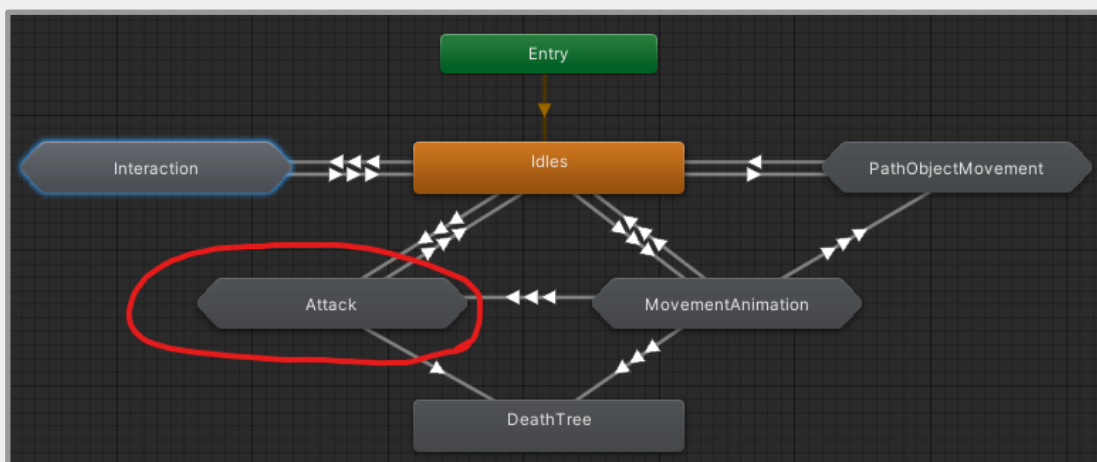
changeDirection : détermine si un personnage change de direction.

ladderTrigger : détermine si un personnage monte sur une échelle.

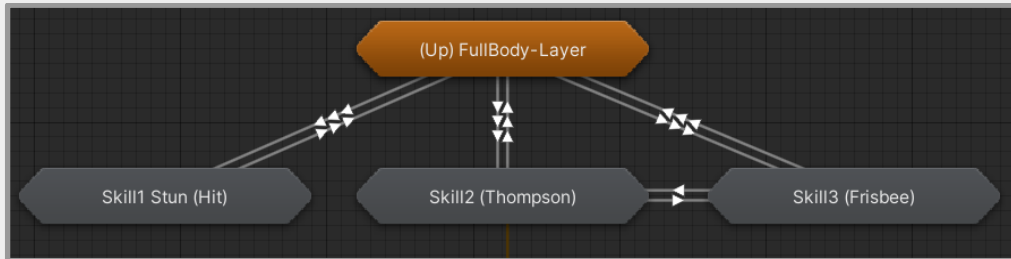
ladderTriggerOut : détermine si un personnage descend d'une échelle.

## Système de compétences

Chaque Animator de Player possède un Sub-State nommé "Attack".



Il est divisé entre 5 Sub-State représentant les différentes compétences du personnage, tous nommé de la façon suivante : SkillX (+ Stun pour la variante au corps à corps).



Ces sub-State contiennent 4 Blend Tree représentant les différents état de la compétence : Draw, Idle, Attack et Idle. Ces Blend Tree sont nommés de la façon suivante : *(Etat en question)*SkillXTree.

Exemple : DrawSkill1StunTree.

Les Sub-States utilisent des variables nommées de la façon suivante :

- drawSkillXBool (*Bool*), permettant d'entrer dans le BlendTree Draw lorsqu'il est activé, et permettant d'entrer dans le BlendTree Holster lorsqu'il est désactivé.
- attackSkillXTrigger (*Trigger*), permettant à n'importe quel instant d'entrer dans le BlendTree Attack de la compétence en question.

Il est possible de rajouter des variables de type Bool pour les cas exceptionnels.

Afin d'assurer le bon fonctionnement de la compétence, les entrées au Sub-State des compétences doivent venir de l'Idle, ainsi que des BlendTree présents dans la Sub-State Movement.

Les transitions n'ont pas de normes et sont placées en fonction de la compétence en question.

**⚠** Certaines transitions sont obligatoires :

- Dans le Layer 0 : chaque BlendTree doit posséder une transition vers le DeathTree sans "HasExitTime" et avec la condition : deathTrigger (*Trigger*).
- Le BlendTree Holster doit avoir une transition vers tous les autres BlendTree Attack des autres Sub-State Skill sans "HasExitTime" et avec la condition : attackSkillXTrigger (*Trigger*).

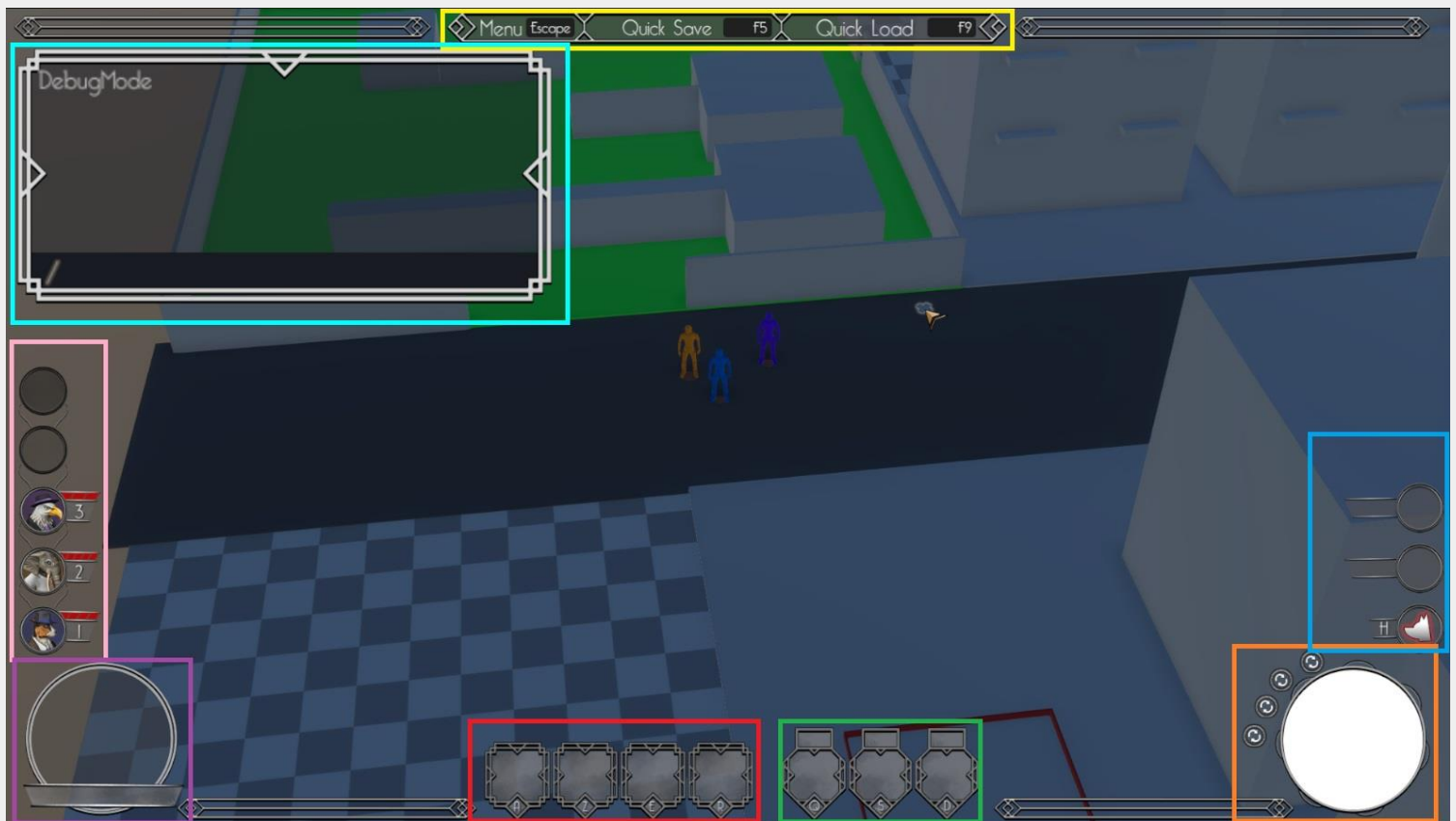
Certaines compétences nécessitent uniquement le haut du corps du personnage, dans ce cas il faut appliquer la même chose dans le Sub-State Attack du Layer 1.

⚠ Le Sub-State Attack du Layer 1 doit être une copie de celui présent dans le Layer 0 afin d'éviter les conflits, même si les animations ne sont pas concernées par le Layer. Ne doivent pas être présents sur le Layer 1 :

- Les transitions vers DeathTree.
- Les transitions d'attaque entre les Sub-State Skill.

## 19. Le HUD

### 19.1. Général



Le HUD est composé de 7 parties :

- La Character List
- La grande Icône
- Les Compétences
- L'inventaire
- La Mini-Map
- La Barre de Menu supérieure
- La Barre des features

L'entièreté de l'affichage des éléments du HUD sont gérés par l'**UI Root** (à l'exception de la mini-map qui est géré par le Camera Manager

### **19.1.1. La Character List**

FSM associé dans l'UI ROOT : FSM-Charlist; FSM-HealthBars

La Character List représente la liste des personnages jouables présents dans la scène, elle est composée d'une icône, d'un onglet avec un chiffre et d'un onglet "Barre de Vie"

- Cliquer sur L'icône d'un personnage permet de le sélectionner.
- Cliquer sur un l'icône d'un personnage déjà sélectionné permet de centrer la caméra sur celui-ci.
- Lorsqu'un personnage est sélectionné, les compétences, l'inventaire et la grande icône deviennent colorés avec la couleur de ce dernier.
- Sélectionner un personnage désélectionne tous les autres.
- Le chiffre correspond à la touche qu'il faut appuyer pour l'activer

### **19.1.2. La grande Icône**

FSM associé dans l'UI ROOT : FSM-Icon

La grande icône permet de savoir quel personnage est sélectionné. Quand un personnage est sélectionné, une icône haute définition s'affiche dans le cercle et son nom s'inscrit dans l'ornement situé en dessous. Le fond de l'ornement prend la couleur du personnage.

Quand aucun ou plusieurs personnages sont sélectionnés, l'icône n'affiche aucune image, l'ornement devient blanc et n'affiche aucun nom.

### **19.1.3. Les Compétences**

FSM associé dans l'UI ROOT : FSM-SkillBar

La barre de compétences permet d'afficher et d'activer les compétences du personnage sélectionné.

Cliquer sur un icône sélectionne la compétence associée. il suffit ensuite de cliquer sur la cible ou à l'endroit où l'on souhaite utiliser la compétence sélectionnée.

Quand aucun ou plusieurs personnages sont sélectionnés, les icônes n'affiche aucune image, le fond des icônes deviennent blanc et ne sont plus cliquables.

#### 19.1.4. L'inventaire

**FSM** associés dans **UIRoot** :

→ **FSM-Inventory** ;

→ **FSM-Inventory-Display**.

La barre d'inventaire permet d'afficher ou non les items du **PJ** sélectionné.

Cliquer sur un bouton de l'inventaire ou passer par l'input du clavier sélectionne l'item associée.

Chaque item s'utilise d'une manière différente.

Lorsqu'aucun ou plusieurs **PJs** sont sélectionnés, les **Sprites** ne sont alors plus visibles, ainsi le fond des boutons devient gris foncé : ils ne deviennent plus cliquables.

#### 19.1.5. La Mini-Map

FSM associés dans l'UI ROOT : FSM-Minimap ;

FSM associé dans le Camera-Manager : FSM-MiniMap-Click

La Mini-Map permet d'afficher la carte du niveau en cours ainsi que la position des personnages (PC et NPC) dans celui-ci.

L'orientation de la carte est la même que l'orientation de la caméra principale dans le niveau.

Cliquer sur la Mini-Map déplace la caméra principale à l'endroit correspondant.

La Mini-Map est également équipée de 4 boutons permettant de bouger la caméra. Les deux boutons du haut permettent de tourner la caméra et les boutons de gauche permettent de zoomer et dézoomer.

#### 19.1.6. La Barre de Menu

FSM associés : WIP

La barre de Menu contient les boutons relatifs au menu et aux sauvegardes.

Le bouton Menu ouvre le Menu principal

Le bouton Quick save crée une sauvegarde rapide et écrase la sauvegarde précédente

Le bouton Quick load charge la dernière sauvegarde effectuée.

### 19.1.7. La Barre des Features

FSM associé dans l'UI Root : **FSM-Outline-Button**

**FSM-Journal-Button**

**FSM-Time-Button**

La barre des features contient les boutons liés à des fonctionnalités spéciales.

Le bouton "Showdown" permet d'activer ou de désactiver le Mode Showdown

Le bouton "Display Alarm" permet d'afficher ou de cacher les zones d'alertes dans le niveau

Le bouton "Speed" permet d'accélérer le temps ou de le remettre à vitesse normale

Le bouton "Journal" permet d'accéder au Journal dans lequel sont consignés les dialogues importants et les objectifs.

