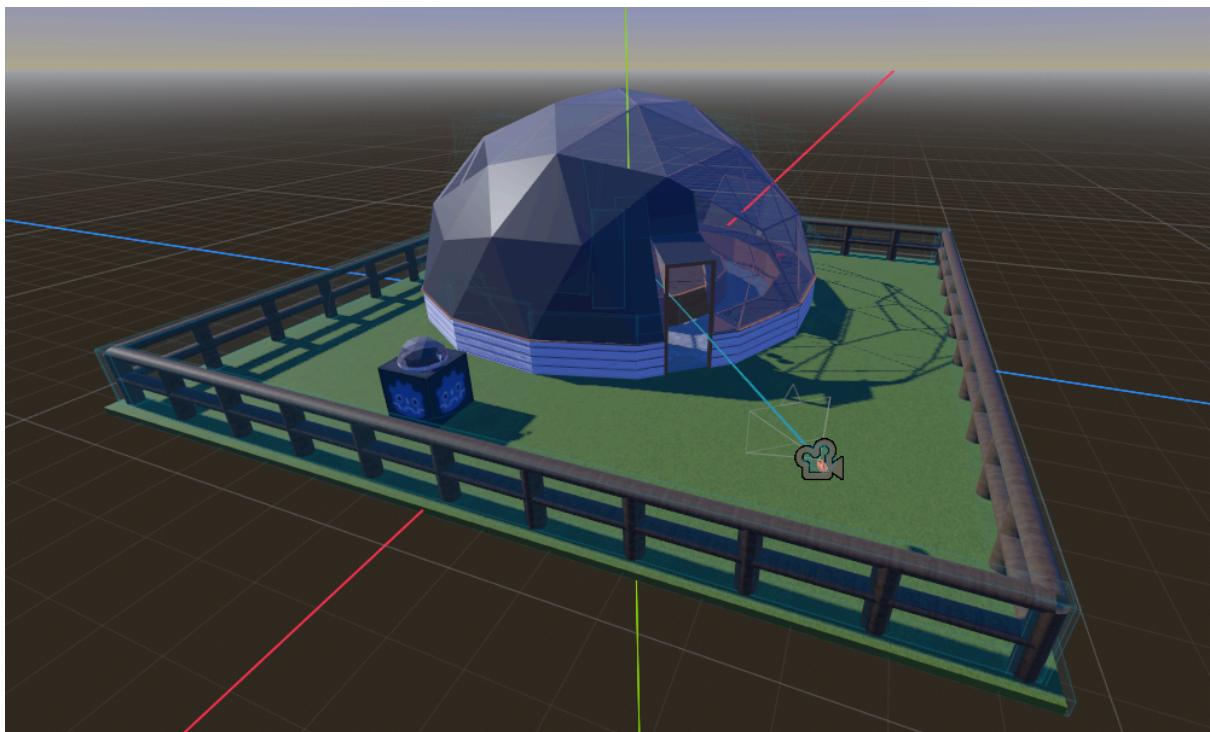


Documentation Partie Godot du Projet

Serre et Usine VR



1. Notes Préliminaires.....	3
2. Applications Nécessaires.....	3
3. Vocabulaire de base.....	4
4. L'Interface de Godot.....	7
5. Arborescence du Projet.....	12
6. Projet.....	14
6.1. Objets de base.....	14
6.2 Interactions.....	16
6.3 Menus.....	17
6.4 Pickable objects.....	19
6.5 Animations.....	20
6.6 Particules.....	24

1. Notes Préliminaires

Comme n'importe quel logiciel de game design, Godot n'est pas un logiciel simple à prendre en main. Ainsi si vous cherchez à apprendre comme faire un jeu sur Godot en utilisant cette documentation, nous vous conseillons de regarder diverses vidéos sur Youtube à côté car cette documentation peut s'avérer insuffisante sur certains points.

Le projet a été réalisé avec *Godot 4.4.1* sur *Windows 10*.

Afin d'éviter de faire une documentation infinie pour ce qui est de la description du projet, nous allons nous attarder sur les concepts théoriques et les subtilités des méthodes utilisées plutôt que de lister l'entièreté de ce qui a été fait dans le code et le projet. En même temps que vous lisez cette documentation, il est conseillé d'avoir le projet ouvert sur Godot sur un écran à côté afin de pouvoir regarder les codes et objets dont nous parlerons dans cette documentation.

Nous préciserons quels codes sont concernés par telle partie lorsque nous en parlerons.

2. Applications Nécessaires

- **Godot 4.0+** (<https://godotengine.org/download/windows/>)

3. Vocabulaire de base

Dans Godot, tout fonctionne avec trois concepts de base : les **scènes**, les **nœuds** et les **scripts**. Il est nécessaire de bien comprendre la distinction entre les trois avant d'aller plus loin dans cette documentation.

Une **scène** est la base d'un monde. Sans scène, il n'y a rien dans Godot. Une bonne pratique de Godot est que, lorsque l'on souhaite créer un objet (joueur, menu, monde...), la première chose à faire est de créer une nouvelle scène.

En sauvegardant une scène, on obtient un fichier .tscn ou .scn, la distinction est expliquée dans le guide d'importation de Blender à Godot.

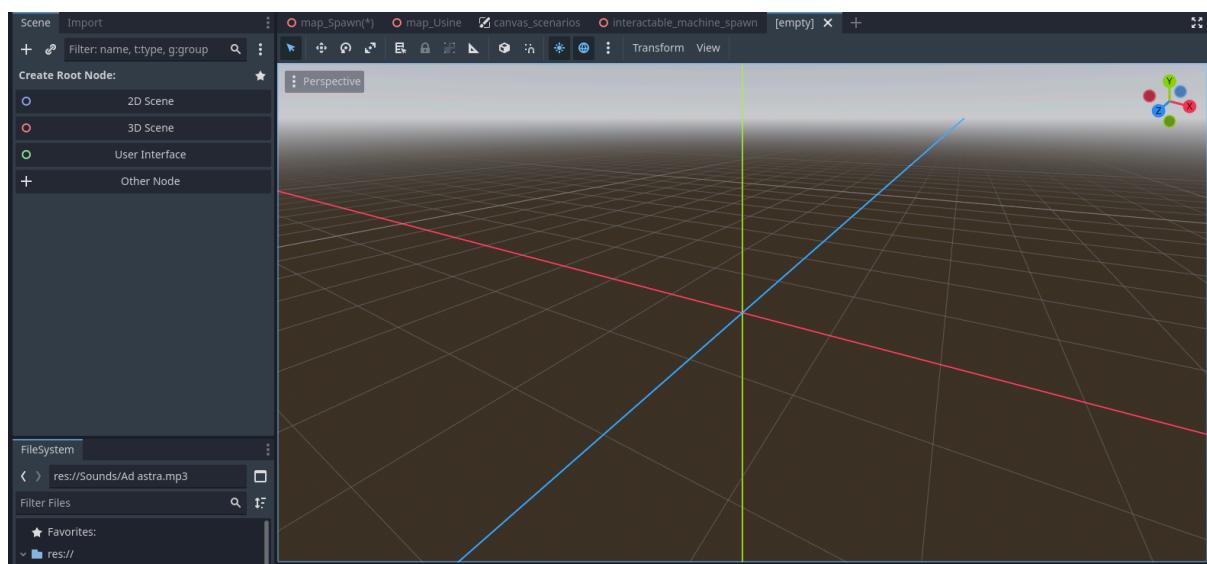


Figure 1 : Exemple de Scène vide

Chaque scène est composée d'un ou plusieurs **nœuds**, il existe des dizaines et des dizaines de nœuds différents sur Godot, chacun ayant des paramètres et sous paramètres dépendant du type de nœud choisis. Il y a par exemple le noeud "MeshInstance3D" qui permet de créer une "mesh" (un objet), le noeud "CollisionShape3D" qui permet de créer une zone de collision, le noeud "Label3D" qui permet d'écrire du texte en 3D...

