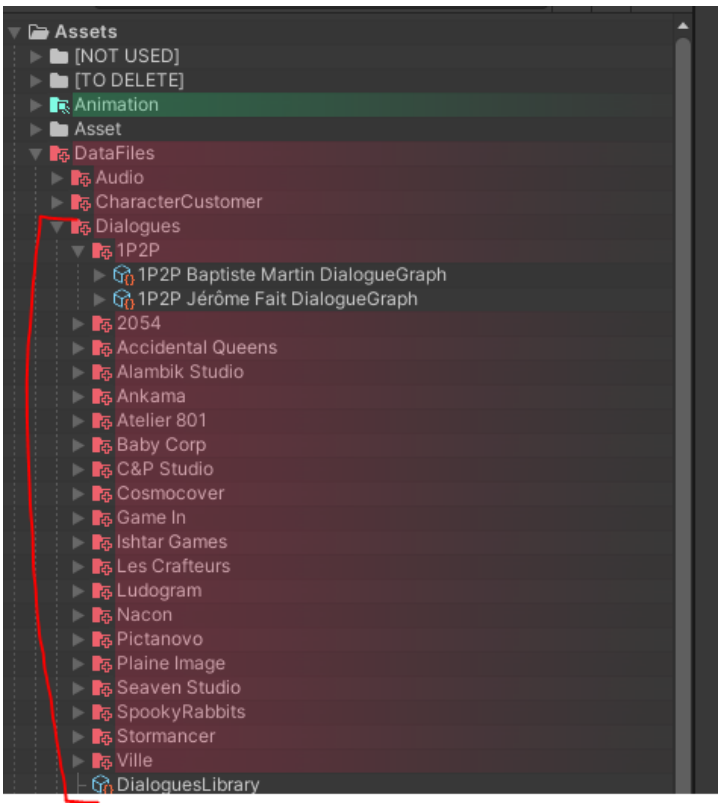
Document Explicatif

DIALOGUES

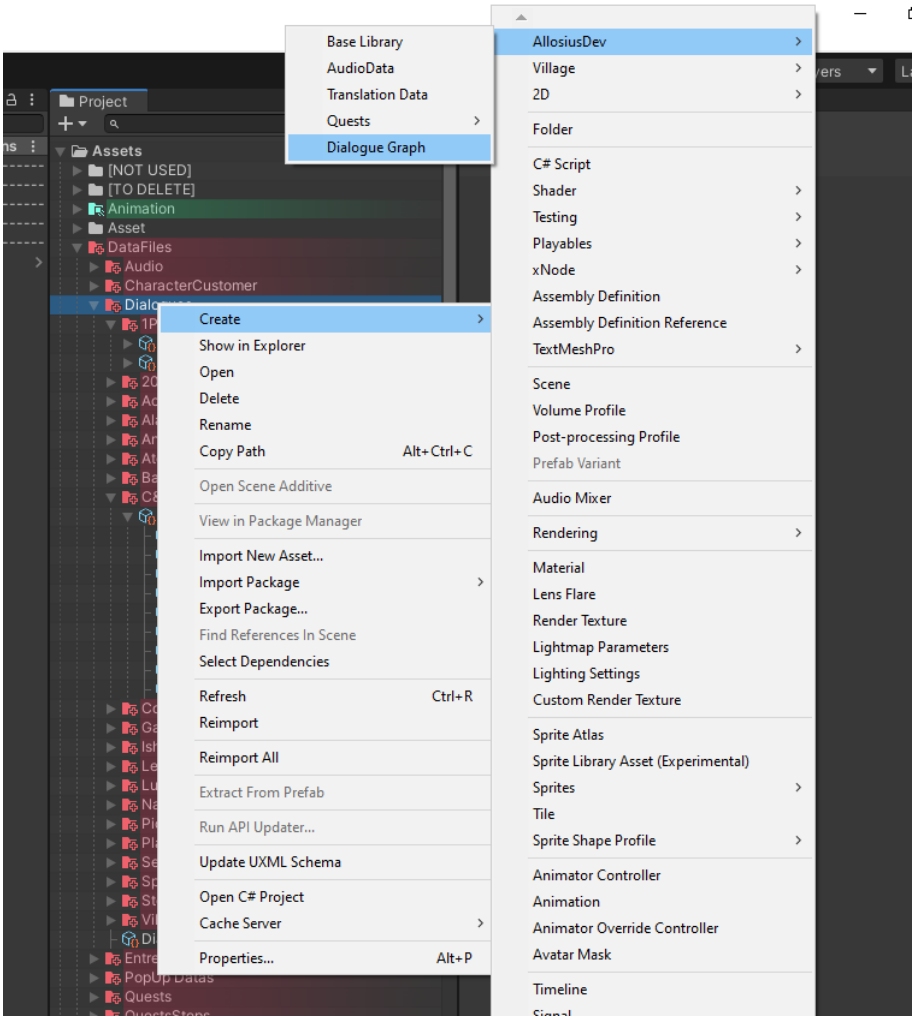
Sommaire :