Le bouton "Outline" permet d'afficher tous les objets interactif du niveau.

## 19.2. L'UI Root

L'UI Root est le Manager responsable de tout changement dans l'affichage de l'UI

Il est composé d'un FSM pour chaque partie du HUD en plus de certaines fonctions nécessaires à l'ergonomie.

Il est situé dans le dossier Assets/Resources/Game/\_Manager.

### 19.2.1. FSM-CharList

Gère l'affichages des boutons de sélections

N'envoie aucun signal, reçoit des signaux en provenance des boutons de sélection uniquement.

Variables :

Il y a 3 familles de variables

- **Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- **Data** : données nécessaires au fonctionnement (floats, strings, int...)
- **Sprites and Labels** : cibles à changer (game objects)

### 19.2.2. FSM-Color

Gère la colorisation du HUD en fonction du personnage sélectionné.

Events importants :

- PLAYER/SELECTED : Envoyé par un PJ

#### Variables :

Il y a 3 familles de variables

- **Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- **Data** : données nécessaires au fonctionnement (floats, strings, int...)
- **Sprites and Labels** : cibles à changer (game objects)

#### **19.2.3. FSM-GetSelf**

Fait de L'UI Root une Variable Globale

Ne reçoit ni n'envoie aucun event

#### Variable :

- uiRoot

#### **19.2.4. FSM-HealthBars**

Gère L'affichage des barres de vies sur la character List

#### Event importants :

- PLAYER/UPDATE HEATHBAR : Envoyé depuis le "FSM-Hp-Manager", met à jour la barre de vie de l'envoyeur

#### Variables :

Il y a 3 familles de variables

- **Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- **Data** : données nécessaires au fonctionnement (floats, strings, int...)
- **Sprites and Labels** : cibles à changer (game objects)

#### **19.2.5. FSM-Inventory**

Permet d'afficher / cacher les items de l'inventaire, ainsi que le nombre d'items qu'il comporte.

#### **19.2.6. FSM-Inventory-Display**

Permet l'affichage du contour de chaque bouton d'inventaire de l'UI (Cont-Inventory-Button-n).

#### **19.2.7. FSM-MissionFailed**

Gère L'affichage de l'écran de Game Over

#### Events importants :

- **PLAYER/IS DEAD** : envoyé depuis le "FSM-Hp-Manager"

#### Variables :

Il y a 3 familles de variables

- **Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- **Data** : données nécessaires au fonctionnement (floats, strings, int...)
- **Sprites and Labels** : images et textes à modifier (game objects)

#### **19.2.8. FSM-SkillBar**

Gère l'affichage et les signaux de la barre de compétences

#### Variables :

Il y a 3 familles de variables

- **Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- **Data** : données nécessaires au fonctionnement (floats, strings, int...)
- **Sprites and Labels** : cibles à changer (game objects)

#### **19.2.9. FSM-Outline-Button**

Gère l'affichage du bouton permettant d'afficher l'outline de tous les objets

#### Variables :

Il y a 3 familles de variables

- Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- Data** : données nécessaires au fonctionnement (floats, strings, int...)
- Sprites and Labels** : cibles à changer (game objects)

Il communique avec tous les autres objets en faisant un global event sur tous les FSM-Outline.

#### **19.2.10. FSM-ChangeButton**

Gère l'affichage et les changements dans la compétence de Corps à Corps

#### Variables :

Il y a 3 familles de variables

- Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)
- Data** : données nécessaires au fonctionnement (floats, strings, int...)
- Sprites and Labels** : cibles à changer (game objects)

Il communique avec le *FSM-Skill-1* du player-root sélectionné et le FSM-Change-Button sur le widget Cont-Change-Button

#### **19.2.11. FSM-Camera-Buttons**

Gère l'affichage des boutons de mouvement de caméra

##### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

#### **19.2.12. FSM-Journal-Button**

gère l'affichage du bouton de journal.

##### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

#### **19.2.13. FSM-Journal-Window**

Gère l'affichage de la fenêtre de journal

##### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

#### **19.2.14. FSM-Pause-Menu**

Gère l'affichage du Menu Pause.

##### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

Il communique avec l'objet Cont-Pause-Menu

### 19.2.15. FSM-Time-Button

Gère l'affichage du bouton d'accélération du temps

#### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

### 19.2.16. FSM-Crouch-Button

Gère l'affichage du bouton de Crouch

#### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

### 19.2.17. FSM-Options-Menu

Gère l'affichage du menu Options dans le menu Pause.

#### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

### 19.2.18. FSM-ChangeButton-Eager

Gère l'affichage du bouton de changement de compétence de tir d'eager

#### Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

#### 19.2.19. FSM-Action-Button

Gère l'affichage du bouton de d'action (ctrl)

Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

#### 19.2.20. FSM-Loading-Screen

Gère l'affichage de l'écran de chargement

Variables :

Il y a 3 familles de variables

**Containers** : contiennent l'ensemble des éléments qui composent un objet (game objects)

**Data** : données nécessaires au fonctionnement (floats, strings, int...)

**Sprites and Labels** : cibles à changer (game objects)

### 19.3. Autre types de HUD

Certains éléments de HUD apparaissent de manière contextuelle et temporaire sur l'écran de jeu et sont séparés du HUD classique. ces éléments ne sont pas interactifs et ne servent qu'à afficher des informations au joueur.

#### 19.3.1. Les barres de vie

En plus des barres de vies présentes sur dans la Character List Des barres de vie apparaissent de manière au dessus de la tête des personnages (Joueurs ET non joueurs) quand ceux-ci subissent un changement de points de vie (dégât ou soin)

Ces barres de vie sont des prefabs stockés dans le l'UI-POOL. Elles sont gérées par les **FSM-HealthBar** présents sur les personnages. Les personnages sont également équipés d'un empty object au-dessus de leur tête servant de point d'ancrage à la barre de vie appelé **Anchor-HealthBar**. A l'init de **FSM-HealthBar**, le point d'ancrage est attribué à un fsm d'ancrage présent sur les barre de vie appelé **FSM-Anchor**.

### **19.3.2. Les icônes de cachette**

Quand un personnage jouable se cache dans une cachette d'un autre type que le buisson, une icône du personnage apparaît pour indiquer au joueur qui utilise la cachette.

### **19.3.3. Les icône d'ennemis**

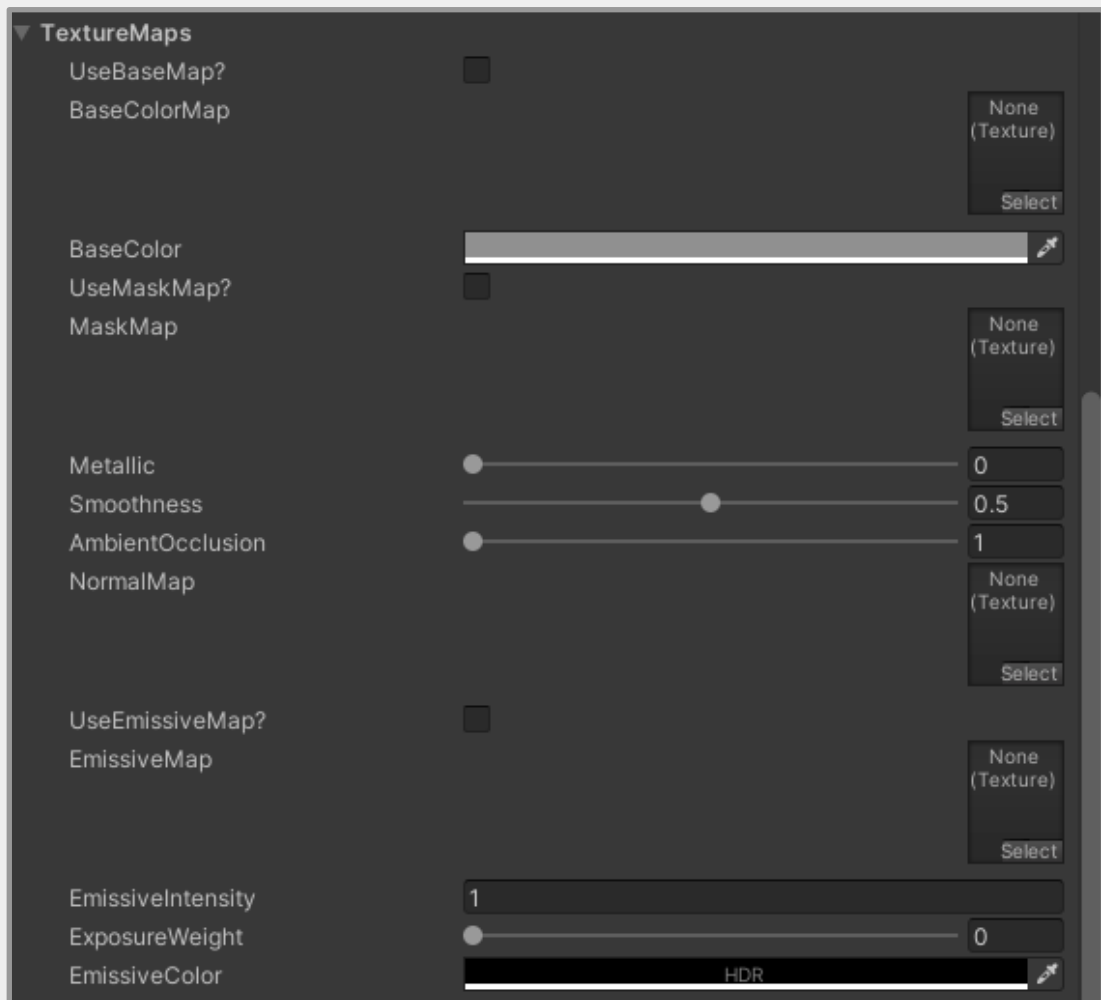
Les ennemis possèdent différentes icônes apparaissent a- dessus de leur tête pour signaler ses différents états au joueur.

Les icônes sont géré par le **FSM-Icons** présent sur chaque ennemis.

## **20. Highlight**

Le Shader Highlight est un shader à appliquer à toutes les textures des objets devant posséder un Highlight (PROPS sauf buisson).

Les différentes données de la Texture dans le Material peuvent être mis directement via les inputs du Shader (Base Color, Mask Map, Normal Map et Emissive Map).



Afin que l'Highlight soit actif en jeu, chaque objet contenant le Shader doivent contenir également un script template "FSM-Highlight".

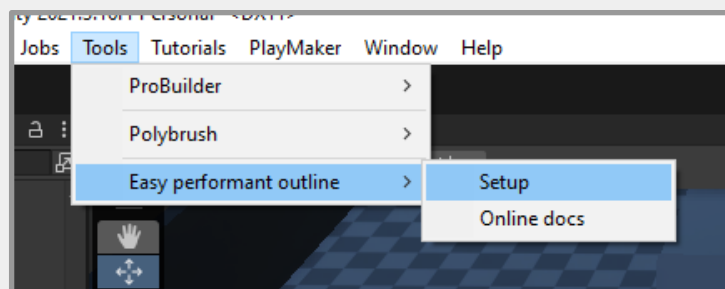
## 21. Outline

### Explication du Package

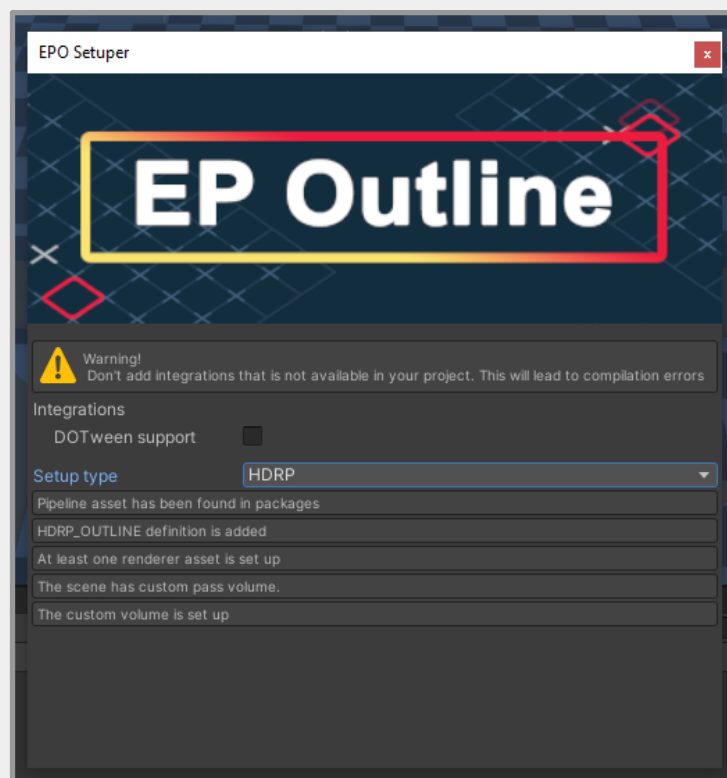
Les Outlines dépendent de 2 composants pour fonctionner :

- Un Outliner, présent sur la Main Camera et gère les différents outlines
- Un Outlinable, présent sur les objets qui doivent contenir un Outline.

Pour faire fonctionner le Package dans une scène Unity, il faut set up au préalable dans Tools/Easy performant outline/Setup.