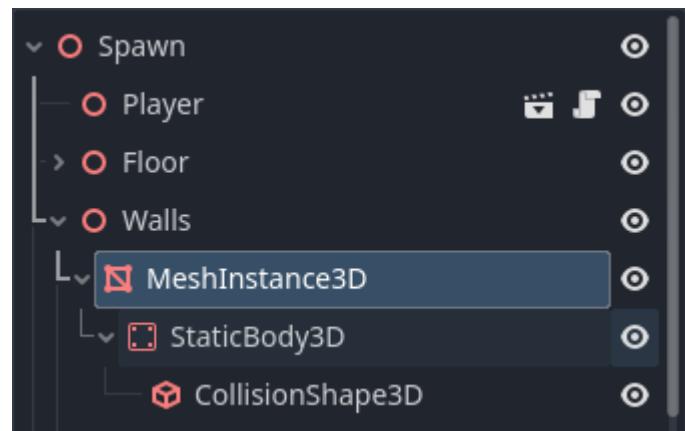


Figure 2 : Ensemble de noeuds

Dans l'image ci-dessus, on peut voir un noeud “Node3D” Spawn qui contient un noeud “Node3D” Walls qui contient un noeud “MeshInstance3D” qui contient un noeud “StaticBody3D” qui contient un noeud “CollisionShape3D” (ces 3 derniers noeuds sont la structure de base d'un objet ne bougeant pas avec des collisions).

Enfin, il est possible d'attacher un **script** à chacun de ces nœuds. Un script est un fichier code que l'on attache à un nœud en particulier, il s'appelle ainsi en raison du langage utilisé par Godot : le GodotScript. C'est pourquoi en enregistrant un script, on obtient un .gd .

```

1  extends MeshInstance3D
2
3  @onready var screen : XRToolsViewport2DIn3D = $screen_explanation_dehuller/Viewport2DIn3D
4  @onready var area : XRToolsInteractableArea = $XRToolsInteractableArea
5  @onready var camera : Camera3D = get_viewport().get_camera_3d()
6
7
8  func _ready() -> void :
9    >> screen.visible = false
10
11
12  func interact() -> void:
13    >> screen.visible = true
14
15
16  func _process(_delta) :
17    >> if screen.visible == true :
18      >> >> var obj_pos = area.global_transform.origin
19      >> >> var cam_pos = camera.global_transform.origin
20
21      >> >> var cam_dir = (cam_pos - obj_pos).normalized()
22      >> >> var cam_yaw = atan2(cam_dir.x, cam_dir.z)
23
24      >> >> var orbit_radius := 1
25      >> >> var screen_angle = cam_yaw + PI * 0.5
26      >> >> var offset = Vector3(sin(screen_angle), 0, cos(screen_angle)) * orbit_radius
27      >> >> var screen_pos = obj_pos + offset + Vector3(0.0, 0.0, 2.0)

```

Figure 3 : Exemple de script

Les fonctions de base d'un script sont : `_ready()` et `_process()`.

`_ready()` est une fonction qui est appelée une fois que l'objet attaché au script a fini de charger et `_process()` est une fonction qui est appelée à chaque image dans le jeu (chaque frame).

4. L'Interface de Godot

Dans cette partie nous allons expliquer comment fonctionne Godot et comment est répartie l'interface lorsque l'on arrive sur le logiciel.

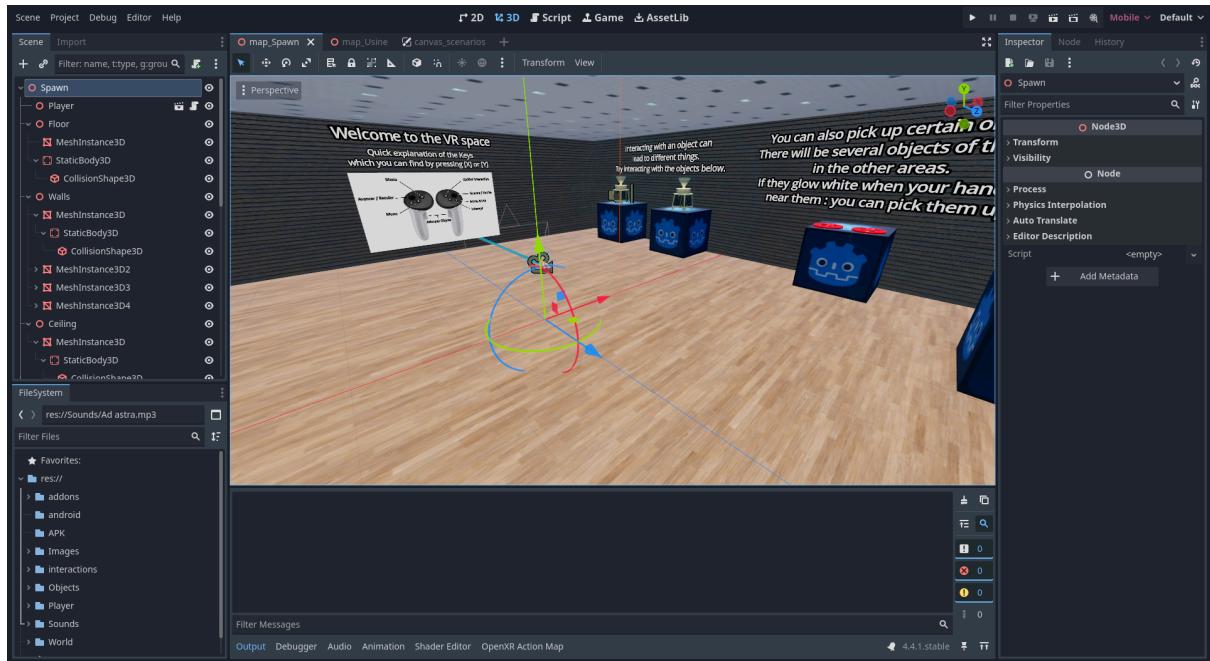


Figure 4 : Interface de Godot

L'interface est séparée en plusieurs parties distinctes : en haut à gauche, on peut retrouver le menu usuel de chaque logiciel avec différents paramètres afin de toucher aux paramètres liés aux scènes, projets ou à l'éditeur. C'est dans cette partie que vous exporterez également le projet afin d'en faire un apk.

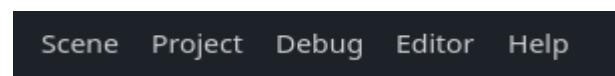


Figure 5 : Menu logiciel usuel

En haut au milieu, on peut retrouver le menu permettant de changer ce qu'affiche la fenêtre centrale de Godot. 2D est utilisé pour la construction de l'Interface Utilisateur et de menus, 3D pour l'ensemble de l'espace que vous allez construire, script pour la partie code, game pour quand vous testez le projet et AssetLib pour les library.



Figure 6 : Menu d'affichage

En haut à droite, il y a entre-autre le bouton pour lancer le jeu, pauser le jeu, charger une scène en particulier et changer la méthode de render de Godot.



Figure 7 : Menu de jeu

A gauche de l'écran, il y a tout d'abord le menu "Scène", c'est dans celui-ci que vous ajouterez des nœuds (avec le bouton +) et des scripts (avec le bouton script+) à votre scène.