1) Éditeur nodal

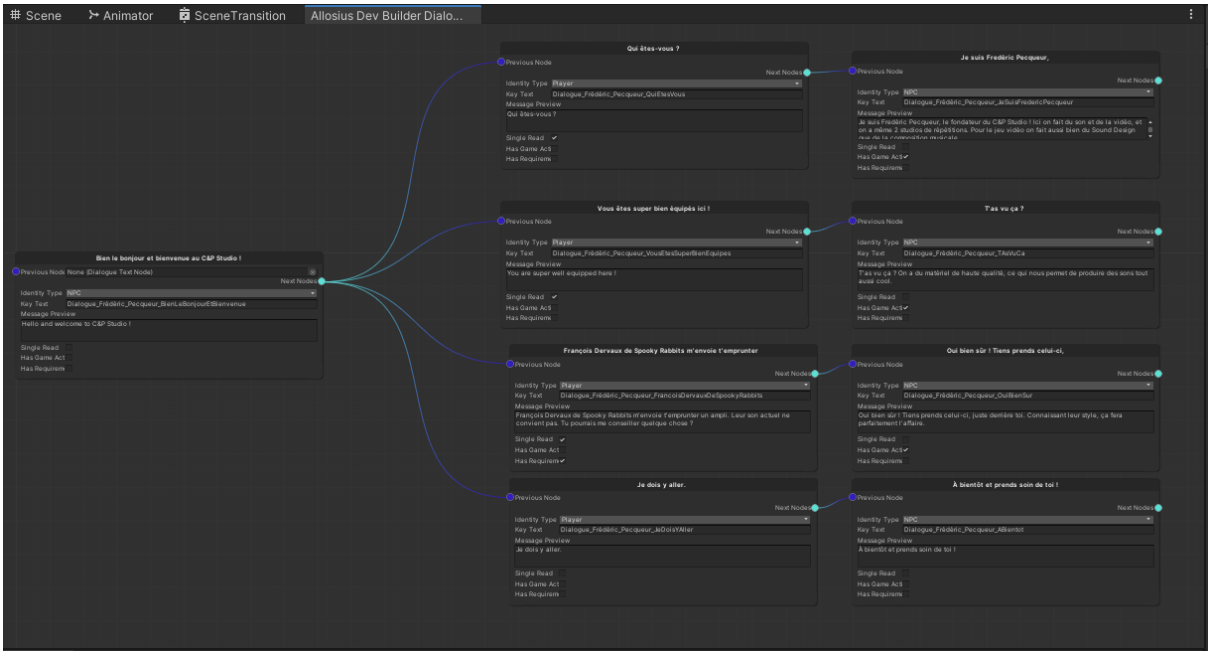
Le **système de dialogues** fonctionne via un éditeur nodal personnalisé qui va être interprété par le jeu lors de l'interaction avec les NPC, chaque dialogue est stocké dans un fichier data trouvables au chemin : **DataFiles > Dialogues** :



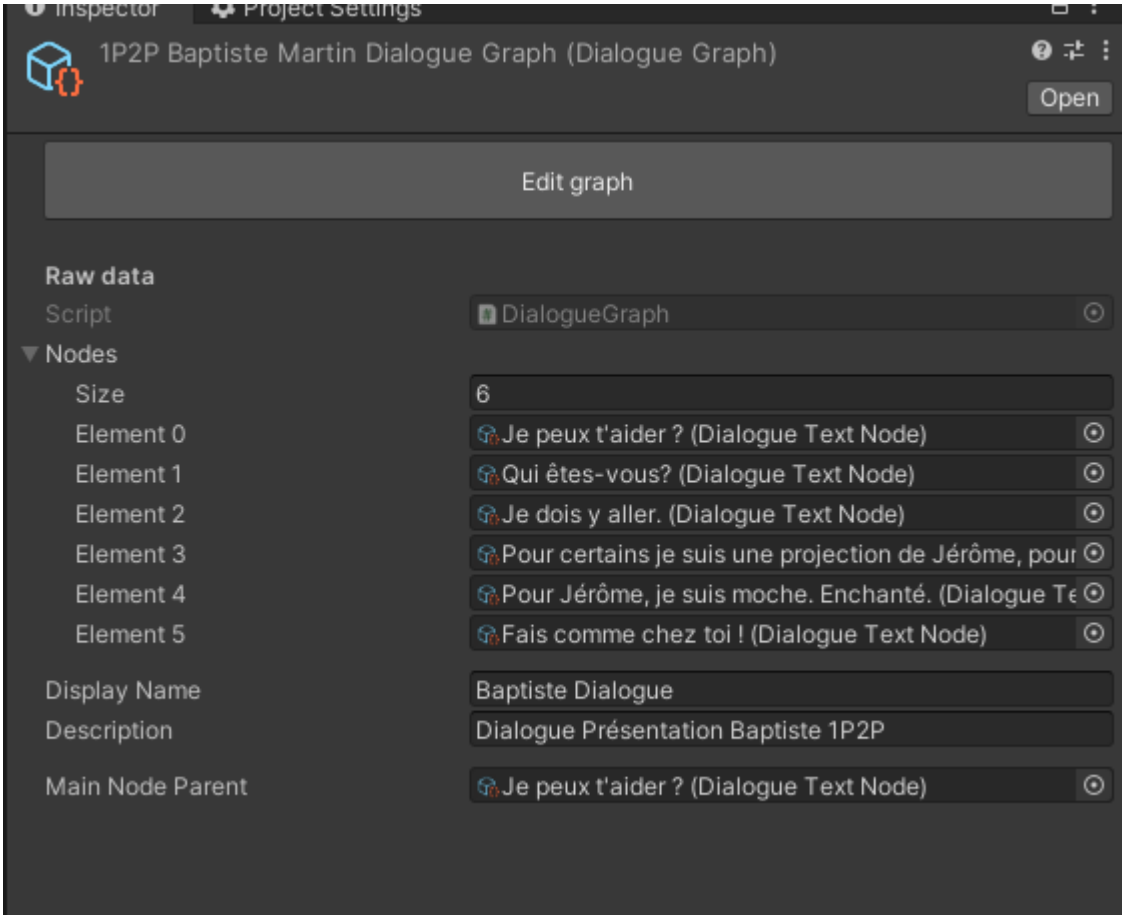
On peut créer un fichier de dialogue à l'emplacement souhaité par le menu : **Create > AllosiusDev > Dialogue Graph** :



L'édition des dialogues se fait ensuite via un graphique dans une fenêtre à part dans lequel on peut créer et configurer les nodes composant le dialogue :



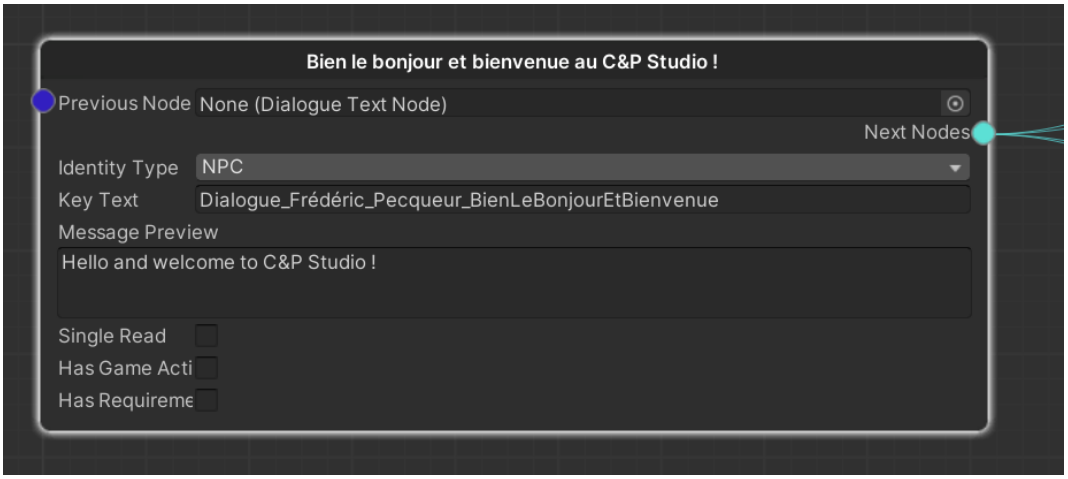
Chaque node peut être relié à d'autres nodes pour créer le "chemin séquentiel" du dialogue, une fois un node lu, sauf action particulière prioritaire, le système va lire son enfant, si il possède plusieurs nodes enfants, et que ce sont des nodes de type **Player**, ils vont chacun apparaître sous forme de choix, sinon, si il s'agit de nodes de type **NPC**, et qu'ils sont tous valides (conditions de lectures validées), l'un d'entre eux sera lu aléatoirement (peut être utile pour un dialogue générique variable, évitant que le pnj dise toujours la même chose, mais système pas spécialement prévu initialement pour ce genre de choses)