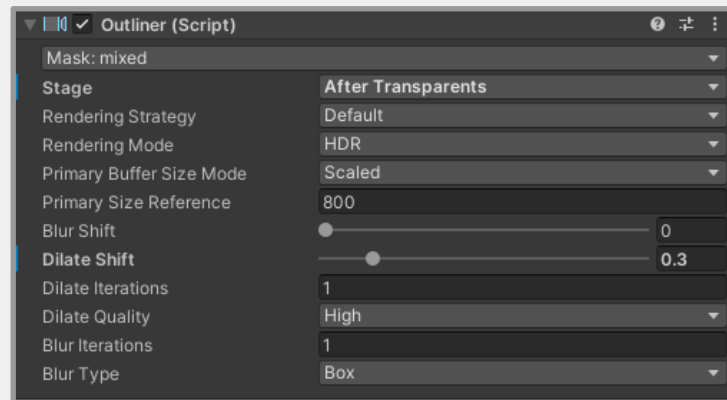


Puis, sélectionnez le moteur de notre projet unity (HDRP), et appuyez sur "Add" jusqu'à ce que le bouton n'apparaisse plus.



## Outliner

L'Outliner est un composant présent sur la Main Caméra. Il gère les outlines selon les layers attribués. La Main Camera peut (*doit*) posséder plusieurs Outline pour gérer plusieurs Outline.



\*Composant "Outliner" sur la Main Camera

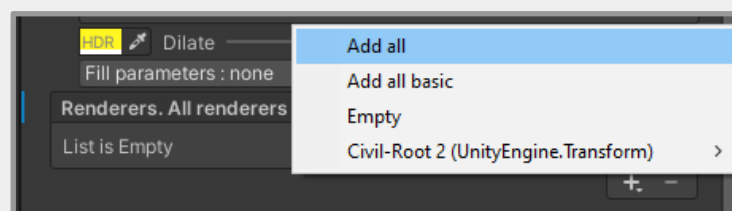
Il possède plusieurs paramètres :

- Mask : permet de paramétrer quel layer d'outline l'outliner gère (*plusieurs possible*)
- Rendering strategy : affiche ou non les outline des objets indépendamment des autres (*ils ne vont pas s'assembler entre eux*)
- Blur Shift : permet de créer un effet de flou autour de l'outline
- Dilate Shift : règle la taille des outlines (*entre 0 et 2*)
- Dilate Iterations : règle la taille des outlines (*int*)

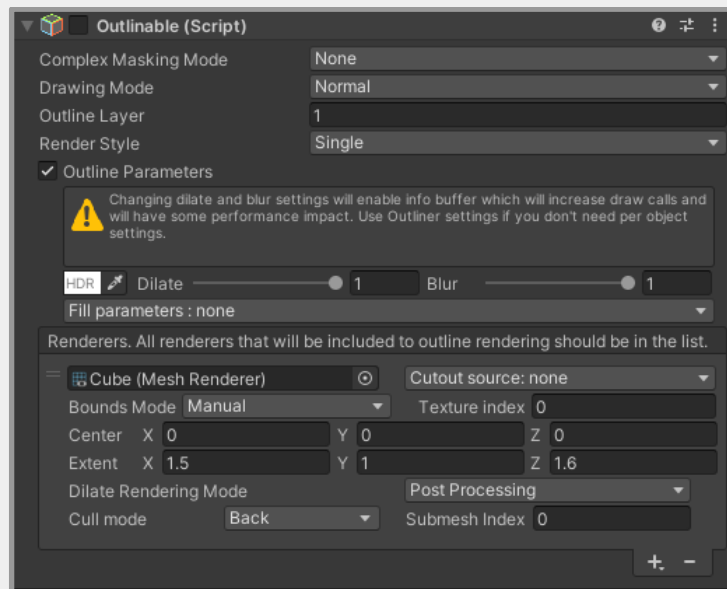
## Outlinable

L'outlinable est un composant présent sur un game object possédant un Outline.

De base, le composant ne rend aucun Outline car aucun mesh ne lui est attribué. Pour ajouter des mesh il faut cliquer sur le + dans la liste des rendus puis sélectionner "Add All" afin de rajouter tous les mesh présents parmi les enfants de l'objet.



Vous pouvez également sélectionner à la main les éléments dont le Outline doit y être affecté, tout comme vous pouvez en supprimer.



\*Composant "Outlinable" sur un PROPS

Il possède plusieurs paramètres :

- Outline Layer : permet de se mettre sur un Layer et se fait gérer par l'Outliner correspondant sur la Main Camera.
- Render Style : permet de différencier l'outline de l'objet lorsqu'il est caché par un obstacle et lorsqu'il est directement visible par la Main Camera.
- Fill parameters : permet de créer des styles d'outlines différents (*remplissages, hachures etc..*)
- Dilate : permet de changer la taille de l'outline
- Bound Mode : permet de définir une surface autour de l'objet dans lequel l'outline apparaît (*permet d'éviter certains artéfacts dans le rendu*).

## Intégration

Le projet possède des scripts permettant d'afficher les Outlines, tous rangés en template dans Asset/PlayMaker/Template/Outline. Ils sont classés dans la catégorie "Outline" des templates.

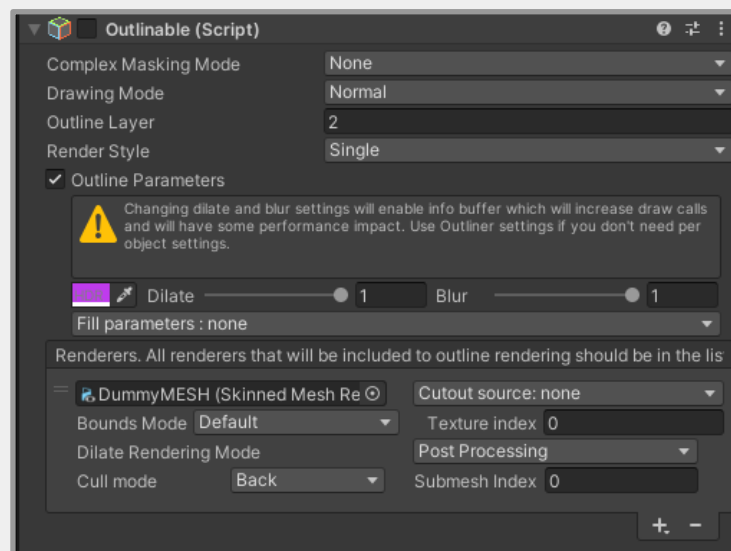
- FSM-Outline-PC : permettant d'afficher les Outlines en roll over et pendant la sélection des personnages jouables.
- FSM-Outline-PNJ : permettant d'afficher les Outlines en roll over et lorsque le bouton Outline est actif des PNJ (et leurs corps inertes).

- FSM-Outline-PROPS : permettant de gérer l'outline de tous les éléments interactifs de la scène, cas particulier compris.

## Intégration Outline d'un PC

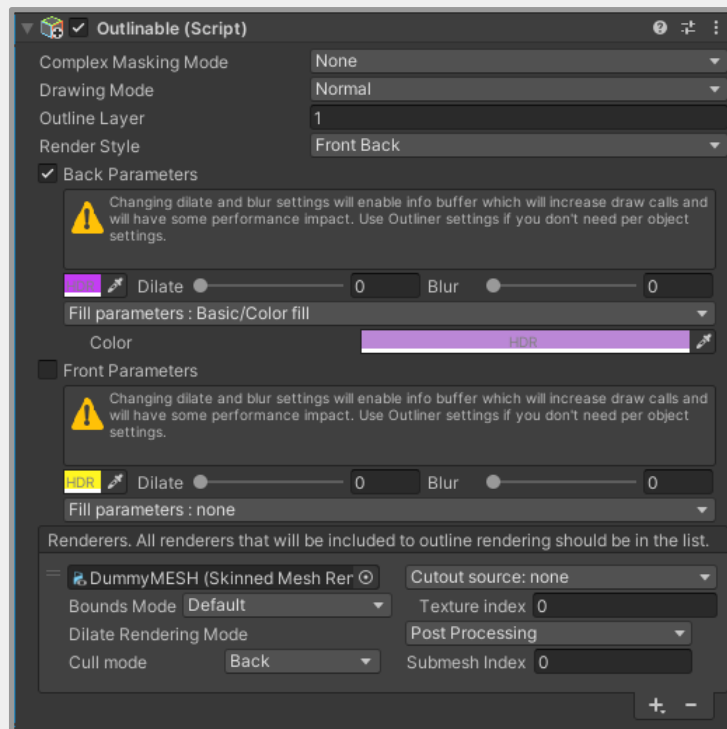
Le PlayerRoot doit posséder le template du FSM "FSM-Outline-PC".

Le Prefabs du Model des characters (*avec les FSM liés aux Statistiques*) doivent posséder un script Outlinable désactivé avec les paramètres ci-dessous et la couleur correspondante au PC.



*\*Paramètres de l'Outlinable d'un PC*

Le root du Model du character doit également posséder un composant Outlinable activé avec les paramètres ci-dessous et la couleur de remplissage correspondante au PC.



*\*Paramètres de l'Outlinable du remplissage du PC*

## Couleurs Outline des PCs

- [Eliot](#)

Outline : R:59 G:66 B:255 A:255

Remplissage : R:98 G:101 B:193 A:255

- [Eager](#)

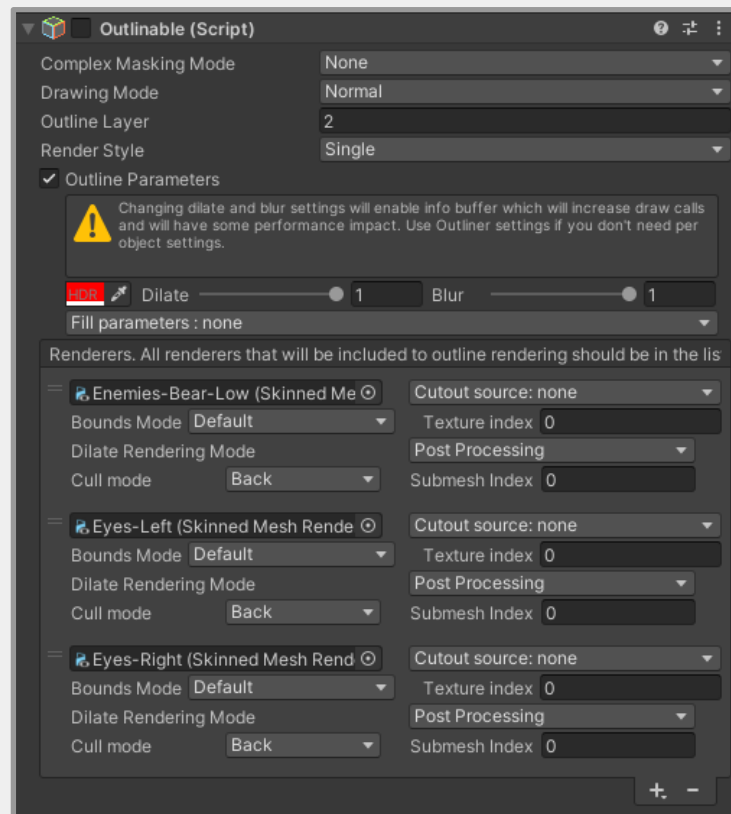
Outline : R:133 G:11 B:255 A:255

Remplissage : R:98 G:101 B:193 A:255

## Intégration Outline d'un PNJ

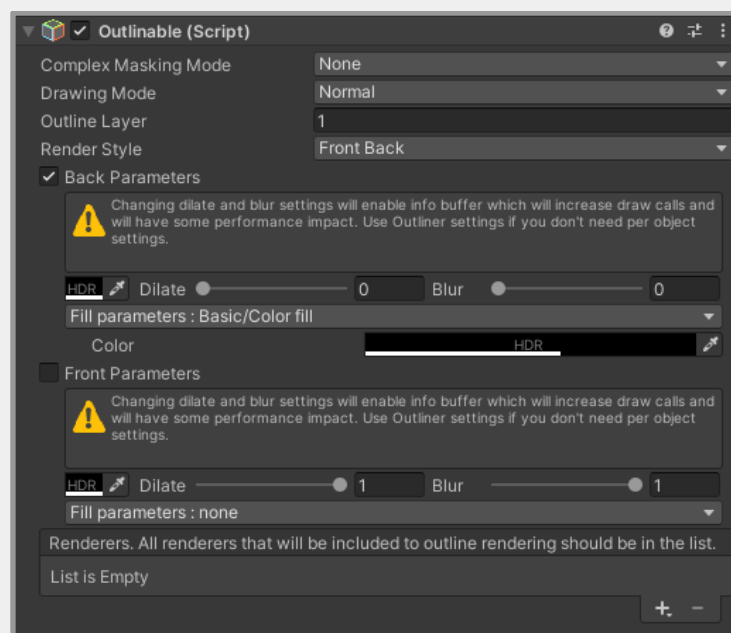
Le Root doit posséder le template du FSM "FSM-Outline-PNJ".

Le Root doit posséder un script Outlinable désactivé avec les paramètres ci-dessous et la couleur correspondante au PNJ.