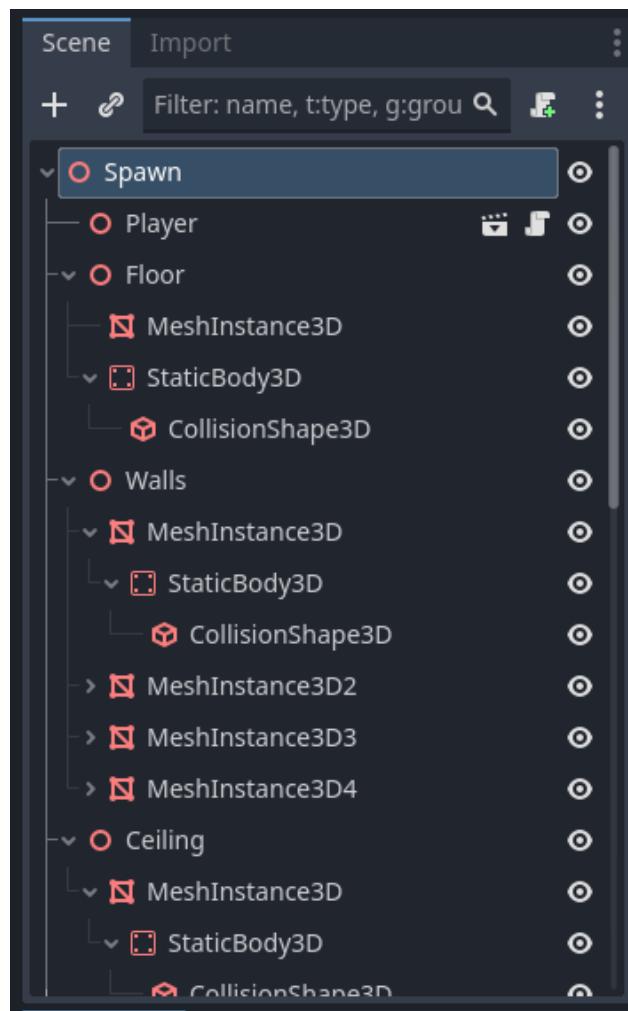


Figure 8 : Menu Scene

En bas à gauche, il y a le menu "FileSystem" qui représente l'arborescence de votre projet (tous les fichiers et dossiers du projet), il fonctionne comme un explorateur de fichier usuel.

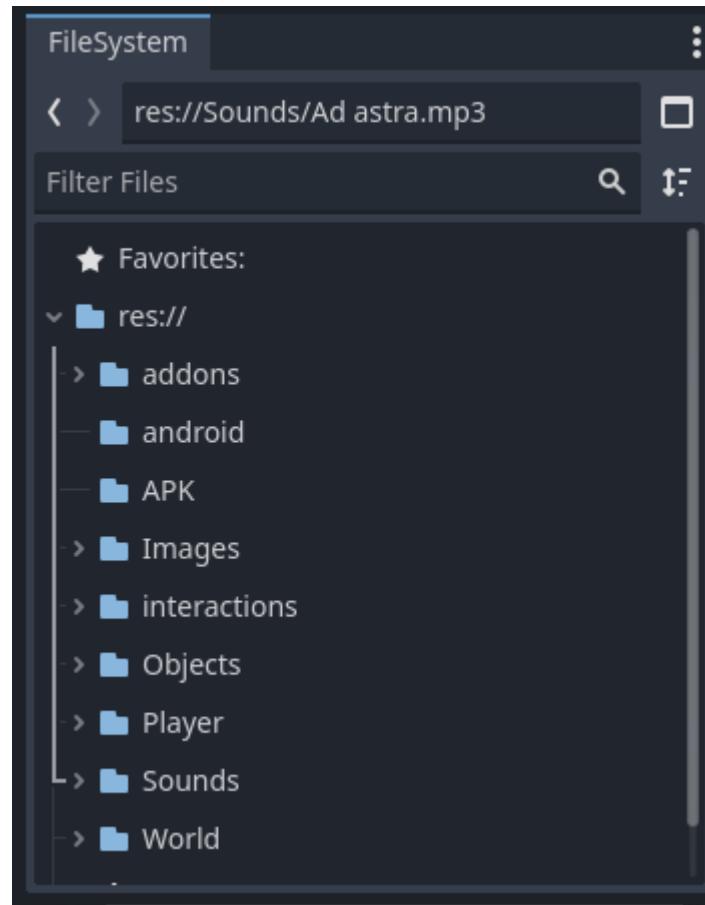


Figure 9 : Menu FileSystem

Au milieu de votre écran il y a la fenêtre de visualisation du projet. c'est ici que vous pouvez créer de nouvelles scènes (avec le bouton +) et changer de scène en cliquant sur la scène ouverte qui vous intéresse.

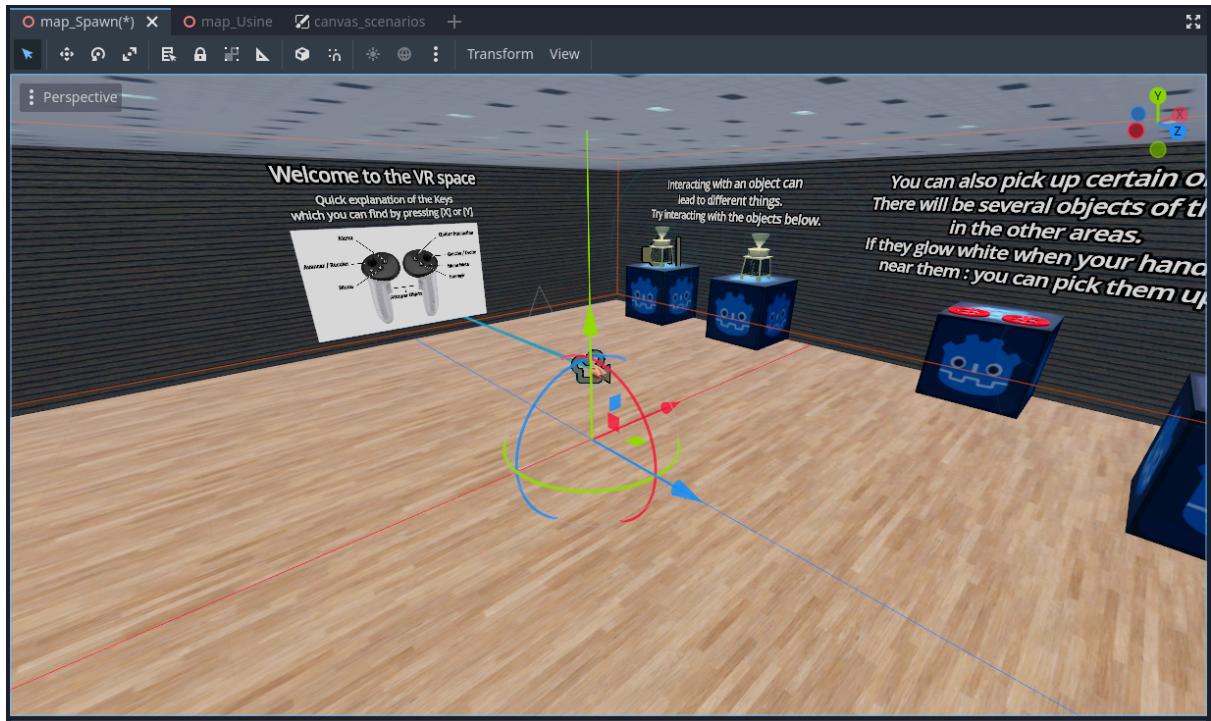


Figure 10 : Fenêtre de Visualisation

En bas de l'écran, il y a le menu d'Output et de Débogage, ainsi que d'autres menus que vous utiliserez moins. Si jamais vous utilisez un “print” dans l'un de vos scripts, le message sera affiché ici.

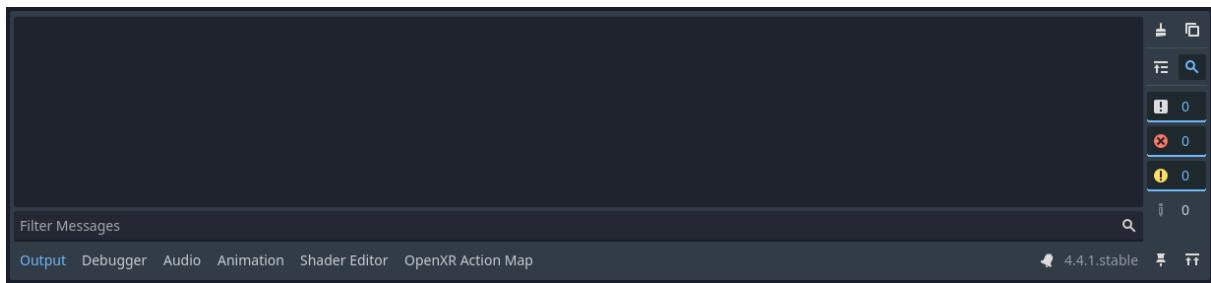


Figure 11 : Menu divers

Enfin, à droite de votre écran, il y a l'inspecteur, c'est là que vous verrez les différentes propriétés, variables et autres dont disposent les différents nœuds que vous créez, vous passerez probablement une grande partie de votre temps dans ce menu à modifier des valeurs ou autre.