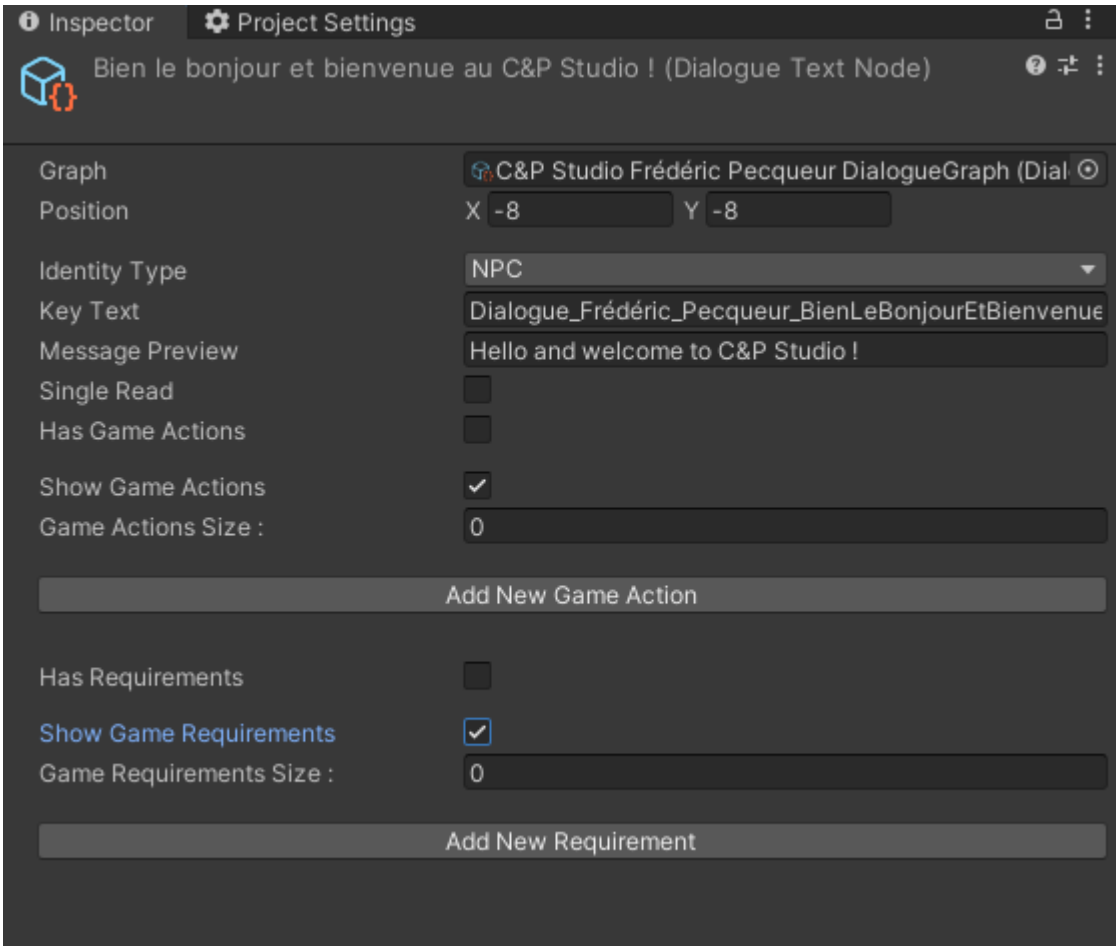


Dans l'inspecteur du dialogue, sont alors affichés :

- la liste des nodes composant le dialogue
- Le nom du dialogue (uniquement à but informatif pour s'organiser)
- La description du dialogue (uniquement à but informatif)
- Le main Node Parent qui est utilisé par certaines fonctions et actions dans le dialogue

Les nodes possèdent également individuellement des informations configurables et affichées dans leur inspecteur :



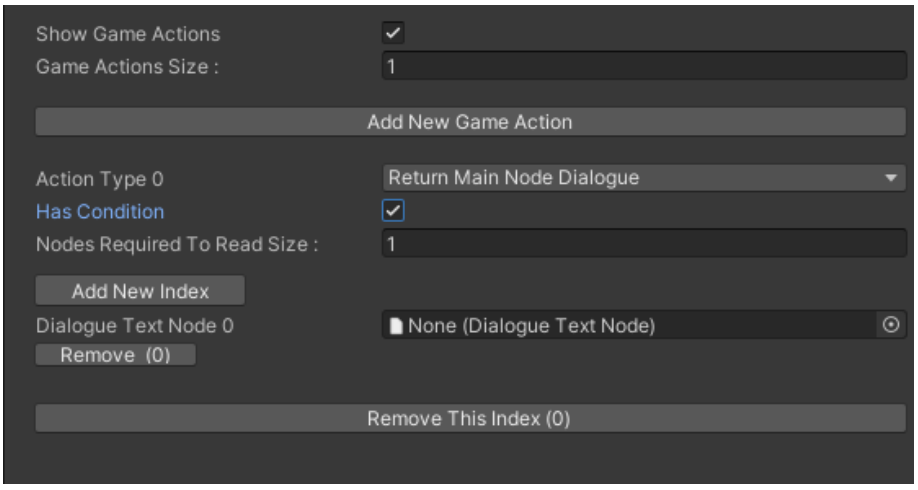


- **Graph** = fichier data de dialogue auquel le node appartient (à ne pas toucher)
- **Position** = position du node dans la grille de l'éditeur
- **Identity Type** = enum déterminant si le node est une réplique de PNJ ou un choix proposé au joueur
- **Key Text** = clé de traduction du texte souhaité à récupérer dans le dictionnaire JSON de traduction des dialogues (pour plus de précision voir document technique système de traduction)
- **Message Preview** = aperçu du message qui sera affiché en jeu dans la boîte de dialogue en fonction de la valeur du key text (à ne pas toucher, est affecté automatiquement par le système)