*\*Paramètres de l'Outlinable d'un PNJ*

Le root du Model du PNJ (*Root-Graph*) doit également posséder un composant Outlinable activé avec les paramètres ci-dessous et la couleur de remplissage correspondante au PNJ.



*\*Paramètres de l'Outlinable du remplissage du PNJ*

## Couleurs Outline des PNJs

- **Ennemis**

Outline : R:255 G:0 B:0 A:255

Remplissage : R:0 G:0 B:0 A:75

- **Civil**

Outline : R: G: B: A:

Remplissage : R:0 G:0 B:0 A:75

## Intégration Outline d'un PROPS

Les PROPS sont séparés en 3 types :

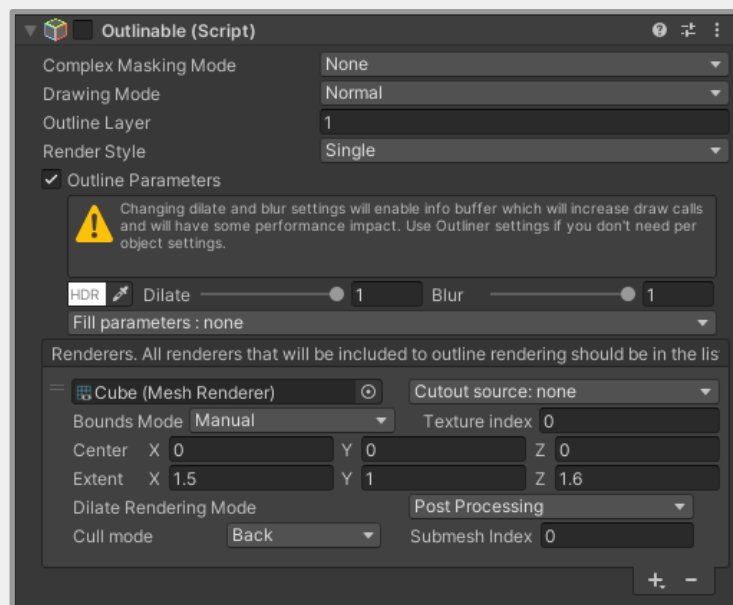
- Les PROPS classiques.
- Les PROPS nécessitant la touche d'interaction pour être utilisés.
- Les PROPS n'affichant pas leurs Outline en roll over (*ex : les bush*).

Le root des PROPS doit posséder le FSM nommé "FSM-Outline" avec le template "FSM-Outline-PROPS".

Ce Template possède 2 variables type bool exposées en input ("*needCtrl*" et "*activateOnRollOver*").

- La variable "needCtrl" est à cocher sur les PROPS pour lesquelles le joueur a besoin d'utiliser la touche d'interaction afin de l'utiliser (*ex : les pièges, les bennes, etc*). Elle va permettre d'afficher l'Outline en gris tant que le joueur n'utilise pas sa touche d'interaction pendant le roll over.
- La variable "activateOnRollOver" est à cocher sur les PROPS pour lesquelles l'outline n'est pas censée s'afficher en roll over (*ex : les buissons, etc*).

Le Root doit posséder un script Outlinable désactivé avec les paramètres ci-dessous. Il doit contenir les différents enfants possédant un mesh afin de fonctionner.

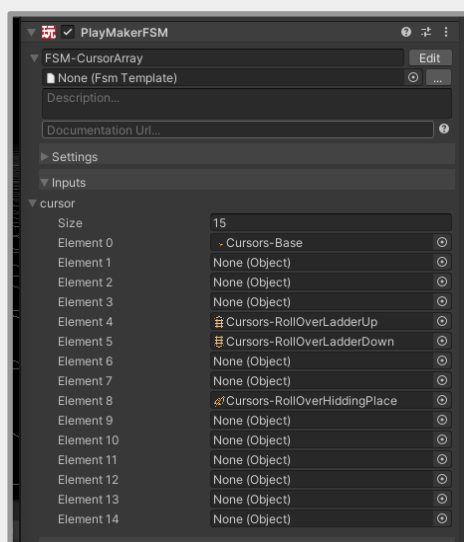


*\*Paramètres de l'Outlinable d'un PROPS cachette type "Buisson"*

L'Outlinable doit posséder la couleur blanche (soit R:255 G:255 B:255 A:255).

## 22. Curseurs

Chaque modèle des playable characters possède dans le FSM-CursorArray une Array qui leur est propre où sont mis les différents curseurs des actions qu'ils peuvent effectuer sauf celle des actions nécessitant la touche Ctrl.



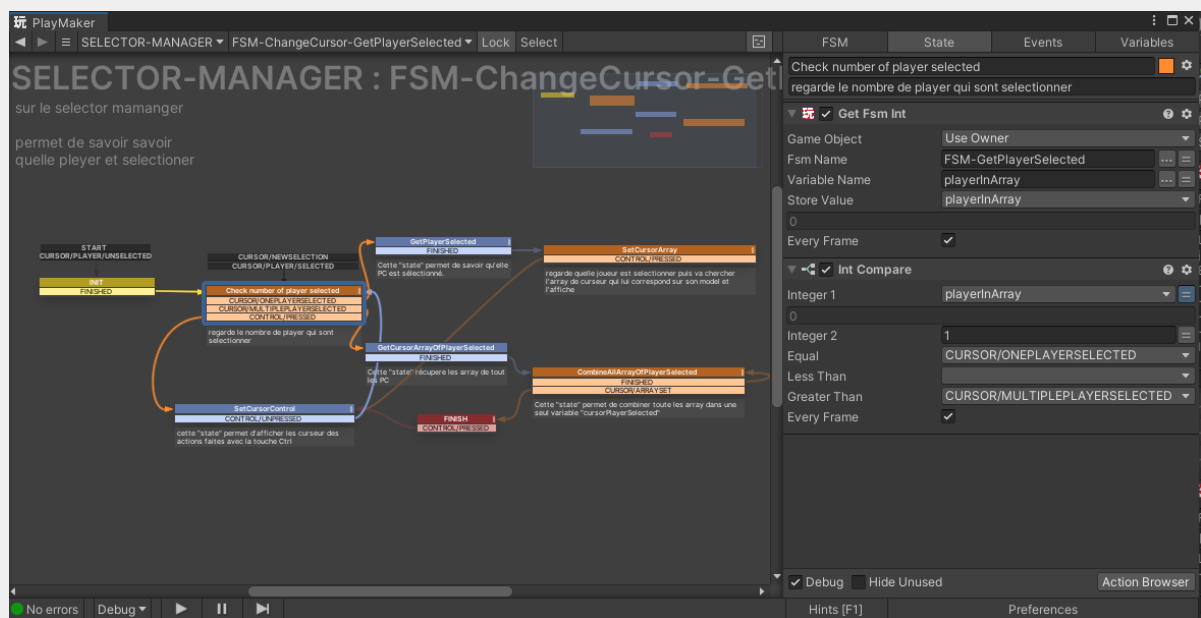
Pour le code dans un 1er temps on envoie l'array de curseur au player root.

Ensuite dans le FSM-SetCursor du PlayerRoot il reçoit 2 events venant du FSM-Selected qui sont CURSOR/PLAYER/SELECTED et CURSOR/PLAYER/UNSELECTED puis une fois l'un des events reçu plusieurs actions sont effectués :

- Prends l'array envoyé par le FSM-CursorArray et la met dans une nouvelle array appeler "arrayToGet".
- Puis envoyer des événements aux différents FSM se trouvant sur le SELECTORMANAGER → FSM-ChangeCursor-GetPlayerSelected, FSM-ChangeCursor, FSM-ChangeCursor-MousePick.

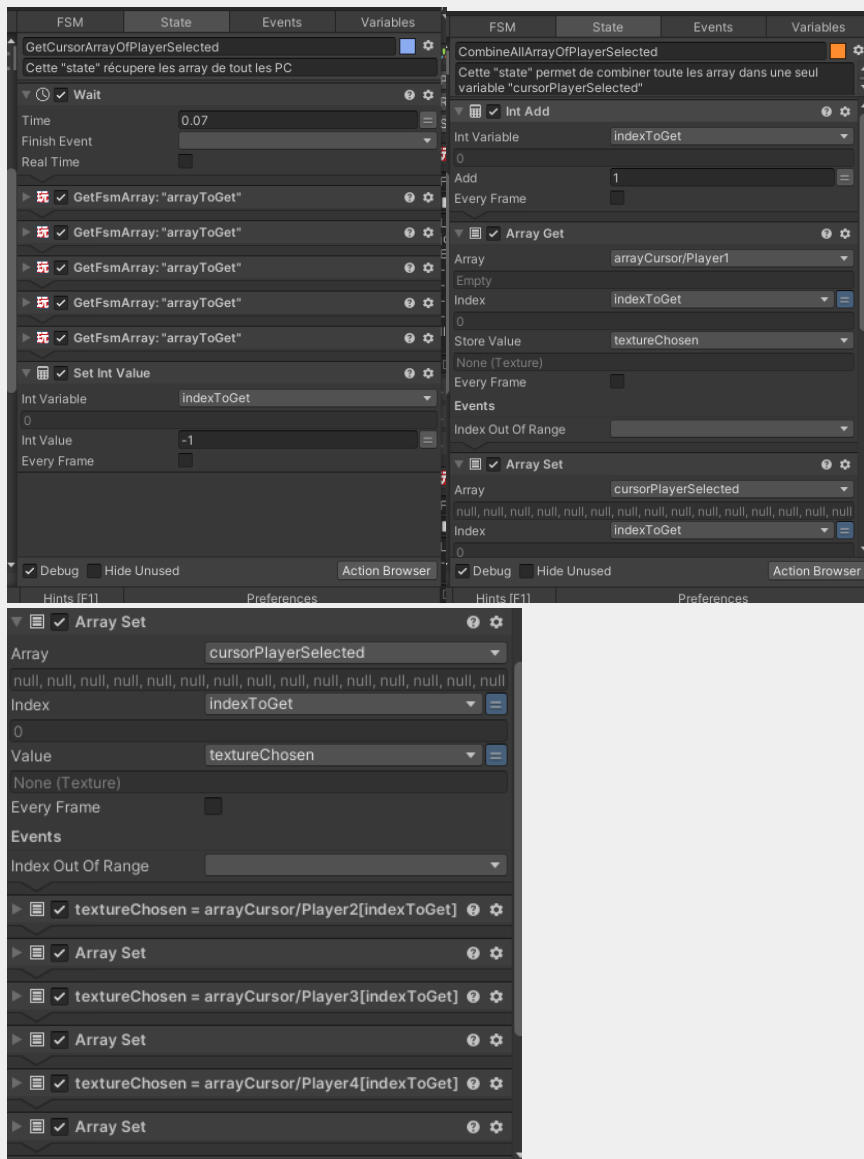
Le FSM-ChangeCursor-MousePick récupère l'objet sur laquelle le curseur se trouve et quand ce dernier change envoie un event à FSM-ChangeCursor.

Le FSM-ChangeCursor-GetPlayerSelected récupère l'information de nombre de joueurs sélectionnés et lesquels sont sélectionnés.



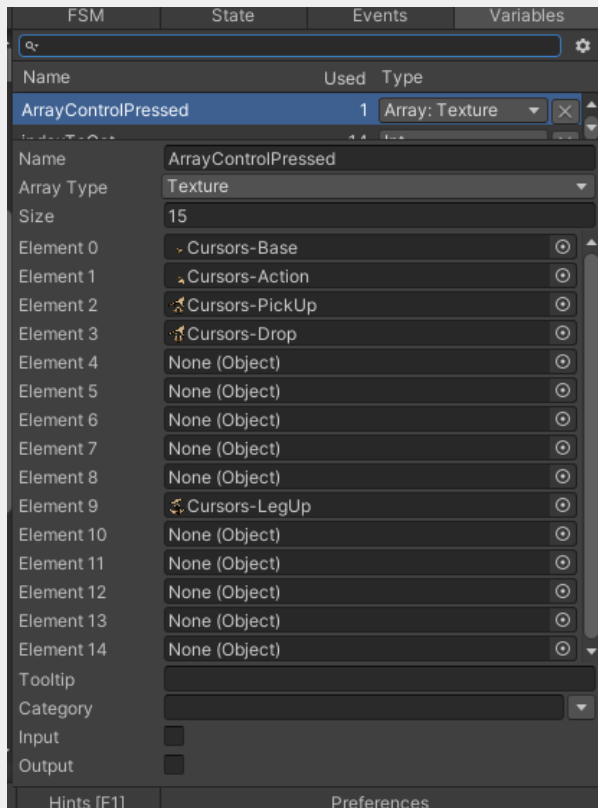
La branche du haut récupère le PC sélectionné et l'array qui lui correspond et l'envoie dans le FSM-ChangeCursor.

La branche du bas récupère tous les arrays des PC sélectionnés et les fusionnent.