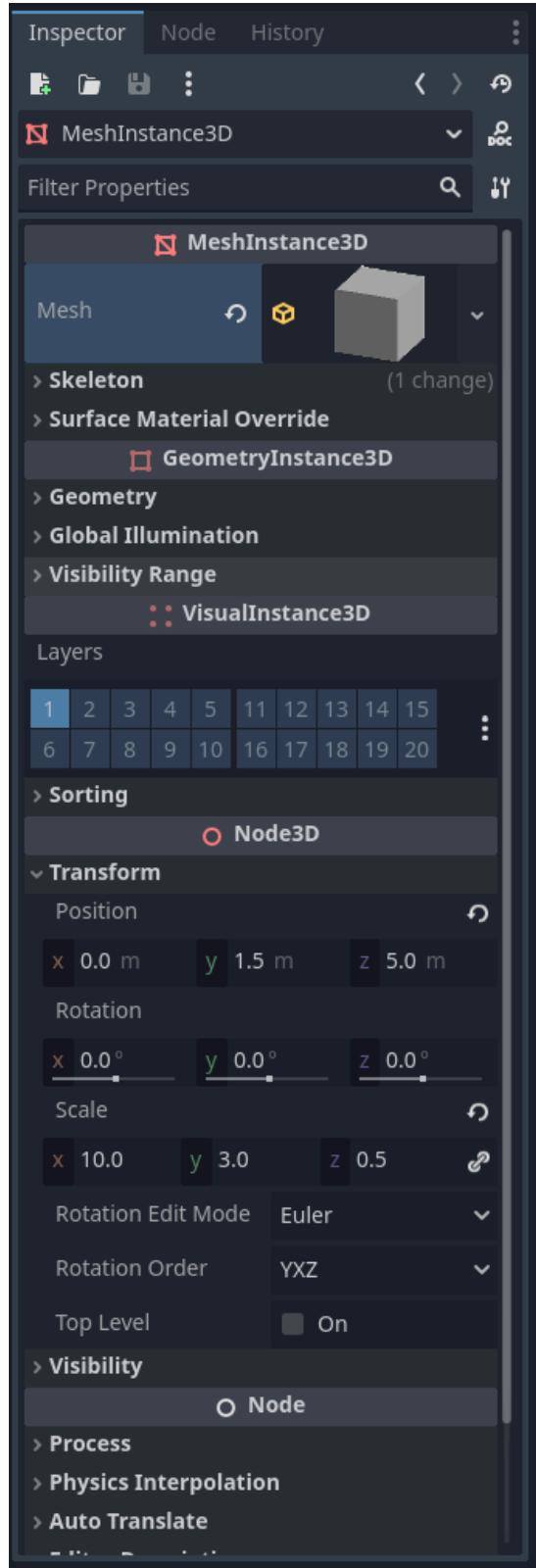


Figure 12 : Menu Inspector

5. Arborescence du Projet

Afin de bien commencer, expliquons tout d'abord comment nous avons séparé les fichiers de notre projet. Commençons par les dossiers automatiques que nous n'avons pas créé à la main :

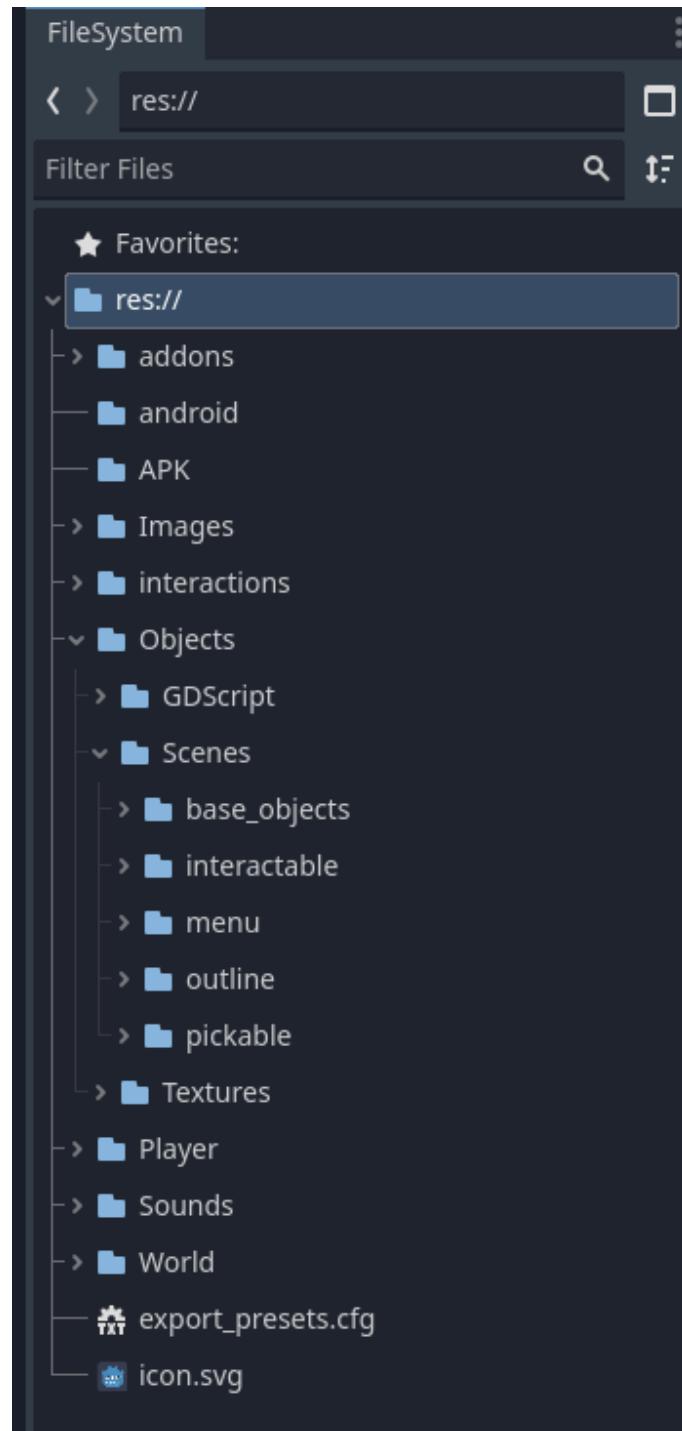


Figure 13 : Arborescence du projet

- le dossier “addons” contient les fichiers de chaque library ajouté au projet de base via l’onglet “AssetLib” en haut de l’écran, c’est un fichier automatique qui contient certains scripts qui vous seront utiles par la suite.

- les dossiers android et APK contiennent des informations nécessaires à l’exportation du projet en .apk pour être utilisé par la suite sur les casques VR Meta Quest 3, nous n’y avons jamais touché. De même, le fichier export_presets.cfg est lié à ces dossiers et nous ne l’avons jamais touché.

- le fichier icon.svg est simplement une image du logo de Godot qui est automatiquement dans chaque projet de base, nous ne l’avons pas supprimé car nous l’utilisons à certains endroits.

Maintenant pour ce qui est de l’arborescence que nous avons développé :

- le dossier “Image” contient uniquement des images (format PNG, JPEG) que nous utilisons à certains endroits dans le projet.

- le dossier “Interactions” contient tous les codes nécessaires aux interactions avec les objets, on passe par ce système pour changer de scène principale, lancer des animations, afficher des écrans. C'est le cœur du projet.