- **Single Read** = détermine si le node ne doit être lu qu'une seule fois pendant la lecture du dialogue ou si il peut réapparaître dans la liste des répliques (dans le cas d'un dialogue bouclé principalement) (la valeur est réinitialisée à la sortie du dialogue)
- **Has Game Actions** = détermine si le node doit lancer une ou plusieurs actions après avoir été lu (en fonction de la liste des actions possibles configurées plus bas)
- **Show Game Actions** = si actif, permet d'afficher la liste des actions potentielles à jouer, et de les configurer
- **Has Game Requirements** = détermine si le node doit être affiché en fonction de conditions (selon la liste des conditions de jeu possibles configurées plus bas)
- **Show Game Requirements**= si actif, permet d'afficher la liste des conditions potentielles pour lire le node, et de les configurer

Les **Game Actions** et les **Game Requirements** représentent une part non négligeable du fonctionnement des dialogues, ce sont d'ailleurs des classes génériques qui peuvent être utilisées pour n'importe quoi dans le jeu, en dehors des dialogues (pour la plupart des actions et des conditions, certaines étant spécifiques aux nodes, seraient inutiles en dehors des dialogues).

Chaque **Game Action** est définie par son **type** qui va déterminer les autres paramètres affichés dans l'inspecteur ainsi que la méthode à appeler correspondante, et par une variable **Has Condition** (utile uniquement dans le système de dialogues) qui si elle est active va permettre d'afficher une liste de nodes devant avoir été lu au moins une fois durant le dialogue pour que l'action se joue à la fin du node sélectionné.



Liste des types d'actions possibles :

Return Main Node Dialogue : Le prochain node lu sera le ou les nodes enfant du main node parent du dialogue défini dans l'inspecteur du data (est utile pour les dialogues tournant en boucle autour d'une série de questions principales par exemple, pour éviter les noeuds dans tous les sens quand on souhaite revenir en arrière dans la lecture de la séquence)

Add Quest : Ajoute une nouvelle quête au journal de quêtes, prend en paramètre un fichier data Quest To Add

- **Quest To Add** = quest data de la quête à ajouter (si le joueur ne possède pas déjà cette quête) (pour plus de précisions, voir document technique système de quêtes)

Complete Quest Step : Complète une étape spécifique d'une quête, prend en paramètres le fichier data Quest Associated ainsi que le fichier data Quest Step To Complete

- **Quest Associated** = quest data de la quête à laquelle appartient l'étape de quête à compléter (il faut que le joueur possède cette quête pour que la fonction s'exécute)
- **Quest Step To Complete** = quest step data de l'étape de quête que l'on souhaite compléter (il faut également que le joueur ait débloqué cette étape de la quête en cours pour pouvoir la compléter et donc que la fonction s'exécute)

Create Popup Scooter : (méthode qu'il faudrait revoir pour la rendre plus générale aux pop ups dans leur ensemble) : Fait apparaître le pop up indiquant qu'on a débloqué le scooter, prend en paramètre un fichier data Scooter PopUp Data

- **Scooter PopUp Data** = Pop Up Data du pop up à instancier

Add Item To Inventory : Ajoute un objet à l'inventaire, prend en paramètre le fichier data Item To Add

- **Item To Add** = fichier data Interactible Object de l'objet à ajouter à l'inventaire

Remove Item To Inventory : Retire un objet de l'inventaire, prend en paramètre le fichier data Item To Remove

- **Item To Remove** = fichier data Interactible Object de l'objet à retirer de l'inventaire (si on le possède, sinon ne fait rien)

Launch Mini Game : (méthode qu'il faudrait revoir pour la rendre plus générale, elle est actuellement trop spécifique) : Lance le mini jeu de rythme

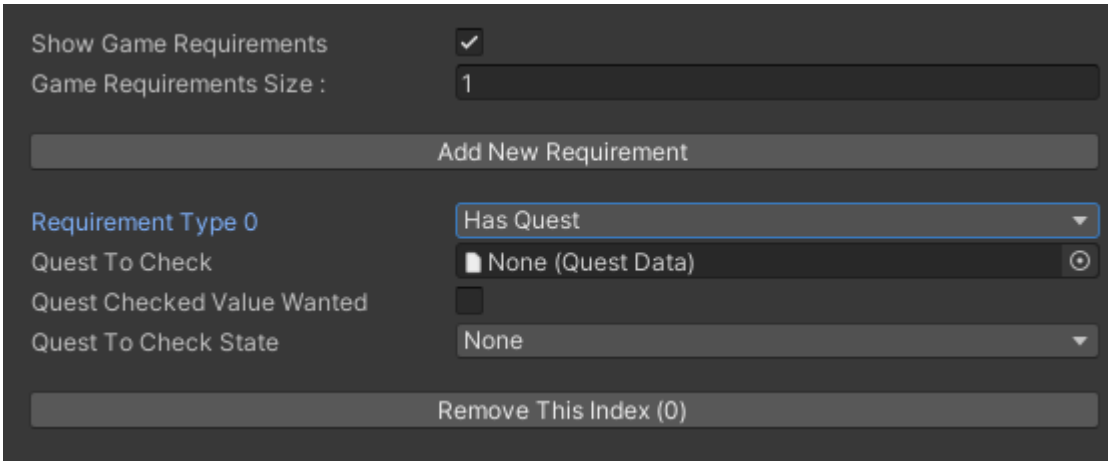
Launch Fade : Lance un fade in / fade out de l'écran avant de passer au node suivant, prend en paramètres la durée Fade Duration, ainsi que la durée Fade Out Switch Duration

- Fade Duration = Durée de la transition vers le fade in ou le fade out
- Fade Out Switch Duration = Durée d'attente avant d'inverser le fade

Launch Dialogue : Lance un nouveau dialogue si c'est possible, prend en paramètres le fichier data Dialogue To Launch, ainsi que le booleen Launch Dialogue To Main Node

- **Dialogue To Launch** = fichier data Dialogue Graph du dialogue à lancer
- **Launch Dialogue To Main Node** = si actif, lance le nouveau dialogue directement après le main node parent du dialogue (équivalent à lancer l'action Return Main Node Dialogue directement au lancement du nouveau dialogue)

Chaque **Game Requirement** est définie uniquement par son **type** qui va déterminer les autres paramètres affichés dans l'inspecteur ainsi que la méthode à appeler correspondante, pour déterminer si la condition est valide ou non.