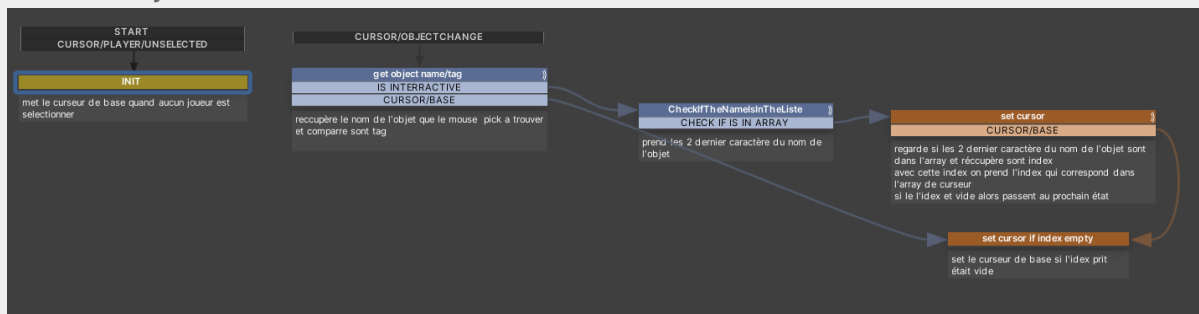


Une fois les arrays sont fusionnés, cette dernière est envoyée dans le FSM-ChangeCursor.

Lorsque le touche Ctrl est appuyé, il envoie au FSM-ChangeCursor le curseur des actions possibles avec la touche Ctrl.



Le FSM-ChangeCursor est le FSM ou le changement curseur s'effectue pour cela on compare le tag de l'objet récupéré par le FSM-ChangeCursor-MousePick, si le tag correspond on prend les 2 derniers caractères du nom de l'objet et on regarde si ces derniers sont contenus dans l'array "arrayPathObject", s'ils le sont on récupère l'index et se sert de ce dernier afin de prendre le curseur qui correspond dans l'array "cursorPlayerSelected".





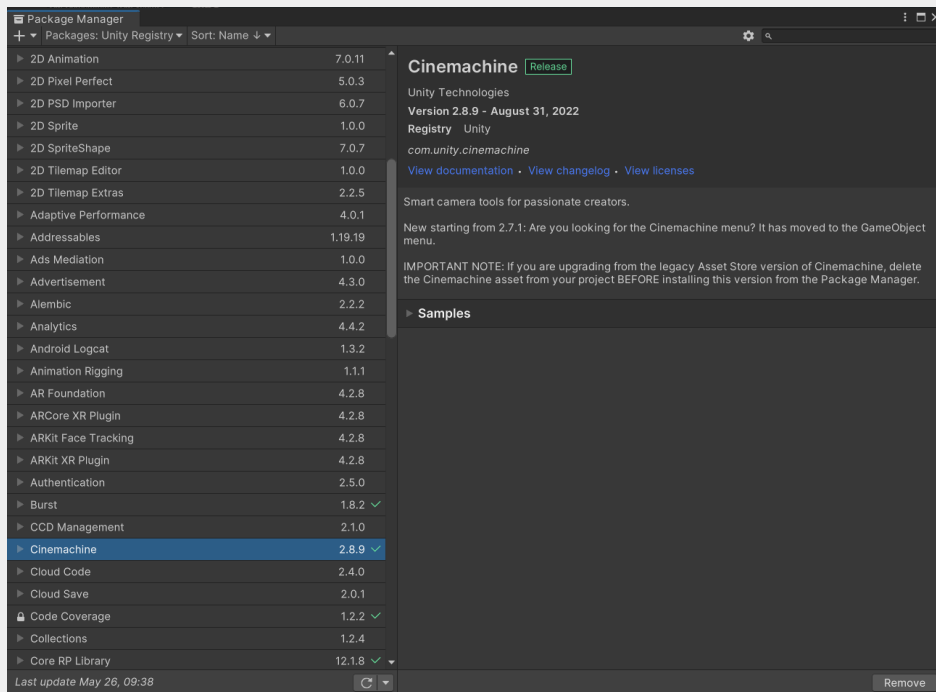
Dans la variable "arrayInteraction" les caractères correspondent aux noms des actions :

- 0. gr → ground / basic form
- 1. ac → action
- 2. pu → pick up
- 4. lu → ladder up
- 5. ld → ladder down
- 6. ju → jump
- 7. sn → sneak in
- 8. hi → hide
- 9. ss → short scale
- 10. lm → ladder when multiple players selected
- 11. ev → enemy's vision
- 12.     → tie enemy
- 13. tr → trap

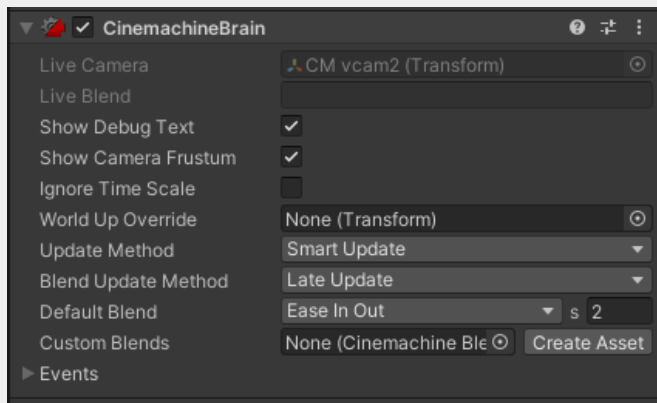
## 23. Cinemachine

### Intégration des éléments

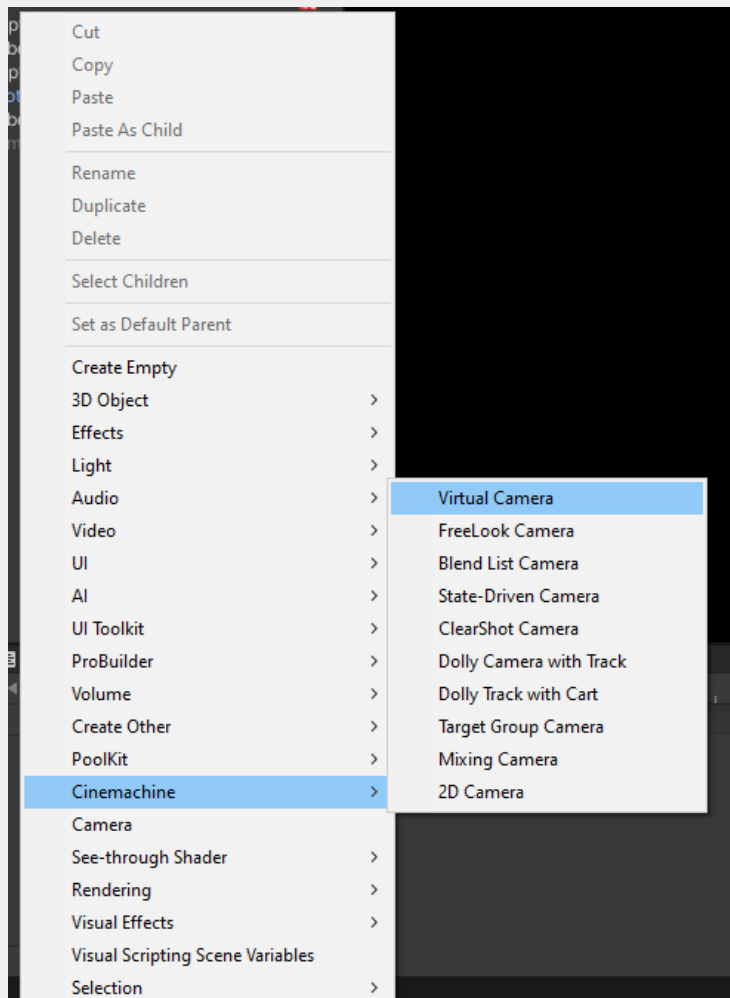
D'abord, installer le package cinemachine depuis le package manager (Unity Registry).



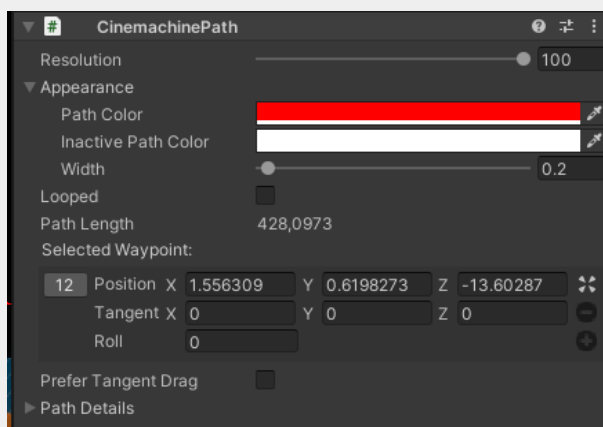
Afin de commencer il faut créer une nouvelle caméra puis on va ajouter un component CinemachineBrain sur la nouvelle caméra.



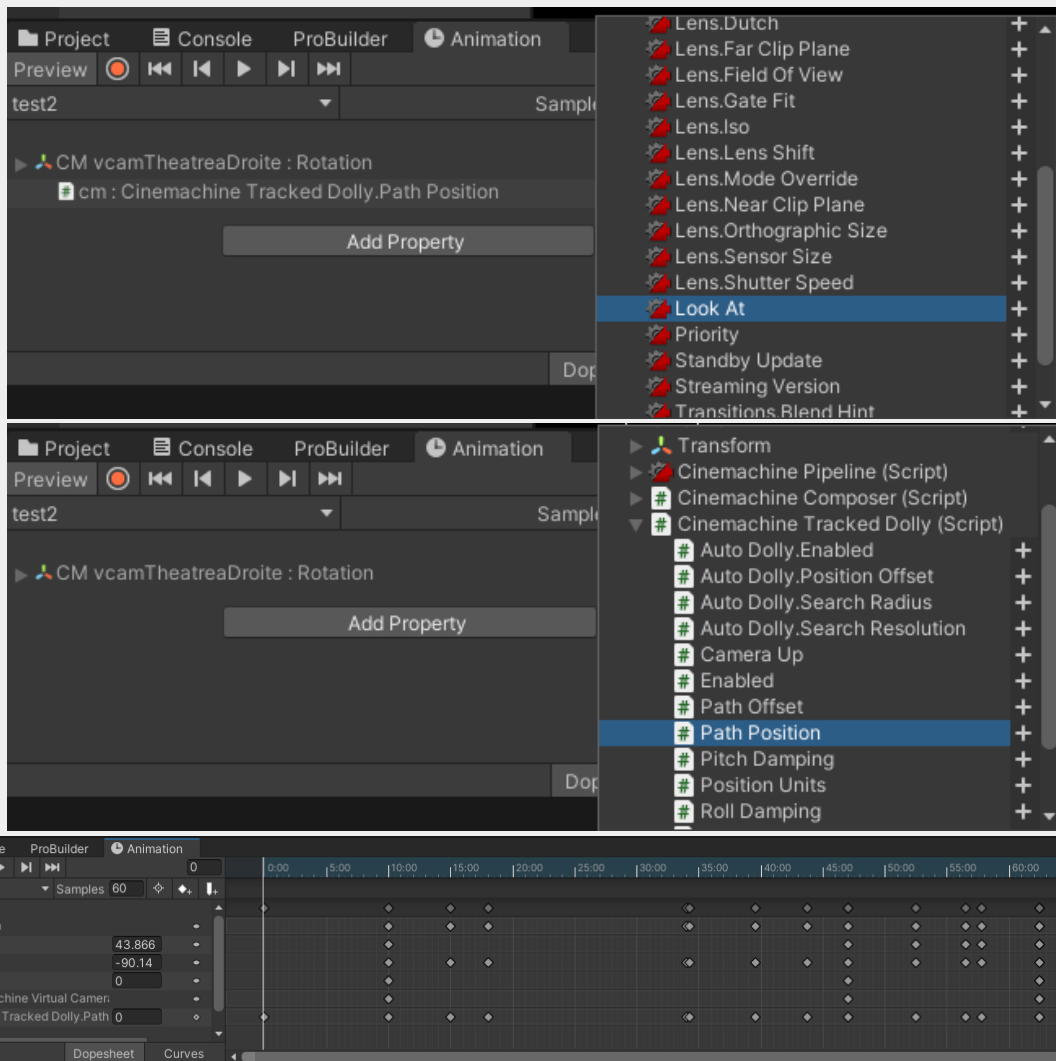
Pour créer les différents éléments de cinemachine il faut faire clique droit puis aller sur l'onglet cinemachine. Dans cet onglet on va créer une virtual camera qui va servir à contrôler notre caméra.



Pour créer le chemin que la caméra va suivre, on va créer un empty sur lequel on met un component CinemachinePath.