- le dossier “Objects” contient tous les fichiers nécessaires permettant de définir l’entièreté des objets présents dans le projet. Il contient 3 dossiers différents : “GDScript”, qui contient tous les fichiers .gd (fichiers code) associés aux objets, “Textures” qui contient les dossiers des textures (jpg) utilisées par les objets et enfin le dossier “Scenes” qui lui contient d’autres fichiers en fonction de l’utilité de l’objet. Nous reviendrons sur chacun des différents types d’objets plus tard dans cette documentation.

- le dossier “Player” contient tout ce qui est relatif au joueur : la scène du joueur, son script attaché ainsi que les fichiers gérant les touches entrées par le joueur.

- le dossier “Sounds” contient les fichiers .mp3 du projet.

- enfin le dossier “World” contient les scènes de tous les différents “levels” que nous avons fait et les scripts associés.

6. Projet

6.1. Objets de base

La première chose à faire lorsque l'on souhaite ajouter un nouvel objet dans le projet est de créer une nouvelle scène. Une fois cela fait, en fonction de la difficulté de l'objet, on va soit, s'il est complexe, le créer sur Blender et l'importer après sur Godot, si vous êtes dans ce cas, veuillez vous référer au Guide d'importation Blender -> Godot. Soit, s'il est simple, le créer directement sur Godot.

Prenons l'exemple d'un cube avec collision ayant le logo de Godot. Pour le faire, on va partir d'un nœud "Node3D" que l'on va renommer en "GodotCube". On peut ensuite rajouter un nœud "MeshInstance3D" et dans l'Inspecteur, cliquez sur "Mesh" puis "New Box mesh" afin d'avoir un véritable cube. Pour l'instant il n'a pas de "Material" (un ensemble de paramètres dont la couleur...), on va donc lui en donner un. Toujours dans l'Inspecteur, cliquez sur la catégorie "Geometry" puis dans "Material Override", cliquez sur "New Standard Material". Cliquez ensuite sur la boule blanche qui vient d'apparaître à côté de "Material Override", vous verrez alors tous les champs liés au material. Pour ajouter une le logo de Godot sur le cube, il suffit d'aller dans la catégorie "Albedo", de cliquer sur la flèche à côté de "Texture" et choisir "Load", ouvrez ensuite le fichier icon.svg, qui contient le logo de Godot de base. Vous aurez alors un cube Godot, si la texture n'est pas placée correctement sur l'objet, vous pouvez aller dans la catégorie "UV1" du matériau et changer le paramètre "scale".

Vous devriez obtenir quelque chose comme ça, sans les contours bleus autour :

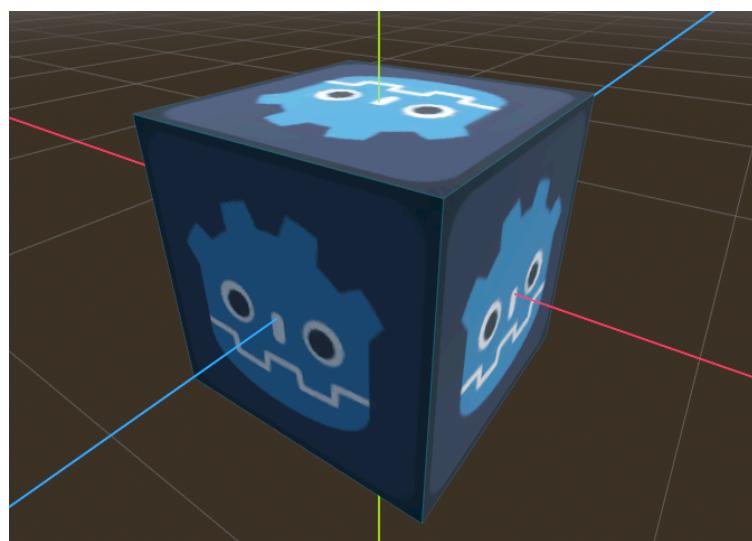


Figure 14 : Cube Godot

Bien, maintenant que l'on a notre cube modélisé, il faut ajouter une collision afin que l'on puisse pas rentrer dedans avec notre joueur. Les collisions sont un système un peu particulier qui ne fonctionne que si elles ont pour parent un noeud "StaticBody" ou "RigidBody". Vu que notre objet n'est pas censé bouger, on va donc rajouter un noeud "StaticBody3D" à notre "MeshInstance3D", puis on va rajouter un noeud "CollisionShape3D" à notre "StaticBody3D", on va ensuite aller dans l'Inspecteur et choisir une forme pour notre collision en cliquant sur "Shape" et choisissant "BoxShape3D". Elle devrait se mettre directement aux bonnes dimensions mais vous pouvez bouger la collision dans la fenêtre de visualisation si jamais. Voilà, vous avez maintenant un cube Godot ayant une collision et étant prêt à être utilisé en jeu.

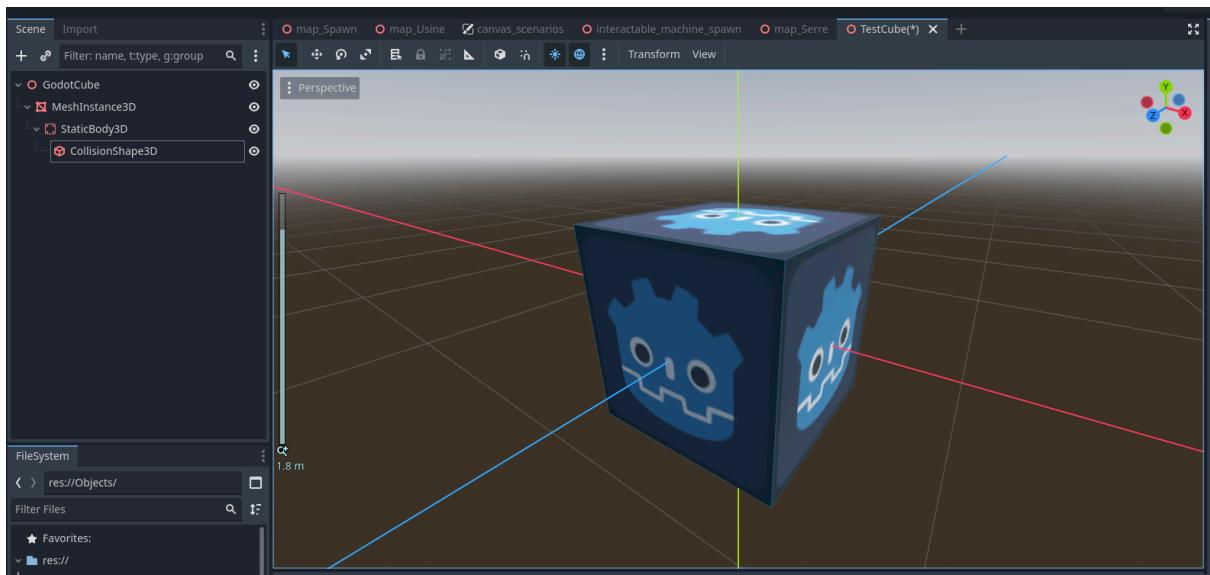


Figure 15 : Rendu final du cube Godot

On est ici dans le cas d'un objet simple et ayant une forme simple dans le cas d'objets plus complexes, il est nécessaire d'ajouter plusieurs collisions simples (comme des cubes et des cylindres) afin de faire la forme souhaitée. Dans le cas de notre objet le plus complexe : le StoneMill de l'usine, l'objet possède presque une quarantaine de collisions différentes afin d'épouser sa forme du mieux possible tout en restant optimisé. De même, la serre possède à elle seule une soixantaine de collisions.

En suivant la même technique mais en modifiant la taille du cube, vous pouvez désormais faire un sol, des murs et un plafond afin de faire un monde pour votre joueur.