Liste des types de conditions possibles :

Has Quest : Vérifie si le joueur possède ou non une quête spécifique, prend en paramètres le fichier data Quest To Check, la valeur booléenne Quest Checked Value Wantend, et une enum Quest To Check State

- **Quest To Check** = fichier Quest Data de la quête qu'on souhaite vérifier
- **Quest Checked Value Wanted** = booléen déterminant si l'on souhaite vérifier que l'on a la quête ou qu'on ne l'a pas (si il est vrai, on vérifier qu'on a la quête, si il est faux, on vérifie qu'on a pas la quête)
- **Quest To Check State** = enum prenant 3 valeurs possibles (si None, n'est pas pris en compte, peu importe l'état de la quête vérifiée, si Uncompleted, on vérifie que la quête n'est pas encore complétée, enfin si Completed, on vérifie que la quête est complétée)

Quest State : Similaire à la condition Has Quest, mais sur une étape de quête, on vérifie si le joueur possède une étape spécifique d'une quête, prend en paramètres, le fichier data Quest Associated To Check, le fichier data Quest Step To Check, la valeur booléenne Quest Step Checked Value Wanted, et l'enum Quest Step To Check State

- **Quest Associated To Check** = fichier Quest Data de la quête à laquelle appartient l'étape de quête qu'on souhaite vérifier
- **Quest Step To Check** = fichier Quest Step Data de l'étape de quête qu'on vérifie
- **Quest Step Checked Value Wanted** = booléen déterminant si l'on souhaite vérifier que l'on a débloqué l'étape de quête spécifie ou au contraire qu'on ne l'a pas (si il est vrai, on vérifie qu'on a l'étape de quête, si il est faux, on vérifie qu'on a pas l'étape de quête)
- **Quest Step To Check State** = enum prenant 3 valeurs possibles (si None, n'est pas pris en compte, peu importe l'état de l'étape de la quête vérifiée, si Uncompleted, on vérifie que l'étape de la quête n'a pas encore été complétée, enfin si Completed, on va vérifier que l'étape de la quête spécifié soit complétée)
-

Has Item : Vérifie si le joueur possède un objet spécifique dans son inventaire, prend en paramètre le fichier data Item To Check

- **Item To Check** = fichier data Interactible Object de l'objet qu'on souhaite vérifier

Has Required Rythme Game Rank : Vérifie si le rang qu'a obtenu le joueur lors de sa dernière partie de mini jeu de rythme est égale à une certaine valeur, prend en paramètre le rang souhaité (méthode potentiellement à revoir pour la rendre plus générale lorsqu'il y aura d'autres minis-jeux intégrés)

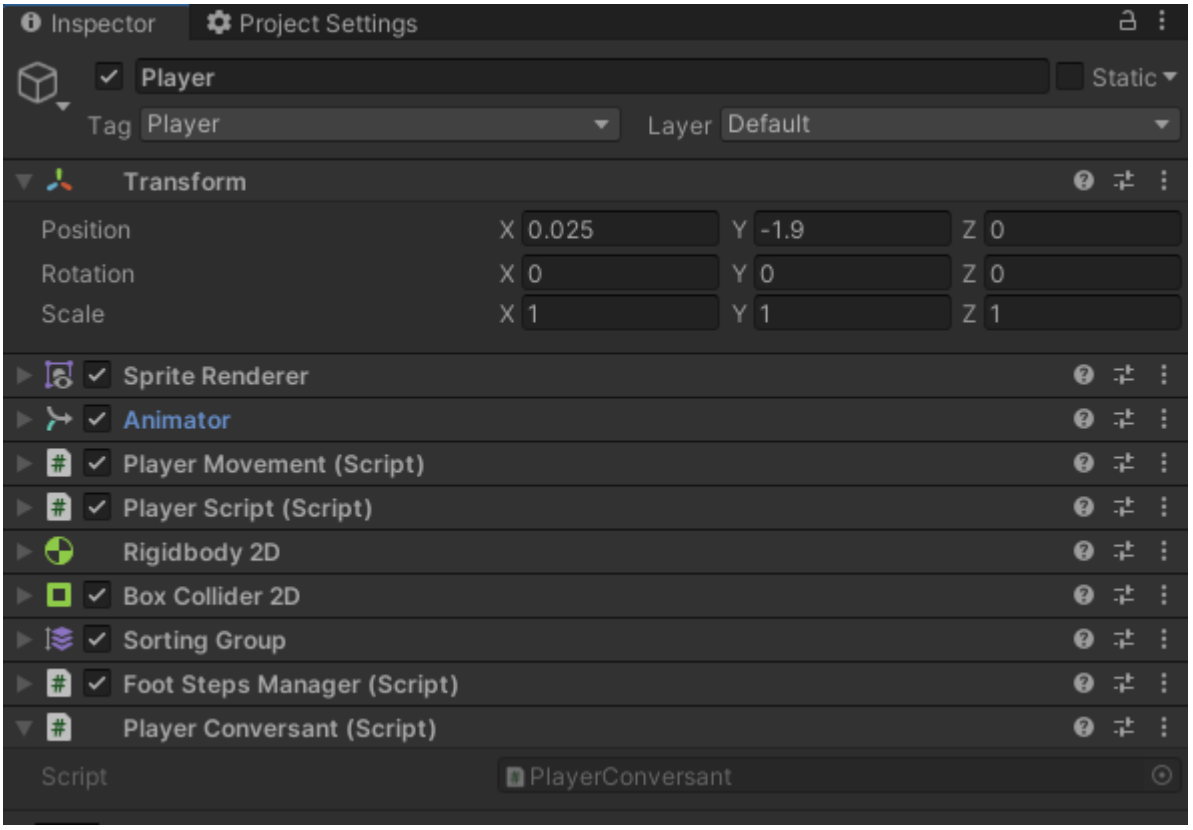
- **Rythme Game Rank Required =** enum du rang souhaité pour valider la condition (F, D, C, B, A, ou S)

2) Fonctionnement du système de dialogues

En jeu, pour que les dialogues fonctionnent et puissent être lu correctement, il est nécessaire de s'assurer que certains scripts soient présents et correctement paramétrés.