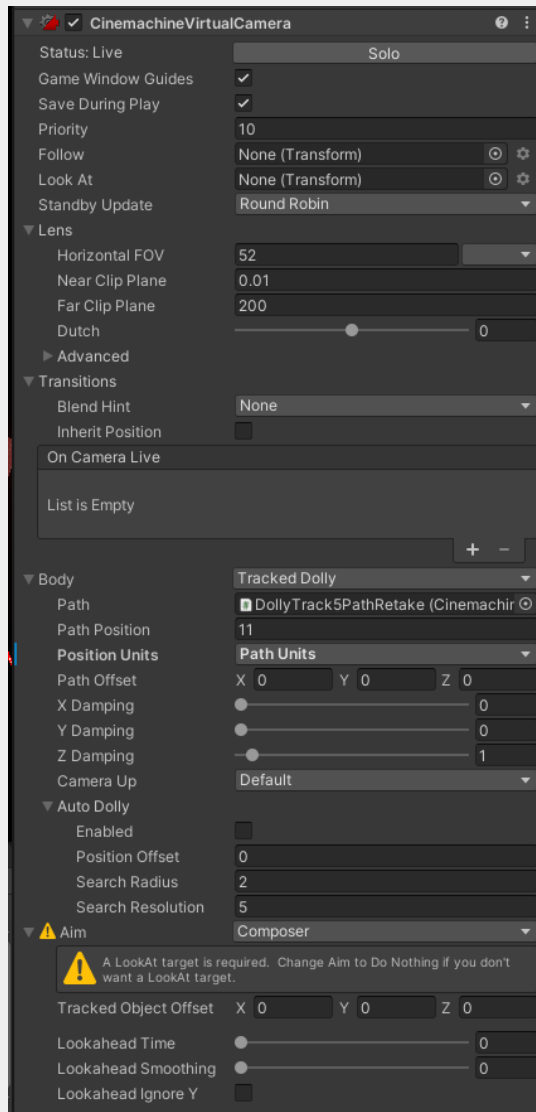


Une fois cela fait, créer une animation sur la virtual camera à l'aide de l'onglet animation. Dans la fenêtre animation cliquer sur Add property=>transform=>rotation afin de pouvoir rotate la caméra. Puis on ajoute les propriétés suivantes :



## Explication des différents éléments

### Virtual camera



**Priority** - Définie quelle Virtual Camera est affichée et si elles ont la même priorité, la dernière activée sera celle affichée.

**Follow** - Fait suivre un objet à la caméra sur un CinemachinePath.

**Look At** - Centre l'objet au centre de la vue et le suis dans ces mouvements.

**Lens** - Permet de changer le FOV, le clip near et clip far de la caméra.

**Body** - Utilisez les propriétés Body pour spécifier l'algorithme qui déplace la caméra virtuelle dans la scène. Pour faire pivoter la caméra, définissez les propriétés Aim.

Le Body possède plusieurs modes dans un menu dépliant :

**Do Nothing** : Ne déplace pas la caméra virtuelle.

**3rd Person follow** : Pivote la caméra horizontalement et verticalement autour du joueur.

**Framing Transposer** : Fixe la caméra mais cible tout de même l'objet concerné (ici, le Player).

**Hard Lock to Target** : Cet algorithme Virtual Camera Body utilise la même position sur la cible suivie. En d'autres termes, la cible agit comme un point de montage pour la caméra virtuelle.

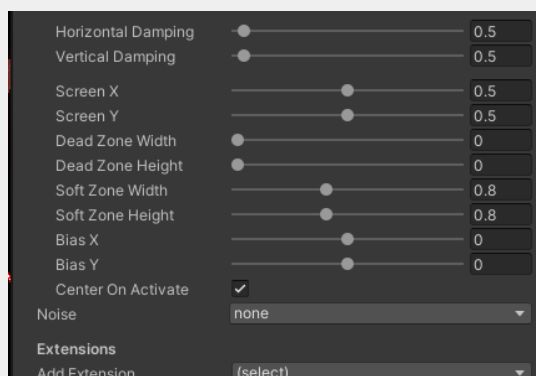
**Orbital Transposer** : Se déplace dans une relation variable à la cible suivie, acceptant éventuellement l'entrée du joueur.

**Tracked Dolly** : Se déplace le long d'une

trajectoire prédéfinie.

**Transposer** : Se déplace dans une relation fixe à la cible suivie.

**Path** : spécifie quel chemin la virtual camera doit emprunter.



**Path position** : Position de la caméra sur le chemin.

**Aim** : Utilisez les propriétés Aim pour spécifier comment faire pivoter la caméra virtuelle. Pour changer la position de la caméra, utilisez les propriétés Body.

*Do nothing* : Ne pas faire pivoter la Virtual camera.

*Composer* : Conserver la cible Look At dans le cadre de la caméra.

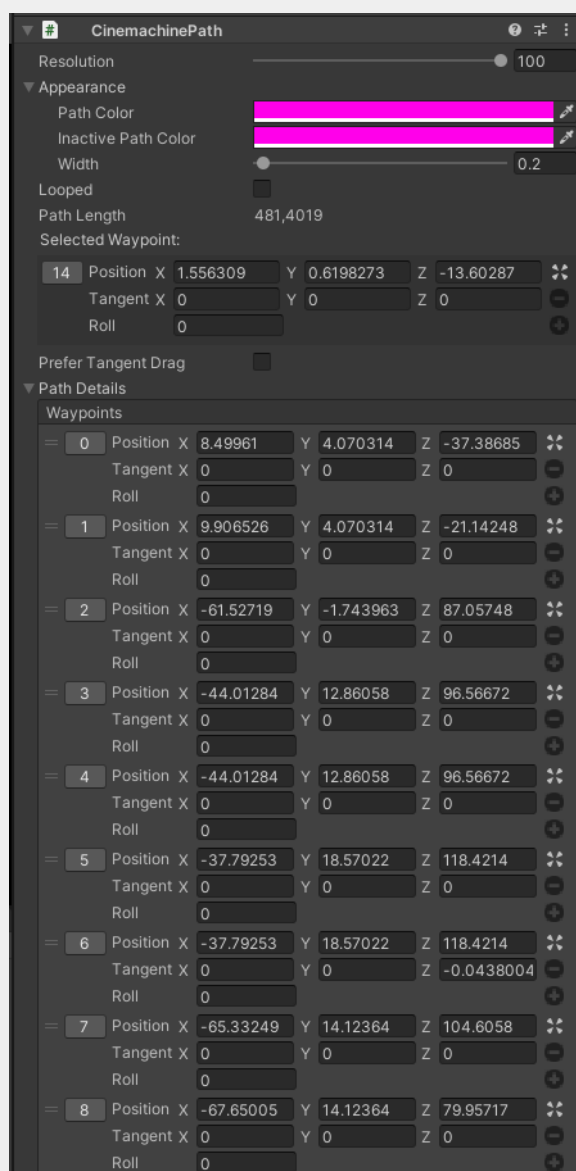
*Group Composer* : Gardez plusieurs cibles Look At dans le cadre de la caméra.

*Hard Look At* : Gardez la cible Look At au centre du cadre de la caméra.

*POV* : Tourner la caméra virtuelle en fonction de l'entrée de l'utilisateur.

*Same As Follow Target* : Réglez la rotation de la caméra sur la rotation de la cible Suivre.

## Cinemachine Path



**Résolution** : Cinemachine utilise cette valeur pour limiter la granularité lors du calcul des distances de parcours. Les hachures sur le chemin gizmo dans la vue scène reflètent cette valeur. Sert aussi à calculer la taille du chemin.

**Appearance** : change l'apparence du chemin lorsque ce dernier est sélectionné ou inactif ainsi que sa largeur, cela n'affecte que la vue scène.

**Looped** : fait boucler le chemin

**Path Length** : La longueur du chemin en unités de distance (read-only)

**Selected Waypoint** : montre le waypoint actuellement sélectionné

**Path details** : liste des waypoints qui composent le chemin

**Position** : position par rapport au transform du chemin

**Tangent** : Décalage par rapport à la position, qui définit la tangente de la courbe au waypoint. La longueur de la tangente code la force de la poignée du bézier. La même poignée est utilisée symétriquement des deux côtés du waypoint, pour assurer la fluidité.

## 24. PROPS

### Traps

Un PROPS "Trap" doit contenir au minimum 3 scripts FSM :

- FSM-Interactive (Template): permettant de détecter lorsque le joueur sélectionne le PROPS et interagit avec. Il sert de FSM de détection. Il doit être nommé "FSM-Interactive".
- FSM-Outline-Trap (Template): permettant de mettre en valeur l'objet en roll over.
- FSM-....: permettant au piège de s'activer (tombée de caisse, tombée de panneau, etc..)

Les Mesh de l'objet 3D doivent contenir le Shader-Highlight ainsi que le script Template "FSM-Highlight" (cf. *Chap-Highlight*) afin que l'objet soit mis en évidence dans le décor.

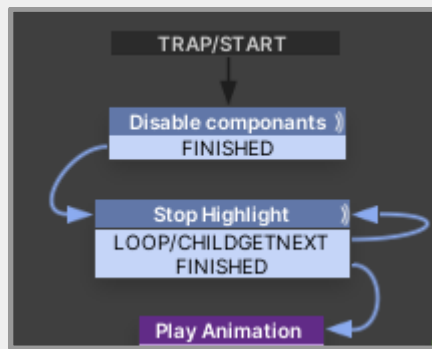
#### **FSM-Interactive**

Le template "FSM-Interactive" possède un variable en input qu'il faut remplir par le nom du FSM présent sur le FSM-Root servant à effectuer l'action commandée. Ici, il faut la remplir du texte "FSM-Trap" présent sur le PlayerRoot (S'il change de nom, il faut changer le texte également afin qu'il fasse le lien avec le FSM présent sur le PlayerRoot).

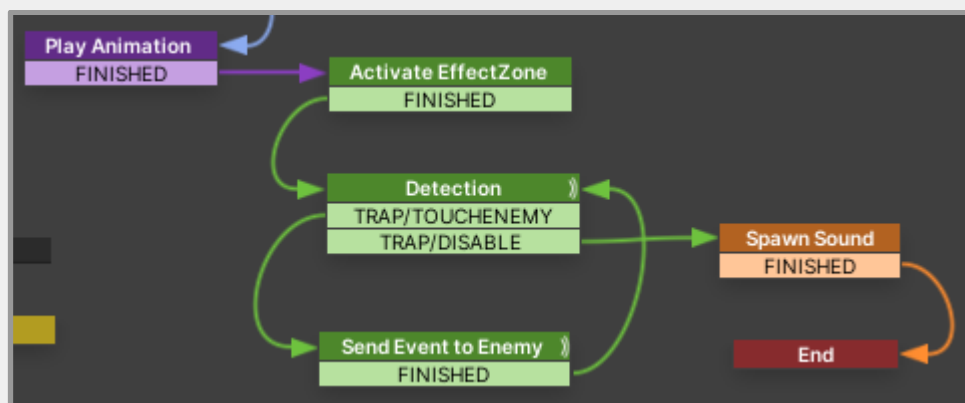
#### **Guide FSM à coder**

Le seul FSM à coder dans ce gameobject est celui de l'animation ainsi que des effets du PROPS "Trap". Ce FSM doit contenir l'Event Global "TRAP/START".

Il vous faut ensuite désactiver les différents éléments de reconnaissance de l'objet : l'Outline, le Highlight, ainsi que le collider d'activation du PROPS "Trap".



Ensuite il faut jouer l'animation du piège suivi de l'activation (bien timé) du collider de l'hitbox du piège.



Enfin, il reste à créer le son émis grâce à la GLOBAL-POOL et au prefab "MovementNoise". L'instance du son créé doit être scale avec la bonne valeur (variable nommée "noiseScale") qui est exposée en input du fsm (même valeur pour x et z).

## Items

*Se rapporter au chapitre Inventaire pour connaître l'emplacement de ces derniers, ainsi que l'utilisation de leurs FSM.*

Il existe actuellement **5 items** différents :

- **Ammo** :  
Sert à recharger entièrement tous types d'armes à feu.
- **Bottle** :  
Sert à distraire un ennemi dû au son causé lorsqu'elle est lancée au sol ou sur un obstacle.
- **CuttingPliers** :  
Sert à couper une grille afin de créer un passage.
- **Handcuffs** :

Sert à menotter les ennemis K.O. afin qu'ils soient immobilisés et inoffensifs.

→ **Matchbox** :

Sert à allumer un feu sur certains types d'éléments interactifs.