6.2 Interactions

Scripts concernés : le script de l'interaction manager “interaction_manager.gd”, le script des zones d’interactions “interactable_area.gd”, un script d’objet dans le dossier “interactable”, et les scripts des inputs du joueur “xr_controller_3D....gd”.

Passons à un système plus complexe : celui des interactions. C'est le système, en jeu, qui s'occupe d'afficher un texte particulier lorsque l'on s'approche d'un objet puis, quand l'on appuie sur une touche, fais quelque chose de différent en fonction de l'objet avec lequel on interagit. Un tutoriel expliquant le système pour un jeu en 2D et sur lequel on s'est basé pour faire le nôtre peut être retrouvé dans cette vidéo (en Anglais) :

<https://youtu.be/ajCraxGAeYU?si=n7IGwSh0PmtTIXaU>.

Pour résumer, le système fonctionne sur le principe suivant : chacun des objets avec lesquels on peut interagir, possède deux noeuds particuliers : un noeud XRToolsInteractableArea (de la library XR Tools for Godot 4) avec un noeud CollisionShape3D, ces 2 noeuds sont une zone de collision qui réagit au passage du joueur mais sans le bloquer. Ils envoient des signaux quand le joueur entre et sort de cette zone d’interaction. Ils rendent simplement du texte visible lorsque le joueur est dans leur zone d’interaction.

Maintenant, imaginons que le joueur soit dans deux zones d’interactions différentes, quand il interagit avec l’un des objets, quel objet est censé interagir avec le joueur et quel objet est censé ne rien faire ? Pour résoudre ce problème, on crée une liste de zones actives (active_areas) qui contient toutes les zones d’interaction dans lesquelles se situe le joueur, et en fonction de celle dont le joueur est le plus proche, on n'affiche le texte que de l'une de ces zones.

Il nous faut par conséquent plusieurs fonctions, notamment pour enregistrer une zone d’interaction (_register_area() de interactable_area.gd), la supprimer (_unregister_area() de interactable_area.gd), des fonctions pour quitter l’interaction(_exit_interaction() de interactable_area.gd), une fonction qui regarde l’input que fait le joueur une fois dans une zone d’interaction (Button_function() des scripts des contrôleurs du joueur) etc...

Pour comprendre le système en détail, veuillez lire les scripts au début de cette partie.

Une fois cette partie du système faite, il suffit maintenant dans chaque objet auquel on souhaite interagir, de rajouter les deux noeuds mentionnés ci-dessus et de leur ajouter un script en fonction de ce que l'on souhaite faire avec eux. Par exemple pour les objets qui permettent de changer de scène (cf change_scene_object.gd), ils possèdent le même script chacun mais qui agit différemment en fonction de leur nom et du nom du noeud de base du monde (donc la scène dans laquelle ils se trouvent), il suffit ensuite de changer de scène en

fonction de ces noms et de rajouter les fonctions permettant de quitter l'interaction et le tour est joué.

6.3 Menus

Scripts et scènes concernés : un script d'un fichier canvas_....tscn dans le dossier "menu" et une scène d'un fichier canvas_....tscn et screen_....tscn.

Pour ce qui est des menus, en fonction de si le joueur peut interagir avec, il y aura la présence d'un script attaché à un nœud de la scène ou pas, s'il n'y en a pas, ça veut dire que le joueur ne peut pas interagir avec le menu et que ce menu sert juste à donner des informations au joueur, c'est par exemple le cas du menu des touches et du menu lié à la machine déhuller de l'usine. S'il y a un script, c'est que généralement l'interaction est faite par un bouton. Ce bouton permet de jouer une animation, de passer à un menu suivant... Puisqu'il y a un bouton, il y a forcément un script avec une fonction qui va définir ce que fait ce bouton lorsque l'on appuie dessus, s'il n'y en a pas, le bouton ne sert à rien.

Les fichiers canvas contiennent les scènes 2D qui seront affichées dans l'espace 3D par la suite. Ils sont composés de nœuds de contrôle qui sont des nœuds qui servent uniquement à la construction d'interfaces utilisateurs. Cette vidéo explique en détail chacun des nœuds de contrôle : https://youtu.be/5Hog6a0EYa0?si=hYcuMi9_8Soiw8HX.

Pour faire un résumé simple des noeuds les plus importants : le noeud de base de la scène sera toujours un "CanvasLayer", et son premier nœud fils sera un "Control" ayant la taille de base de l'interface (dans notre cas c'est généralement 1920 x 1080). Le noeud "Panel" "Container" sert à contenir un ensemble d'autres noeuds dans l'interface définie par le noeud "Control", "VBoxContainer" et un noeud contenant plusieurs autres noeuds verticalement et "HBoxContainer" les contient horizontalement. "Margin Container" permet de créer des marges entre les différents éléments à l'intérieur de l'interface en fonction de ses nœuds parents en enfant. "Label" est un nœud dans lequel on peut mettre du texte et régler sa taille.

C'est l'agencement de l'ensemble de ces noeuds qui permet de créer une interface de base comme celle-ci :

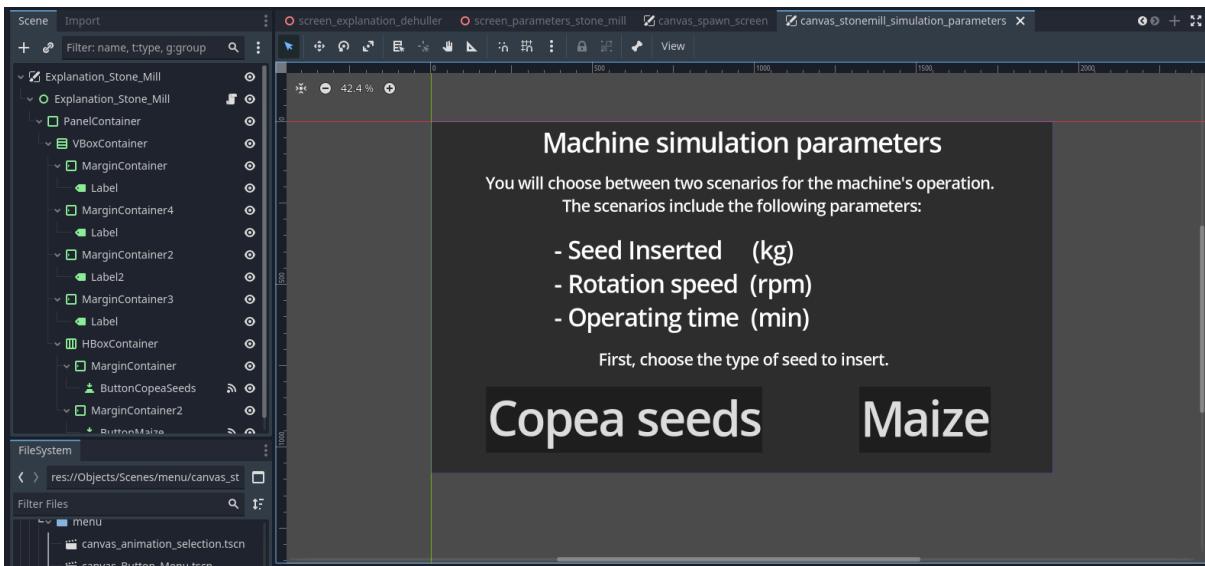


Figure 16 : Exemple d'Interface Utilisateur disposant de boutons

Afin de pouvoir placer cet écran 2D dans notre espace 3D, on utilise un autre noeud de la library XR Tools for Godot 4 appelé “Viewport2Din3D”, qui permet d’insérer notre interface utilisateur dans l’espace 3D et que le joueur puisse interagir avec via des pointeurs (les pointeurs sont des noeuds issues de la même library attaché aux contrôleurs de la scène “Player”), afin que celui-ci puisse cliquer sur les boutons des écrans.

Ainsi tous les objets screen_....tscn sont simplement des scènes contenant un noeud “Node3D” renommé et en enfant, un viewport2Din3D contenant dans son champ scène la scène canvas_....tscn du même nom.