Player Conversant :

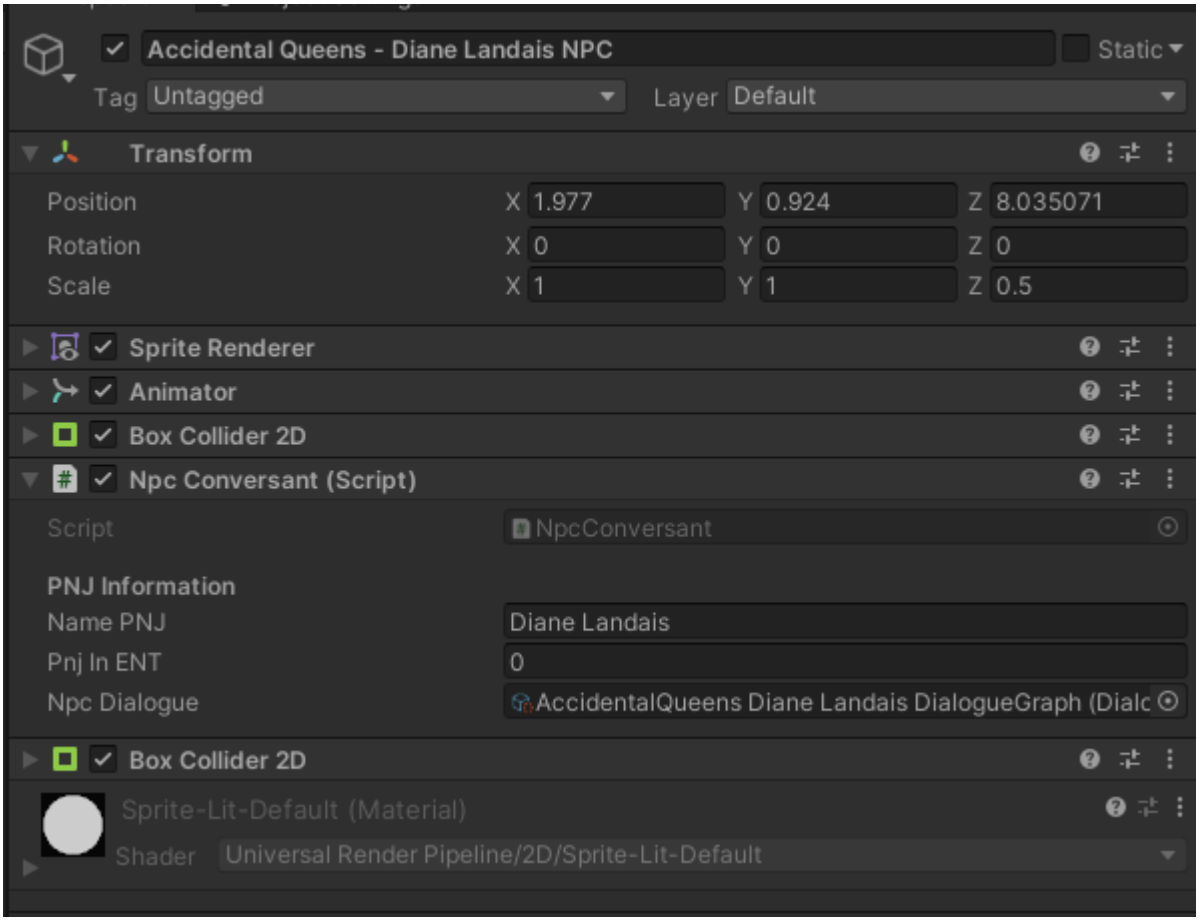
Tout d'abord, le prefab du joueur doit posséder dans ses components le script **Player Conversant** :



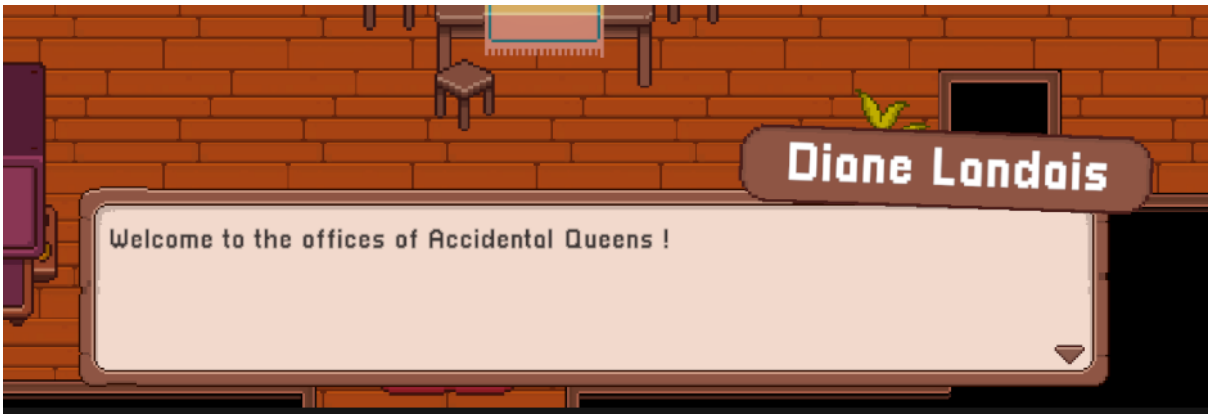
C'est par ce script que passe toute une partie de la logique du système de dialogue lors de l'interaction avec un PNJ ou toute entité possédant un dialogue.