## 25. Model 3D

### 25.1. Personnages

Name	Nbr Polys	Texture	Material	Name/Artist
Eliott Seanness	10000	2048	Lit/Full	Théo MP
Malone B. Clonan	10000	2048	Lit/Full	Clément B
Frank E. Eager	10000	2048	Lit/Full	Ambre
Wolf	10000	2048	Lit/Full	Juliette M
Vulture	8000	2048	Lit/Full	Nicolas G
Bear	9000	2048	Lit/Full	Enola

#### Personnages Principaux Gentils

Eliot Seanness



Malone B. Clonan

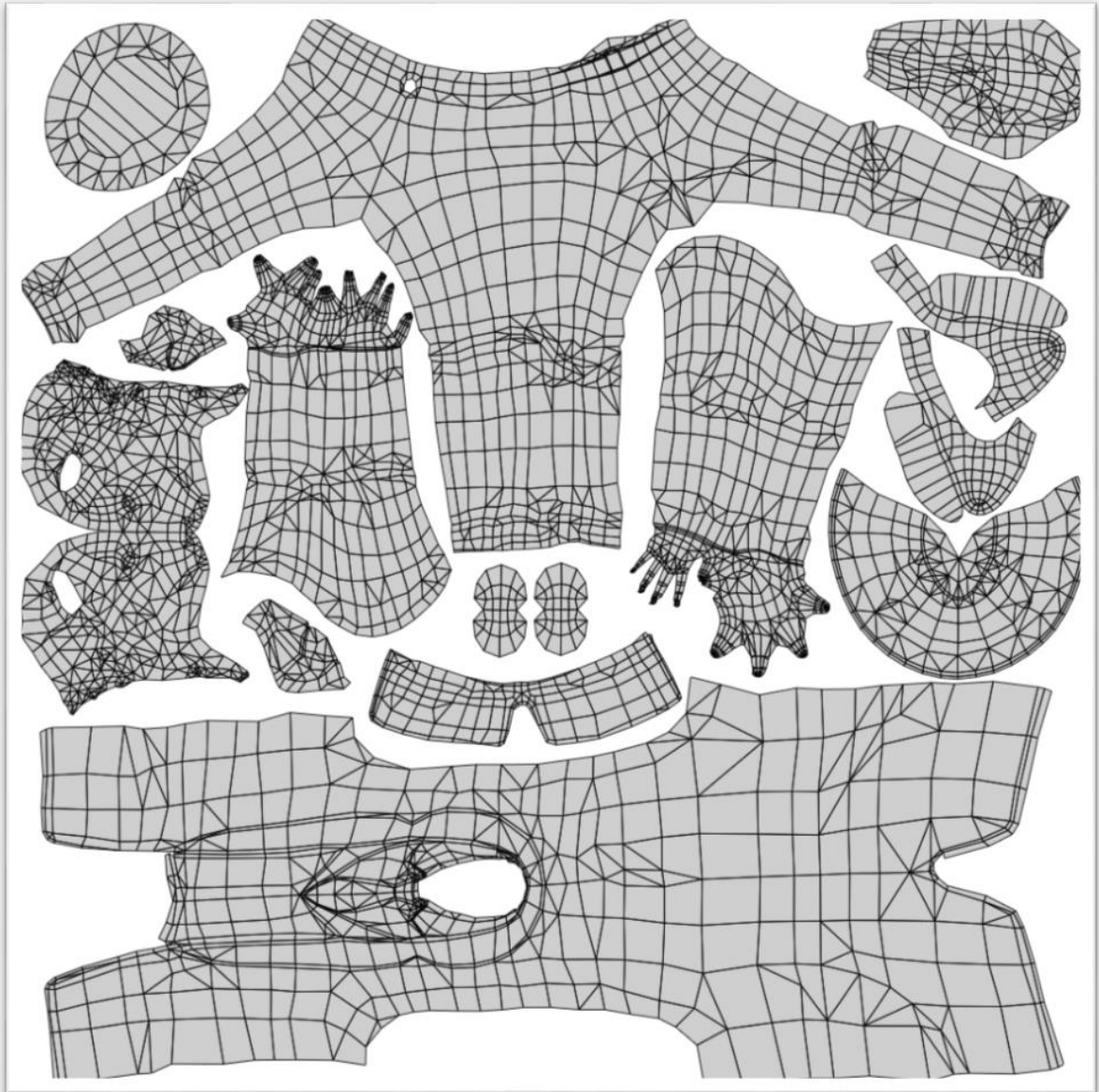




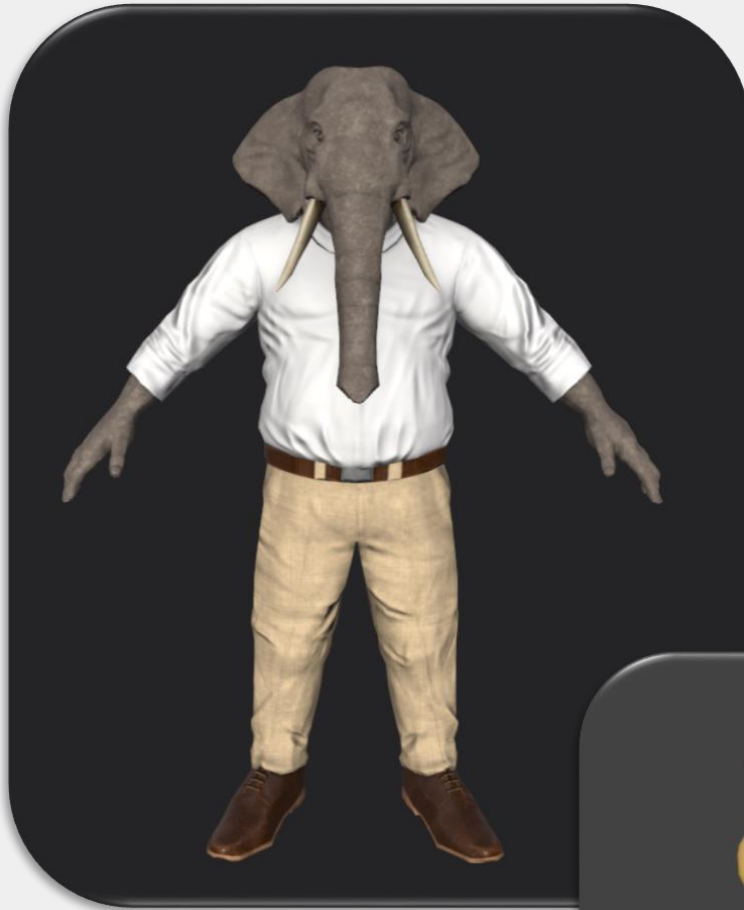
Frank E. Eager



UV's



Texture



## Ennemis

Etape 1 : Blender



Concernant les personnages, le modèle ci-dessus a été choisi comme base commune de modélisation. Un Low Poly du corps a été effectué puis modifié selon les différentes espèces animales à produire.

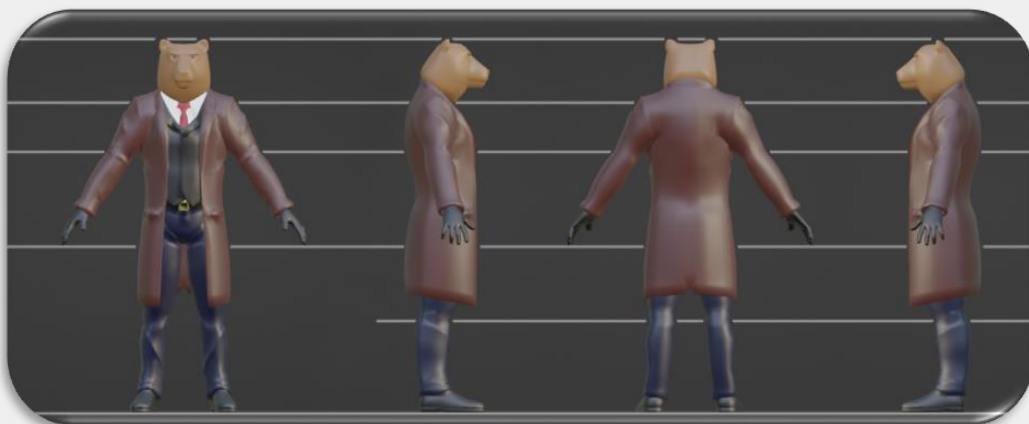
Vulture



Wolf

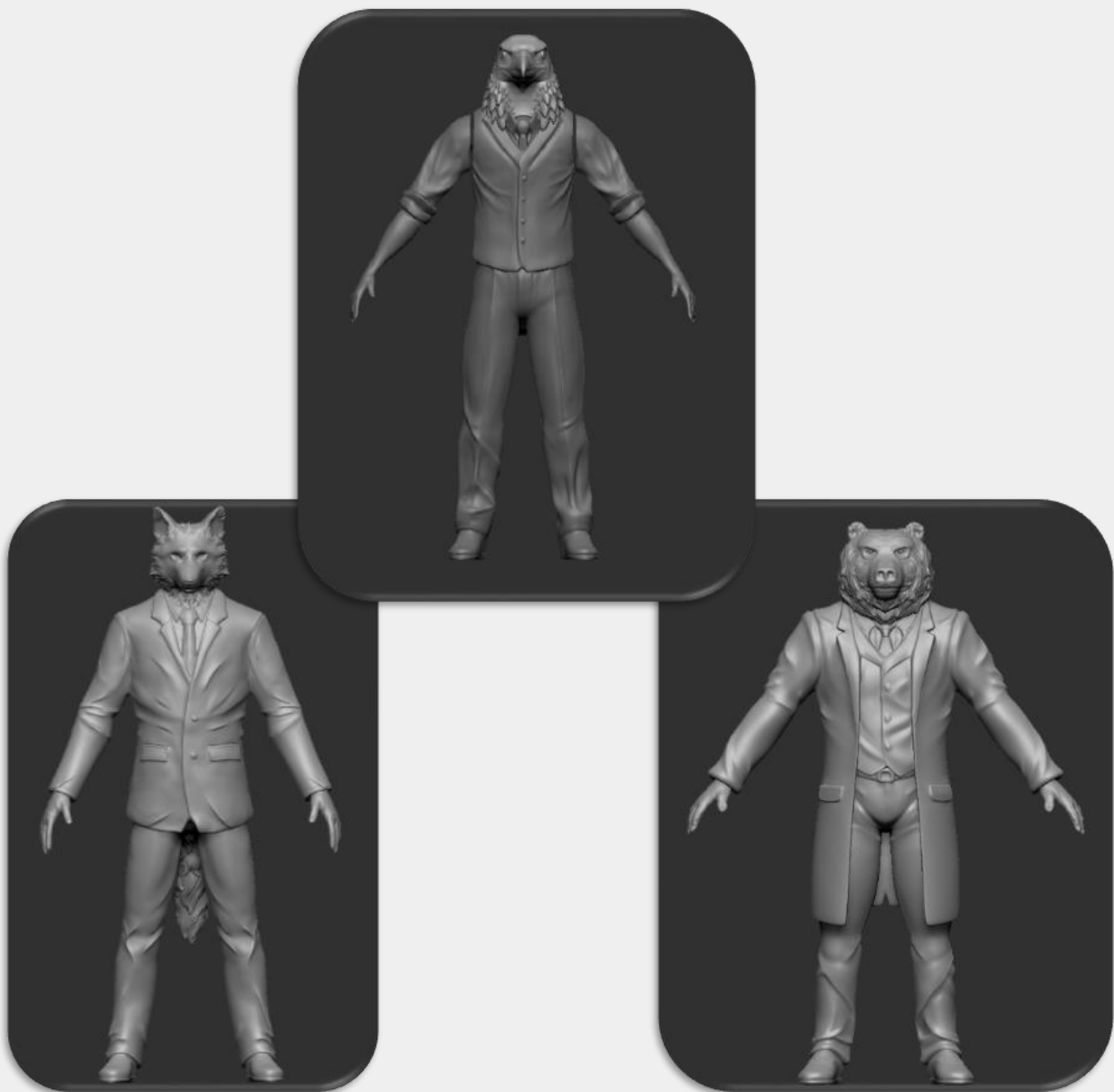


Bear



Après une recherche de références, une base de tête et de vêtements ont été ajoutés à la modélisation. Et pour finir, le modèle a été complété avec des détails comme les plis des vêtements.

Etape 2 : Zbrush



Les différents modèles ont été importés sur Zbrush pour y ajouter encore plus de détails et donc confectionner un High Poly.

### Etape 3 : Low Poly



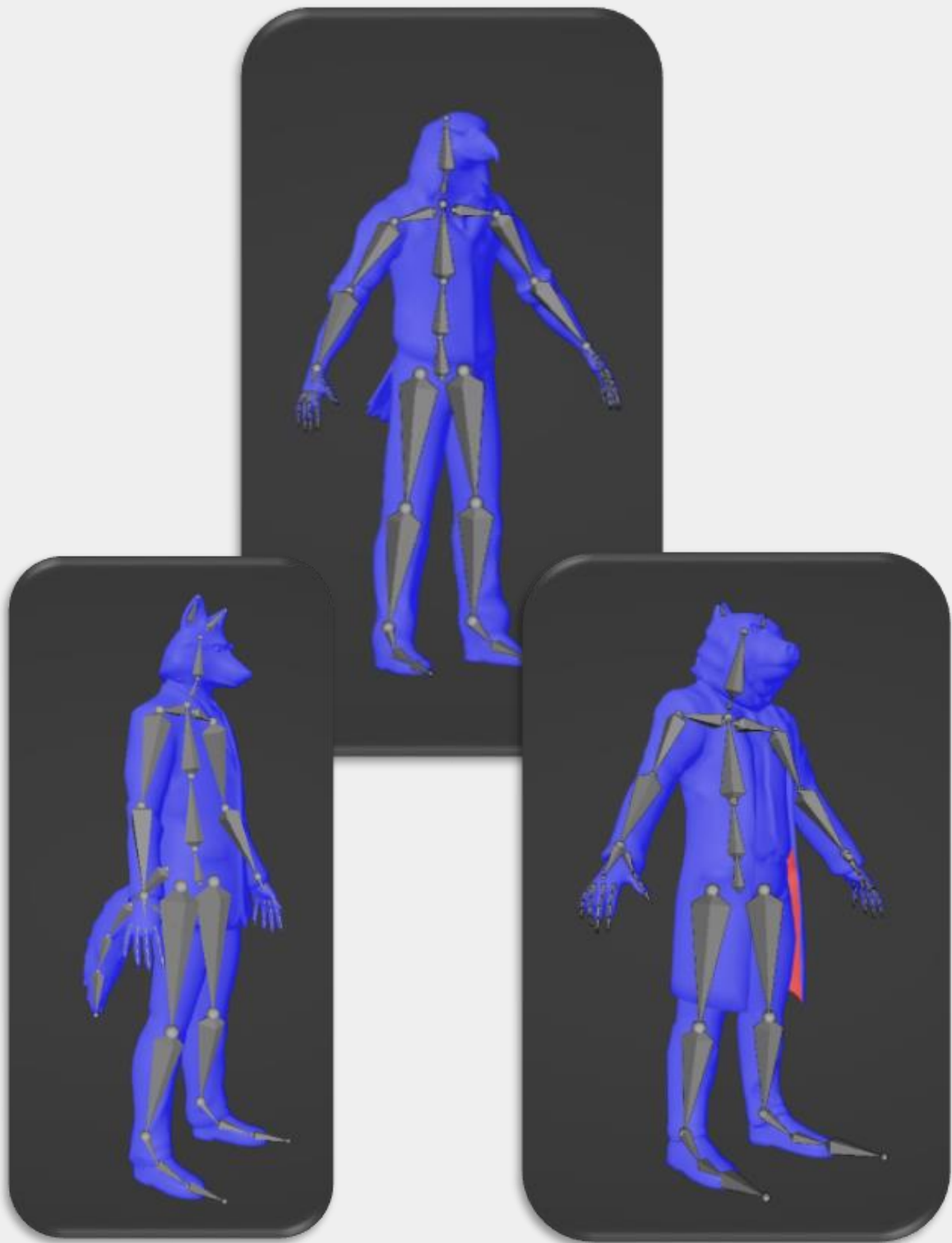
Une fois le High Poly terminé, un Low Poly a été modélisé par dessus celui-ci.