6.4 Pickable objects

Scènes concernées : une scène d'un fichier `pickable_....tscn` dans le dossier "pickable".

Nous avons également ajouté quelques objets que l'on peut prendre avec les contrôleurs afin de s'amuser avec une fois en jeu. Ces objets ont pour base un nœud "XRToolsPickable" disposant du script attaché de base de la library XR Tools for Godot 4. Ils possèdent ensuite une collision shape afin qui va définir comment la forme qu'à la collision et un nœud "MeshInstance3D" qui leur donne une mesh. Pour que le système fonctionne, il faut rajouter, à chaque contrôleur du joueur, un noeud "XRToolsFunctionPickup" qui va permettre, en appuyant sur un bouton que l'on peut définir dans l'Inspecteur, de prendre les objets qui ont pour noeud père le noeud "XRToolsPickable".

Étant donné qu'il était difficile de savoir quel objet on peut prendre et quand on peut les prendre, nous avons rajouté une fonctionnalité de highlight aux objets que l'on peut prendre. En ajoutant un nœud "XRToolsHighlighVisible" avec un nœud "MeshInstance3D" en enfant. Ce noeud deviendra visible uniquement quand on pourra prendre l'objet et restera invisible dans le cas contraire. Tous les codes associés sont déjà écrits dans les scripts associé aux nœuds "XRToolsFunctionPickup" et "XRToolsPickable".

Nous avons donc par la suite rajouté une outline (contour lumineux), en changeant certains paramètres dans le matériau associé aux "MeshInstance3D" de ces objets à chacun des modèles des objets que l'on peut prendre et nous les avons stockés dans le dossier "outline".

Pour que la physique de ces objets soit la même que celle de notre projet, il faut bien penser à aller dans "Project Settings" dans l'onglet "Project" en haut à gauche, aller dans la catégorie "Physics" puis 3D et changez le Physics Engine à "Jolt Physics", cela donnera une physique bien plus réaliste que la physique du moteur physique de base de Godot (Godot devrait d'ailleurs bientôt changer de moteur physique de base donc cette option sera peut-être mise à la bonne valeur de base sur votre projet en fonction de votre version de Godot).

6.5 Animations

Nous allons dans cette partie aborder la création et la modification d'animations sur *Godot* à partir d'une armature. Sachez toutefois qu'il est également possible de faire des animations depuis *Blender* puis de les importer sur *Godot*.

Pour illustrer cette partie, nous allons partir du tutoriel "Assigner une armature à un ou plusieurs objets" de la Documentation Partie Blender et de ce qui a été fait, à savoir deux cubes, chacun lié à un bone.

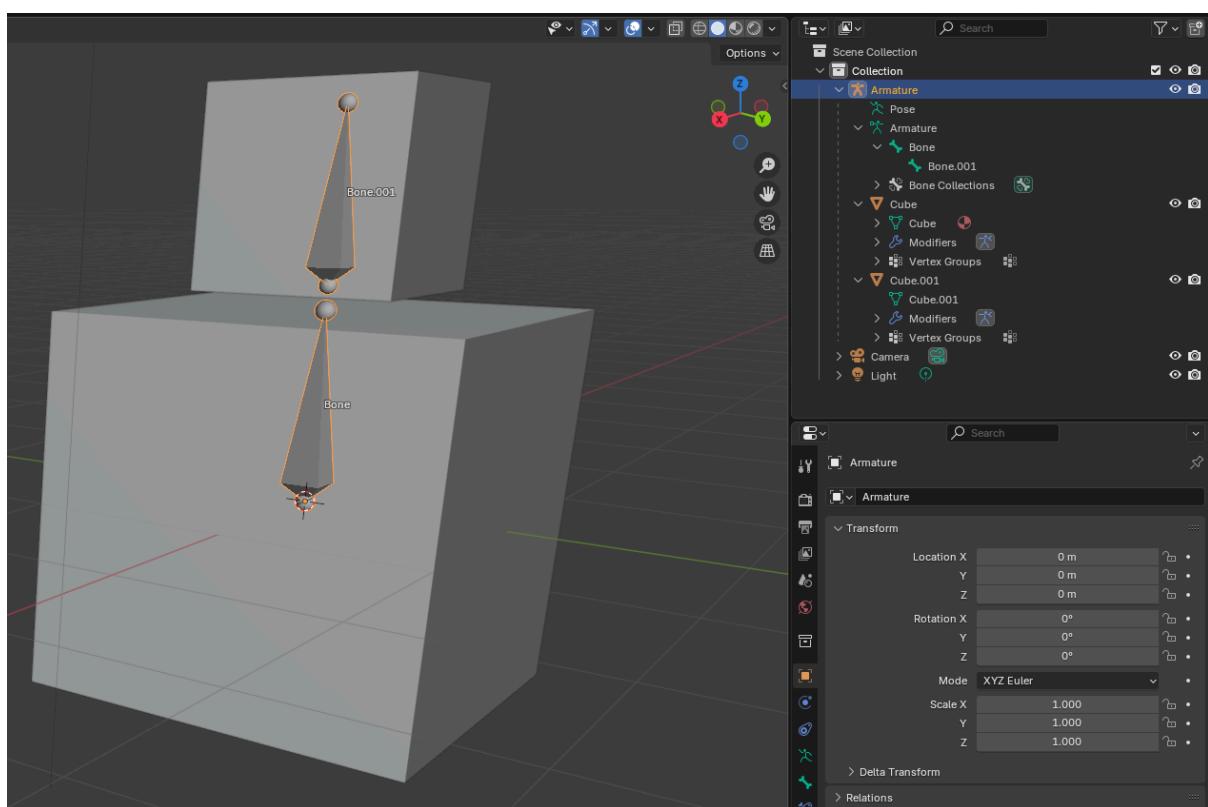


Figure 17 : Point de départ

Avant de passer à la suite, vous devez exporter ce projet au format .glb (veuillez vous reporter au Guide d'Importation d'Objets Blender à Godot).

Après avoir importé le projet en suivant les étapes, voici ce que vous devriez voir sur *Godot*:

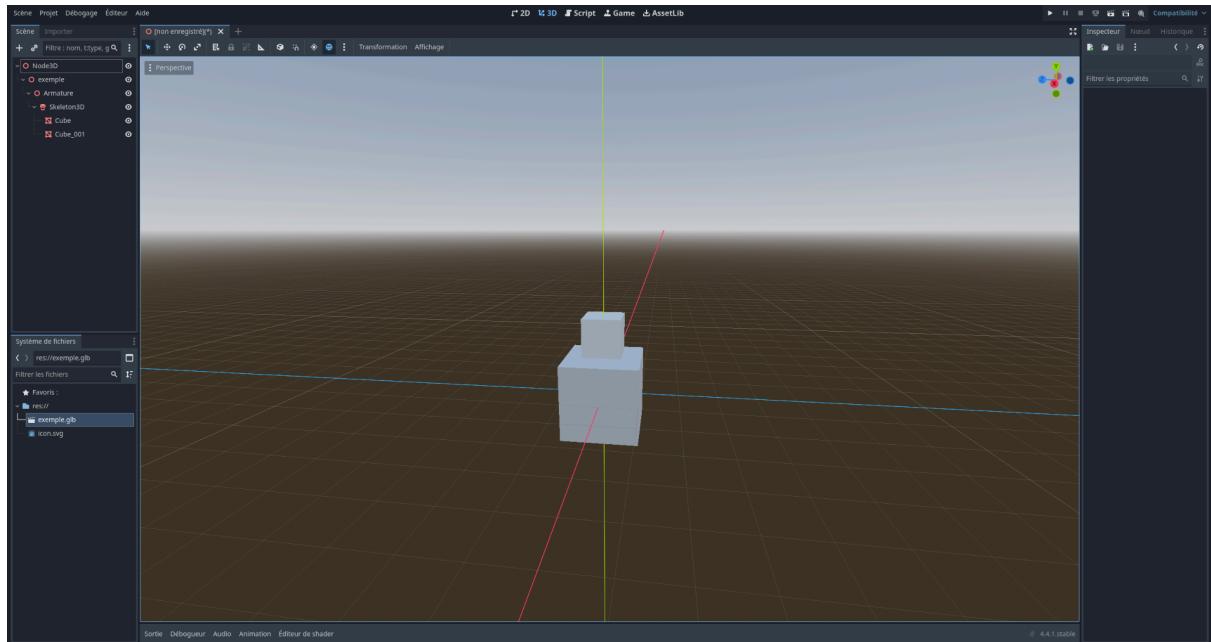


Figure 18 : Importation sur *Godot*

En Sélectionnant l'objet *Skeleton3D*, vous devriez voir apparaître un bouton au-dessus de la fenêtre de vue 3D de votre scène:

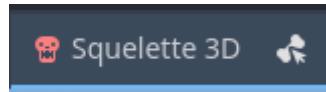


Figure 19 : Edition Mode Bouton

Si vous avez lu la Documentation Partie Blender, il s'agit de l'équivalent du *Pose Mode* pour *Godot*. Une fois que vous cliquez sur l'icône à droite sur l'image, cette fenêtre devrait apparaître à droite de votre écran :

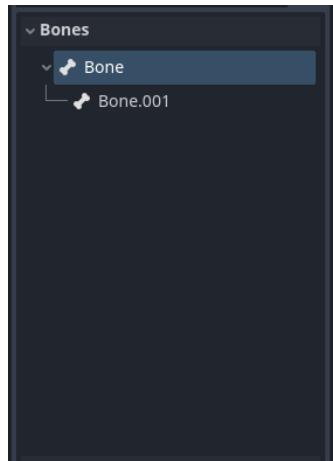


Figure 20 : Inspecteur de Skeleton

Vous pourrez ainsi sélectionner chacun de vos bones, les bouger/tourner et vous apercevoir que les objets qui leur sont associés bougent avec eux.

Maintenant que cette introduction sur les Bones et Skeleton3D est terminée, nous pouvons passer aux animations.

Pour cela, il vous suffit d'ajouter un nœud appelé AnimationPlayer en tant qu'enfant. Vous verrez alors la fenêtre suivante apparaître en-dessous :



Figure 21 : Éditeur d'animation

Appuyez sur Animation > Nouveau > nommez votre animation > ok.

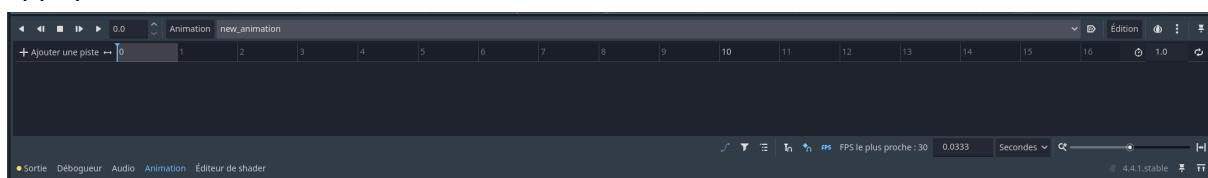


Figure 22 : Éditeur d'animation (2)

En cliquant de nouveau sur Skeleton3D dans l'arborescence de votre scène, vous verrez alors ces nouvelles icônes apparaître à droite du bouton du mode édition.

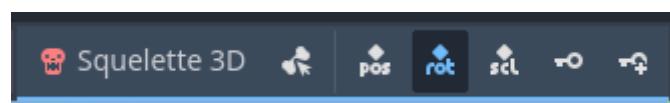


Figure 22 : Boutons d'ajout de piste

Le bouton “pos” permet de sauvegarder la position (coordonnées) de l'objet associé au bone à un timecode donné.

Le bouton “rot” permet de sauvegarder la rotation de l'objet associé au bone à un timecode donné.

Le bouton “scl” permet de sauvegarder l'échelle de l'objet associé au bone à un timecode donné.

Le système d'animation dans *Godot* est simple à comprendre.

Dans la frise chronologique , vous choisissez un Timecode (vous pouvez faire glisser le curseur bleu ou directement saisir le timecode dans la case à droite des options de lectures de l'animation), vous bougez/tournez vos bones dans la position/rotation voulue, et appuyez sur le bouton “Insérer une clé” (le bouton tout à droite sur la Figure 22).

Par défaut, *Godot* configure les animations de telle sorte à se terminer au bout d'une seconde. Pour modifier cela, il suffit d'insérer une nouvelle valeur dans le champ situé à droite de la frise chronologique.

6.6 Particules

Dans cette dernière partie, nous allons nous concentrer sur la création et la configuration de particules dans une scène 3D.

Pour cela, ajoutez en tant que nœud enfant à votre scène un nœud appelé CPUParticle3D.

Pour le moment, vos particules n'ont pas de forme (*Godot* le signale avec un warning). Dans l'inspecteur de particule, allez dans Drawing > Mesh > sélectionnez la forme que vous souhaitez donner à vos particules.

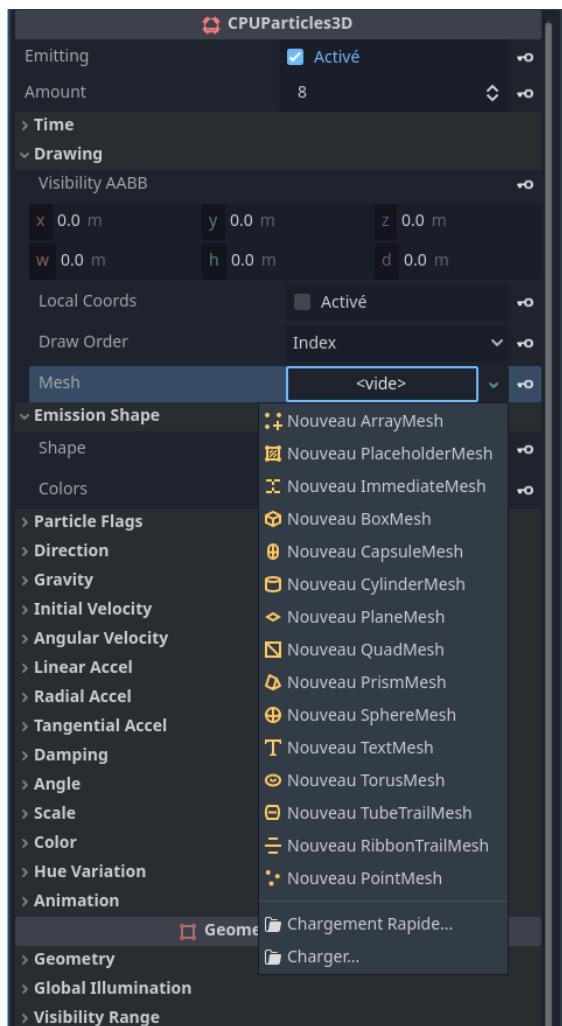


Figure 23 : Donner une forme à ses particules

Dans cette section, vous pourrez également configurer la taille de vos particules et leur Material.

Dans la section Emission Shape : permet de modifier la forme de l'émission des particules (depuis un point de l'espace, depuis une sphère, etc).

Enfin, la section Gravity modifie (littéralement) la gravité local à l'objet CPUParticle3D. Par défaut, elle est à 0 pour x et z, et à -9.8 pour y (comme l'accélération de pesanteur terrestre).

Configurer les particules pour les animations n'est pas compliqué non plus.

Après avoir créé votre animation, sélectionnez un timecode. Dans l'inspecteur de particule, il y a une section (la première) Emitting. Vous pouvez choisir d'activer ou désactiver l'émission de particules au timecode que vous avez choisi précédemment, puis appuyez sur la petite clé (ajouter une piste).

Cela conclut donc l'ensemble des objets que nous avons pu créer dans ce projet, les objets complexes ne sont qu'une somme des différentes mécaniques et objets que l'on vient de vous décrire dans les 6 dernières sous-parties.