NPC Conversant :

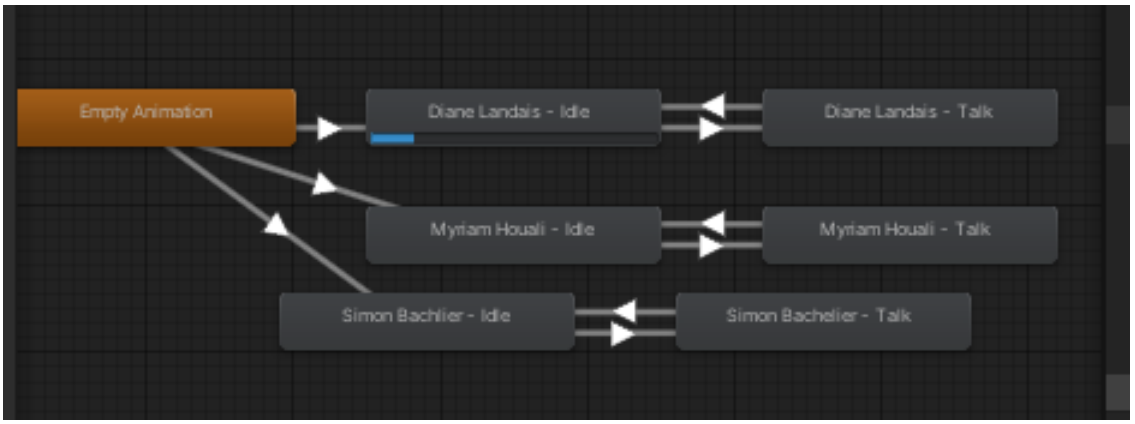
Ensuite, chaque entité du jeu possédant un dialogue, doit se voir affecter un script **NPC Conversant**, qui permettra lors de l'interaction avec le joueur de lancer le dialogue passé en paramètre :



La variable **Name PNJ** correspond au nom qui sera affiché dans la boîte de dialogue, lorsque le PNJ va parler.



La variable **Pnj In ENT** correspond à quel index de transition de l'animator le personnage correspond, c'est une variable utile uniquement pour les entités possédant une animation et pour un même animator, pouvant prendre plusieurs séries d'animations possibles :

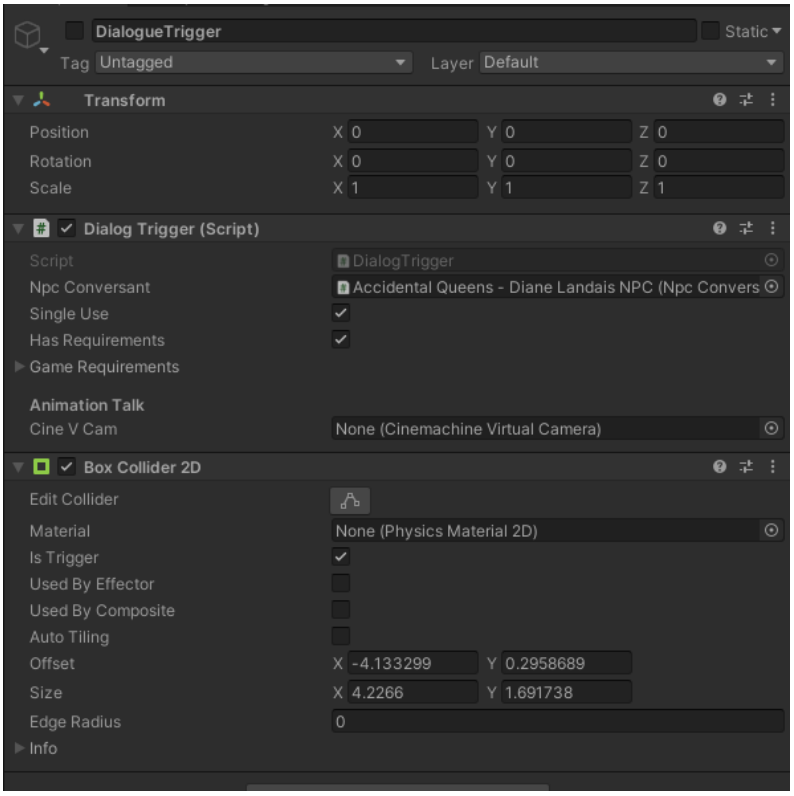


Pour une valeur du **Pnj In Ent** de 0, l'animator suivra le chemin commençant par l'état **Diane Landais - Idle**, si elle était de 1, ce serait le chemin **Myriam Houali - Idle**, et ainsi de suite.

Enfin la variable **Npc Dialogue** correspond au fichier data de dialogue qu'on veut lire lors de l'interaction avec le PNJ.

Dialog Trigger :

Le script **Dialog Trigger** peut être une alternative au **NPC Conversant**, dans le cas où dès que le joueur entre dans une certaine zone un dialogue se lance automatiquement :

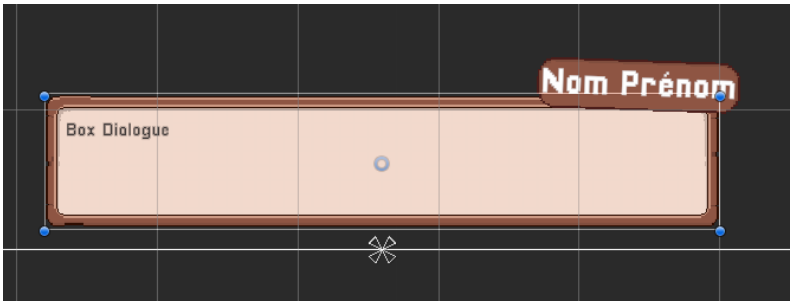


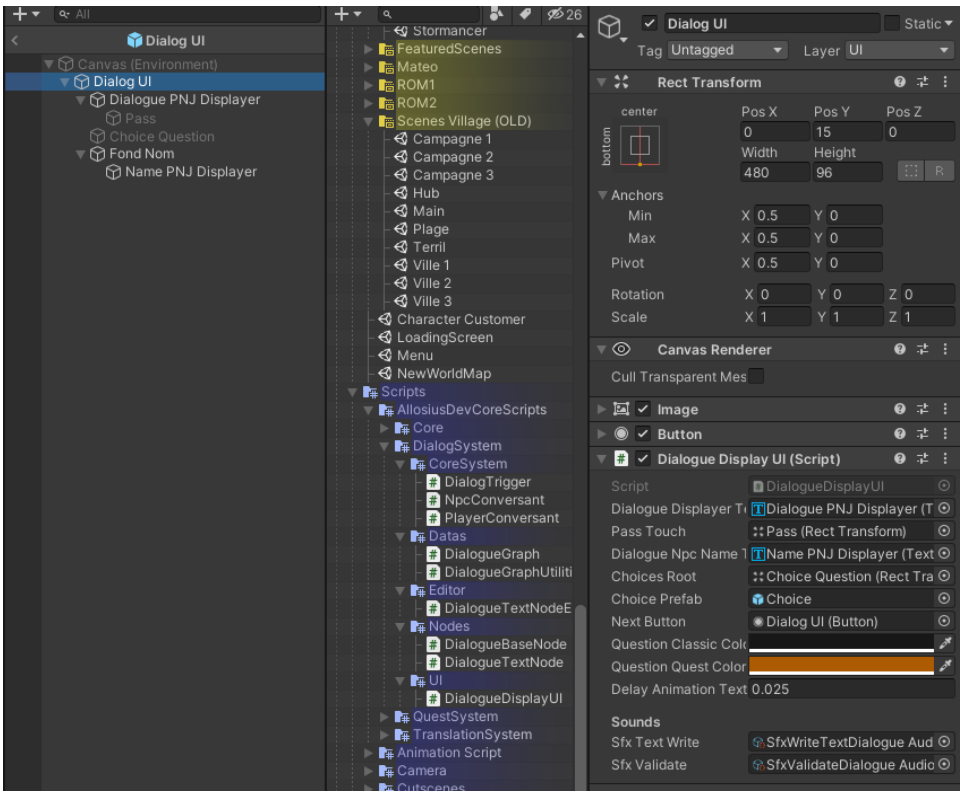
On peut définir sur ce script :