Etape 4 : Texture



Etape 5 : Rigging/skinning



## 25.2. Props

Name	Texture	Material	Name/Artist
Vehicles	1024	Lit/Full+ Emissive	Victor, Quentin, Bastien, Kiryan
SpecificTransports	1024	Lit/Full	Juliette, Nicolas.P, Thomas, Victor
HidingPlace	1024	Lit/Full	Nicolas.P, Nicolas G, Juliette, Théa, Kiryan
Trap	1024	Lit/Full	Kiryan, Nicolas P
AccessPath	1024	Lit/Full	Kiryan, Nicolas P, Nicolas G, Thomas
Universals	1024	Lit/Full	Victor, Juliette, Nicolas P, Nicolas G

### Specifics transports

### 25.3. OBJ

Name	Texture	Material	Name/Artist
SpecificTransports	1024	Lit/Full + Emissive	Thomas, Théo D, Quentin
UrbanElements	2048	Lit/Full	Nicolas.P, Thomas, Thea, Kiryan, Quentin, Théo.D, Victor, Enola
IndustrialElements	1024	Lit/Full	Kiryan, Nicolas.P, Thomas, Quentin, Juliette
ConstructionElements	1024	Lit/Full	Kiryan, Nicolas P, Nicolas G,
Universals	1024	Lit/Full	Nicolas P, Nicolas G, Kiryan, Thomas, Théo D, Clément

#### Urban Elements



## 25.4. Levels

Name	Texture	Material	Name/Artist
Floor	2048	Lit/Full	Bastien, Nicolas.P, Théo D, Clément, Thomas
Building	1024	Lit/Full	Kiryan, Thomas, Ambre, Victor, Enola, Théo D, Clément
Structure		Lit/Full	Théo D, Clément
SpecificsElements	Varié	Lit/Full	Théo M, Clément, Théo D, Victor
Obstacles	256	Lit/Full	Thomas
Decals	/	HDRP/decal	Bastien, Juliette M, Enola

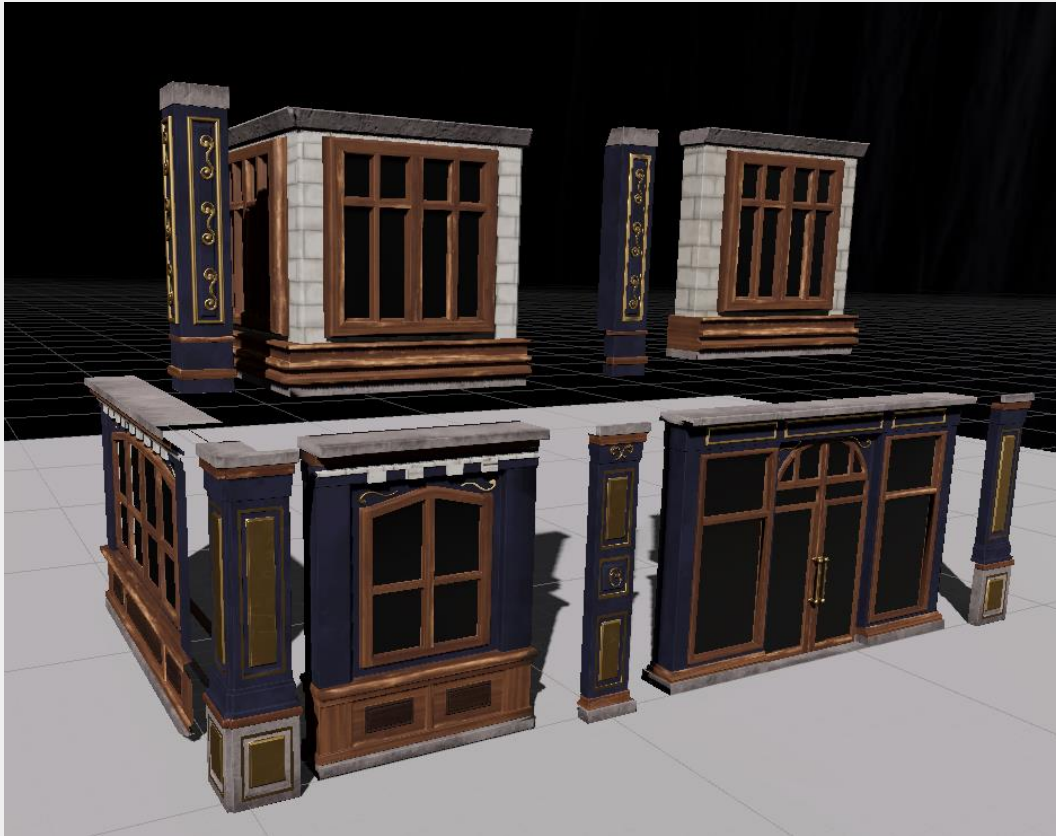
### Building:

Épaisseur des murs porteurs : 0.3m

Épaisseur cloison: 0.1m

Taille jouable building : -3m, 0m, 3m, 6m, 9m

## Modulaire



Chaque module est unique et vient se connecter parfaitement avec les autres. Pour cela on utilise un processus particulier.

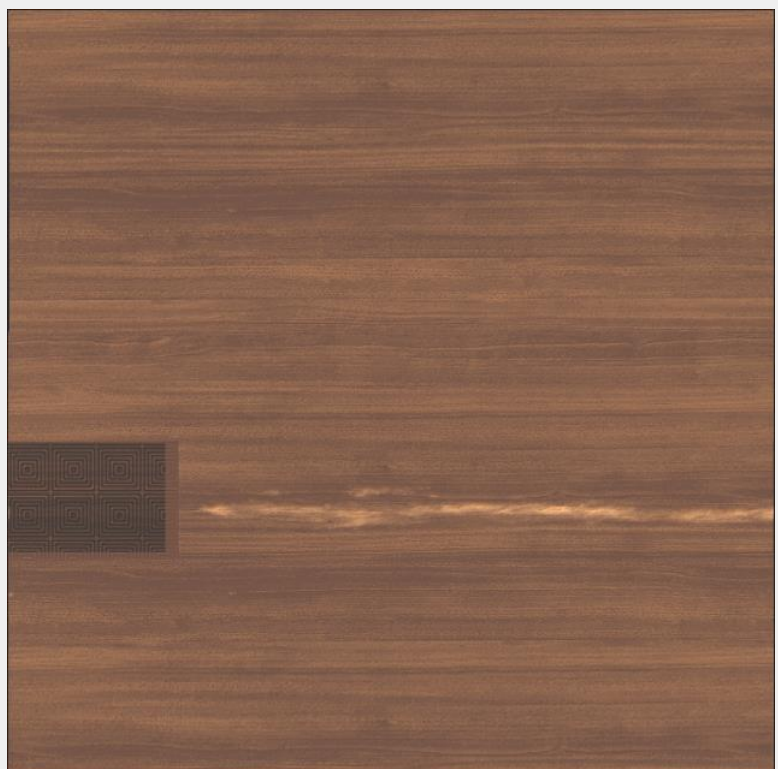


## Texture

On utilise 1 à 3 planches de textures environ pour toute une série de modules afin d'économiser un maximum d'espace. Ces textures sont également réutilisables pour d'autres modules du jeu si besoin.



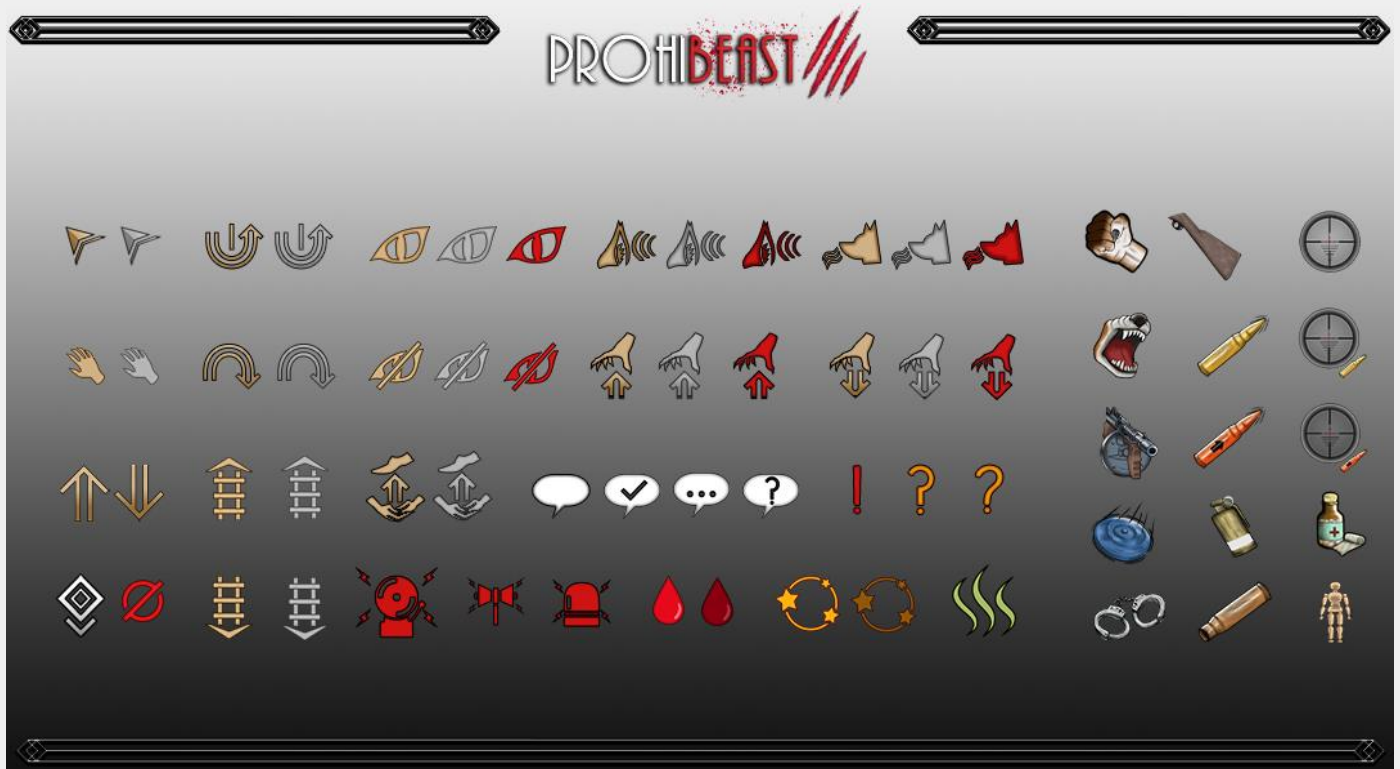
Chaque texture tiles de gauche à droite et de haut en bas.



## 25.5. GUI

Name	Texture	Material	Name/Artist
Icons	256x128 128x128	HideUIIcôneAtlas SkillIndic-Atlas	Ambre, Bastien; Théo.D
Cursors	64x64	/	Théo.D, Bastien
HUD	2048x1024 4096x2048 4096x2048 1024x1024 2048x2048 4096x2048	InGameUIAtlas Atlas-MissionFailed Atlas-MainMenu Atlas-Inventory Atlas-Alarm Atlas-PauseMenu	Théo.D, Bastien





## 26. Jeu Similaire

### 26.1. Desperados III :



## 26.2. Empire of sin :