- Le **dialogue** à lancer comme pour le **NPC Conversant**
- La valeur du booléen **Single Use** qui va définir si le trigger doit s'activer à chaque fois que le joueur entre en contact avec, ou une seule et unique fois
- La valeur du booléen **Has Requirements**, qui de façon similaire au système de dialogue, ne va permettre d'activer le trigger qui si les conditions stockées en dessous sont validées
- La liste des **Game Requirements** à valider pour l'activation du **Dialogue Trigger**, si **Has Requirements** est active
- De façon facultative, on peut référencer une **camera virtuelle de cinemachine** si on souhaite que le point de vue de la caméra changer pendant le dialogue (on laisse sur none si on ne veut rien faire de spécial)

Dialogue Display UI :

Enfin, le dernier script indispensable pour le fonctionnement du système de dialogues, est le **Dialogue Display UI**, ce script va permettre d'interpréter le dialogue lu par le joueur lors de l'interaction avec un PNJ, et l'afficher correctement dans la boîte de dialogue. Il doit donc impérativement être placé et configuré sur l'objet d'UI contenant la boîte de dialogue et qui est instancié au lancement du jeu :



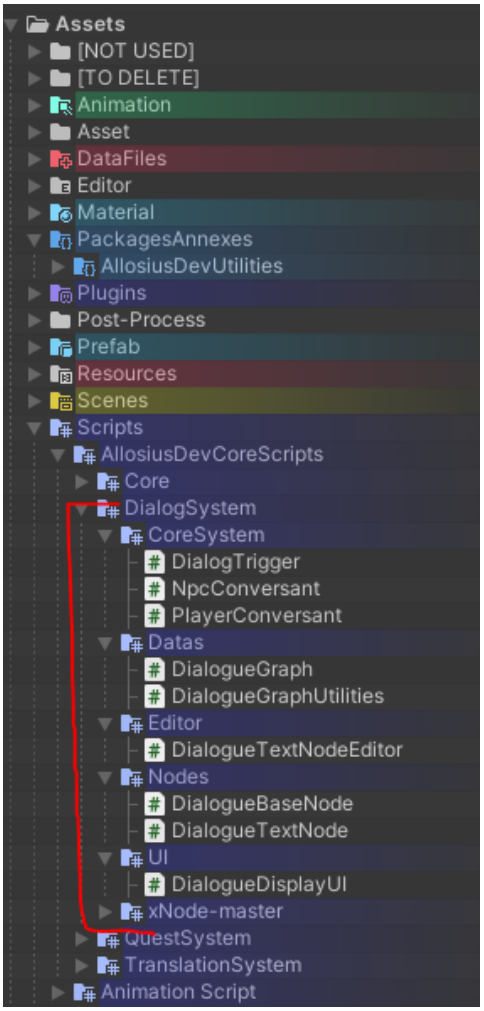


Les différents paramètres qui doivent être configurés sur ce script sont :

- **Dialogue Displayer Text** = UI de texte qui va recevoir le message du dialogue en cours
- **Pass Touch** = Visuel sans interaction du petit curseur en bas à droite de la boîte de dialogue indiquant que le joueur peut passer à la suite
- **Dialogue Npc Name Text** = UI de texte qui va recevoir le nom du PNJ en train de parler
- **Choices Root** = Objet vide parent dans lequel seront instanciés en enfants, les choix du joueur
- **Choice Prefab** = Prefab d'UI de choix du joueur qui va être instancié dans le Choices Root lorsque le joueur devra faire un choix de réponses
- **Next Button** = Bouton sur lequel le joueur devra appuyer pour passer à la séquence suivante du dialogue
- **Question Classic Color** = Couleur dans laquelle va apparaître le texte à l'écran par défaut
- **Quest Quest Color** = Couleur dans laquelle va apparaître le texte à l'écran, si il s'agit d'une réplique concernant l'accomplissement d'une quête
- **Delay Animation Text** = Influe sur la vitesse à laquelle le texte va s'écrire à l'écran durant la discussion
- **Sfx Text Write** = fichier data de son qui va se jouer tant que le texte n'a pas fini de s'écrire à l'écran
- **Sfx Validate** = fichier data de son qui va se jouer lorsque le joueur valide la séquence en cours et passe à la suivante

3) Répertoires des principaux scripts utilisés

Vous pouvez retrouver l'ensemble des scripts utilisés par le système de dialogues (éditeur + scripts de jeu) au chemin : **Scripts > AllosiusDevCoreScripts > DialogSystem :**



Les scripts du dossier **CoreSystem** sont les scripts utilisés en jeu pour interpréter et lancer les dialogues

Les scripts du dossier **Datas** correspondent aux scripts permettant de créer les fichiers datas de dialogues utilisés par l'éditeur

Le script du dossier **Editor** correspond à l'éditeur personnalisé qui a été fait pour afficher comme souhaité les informations stockées sur les nodes du data de dialogue

Les scripts du dossier **Nodes** correspondent aux scripts gérant les fichiers datas de ces nodes ainsi que leur affichage dans le graph d'édition

Le script du dossier **UI** stocke le script permettant de gérer l'affichage des dialogues lus en jeu

Enfin le dossier **xNode-master** stocke le tool permettant de créer et gérer l'affichage du graphique personnalisé et d'y dessiner les nodes de dialogues (C'est dans ces scripts qu'on gère comment sont affichés la grille ainsi que les nodes, et les informations qu'on peut ou non voir dessus)