InGame Dialogue - Notice

17-06-2021 par Alexandre Venet https://twitter.com/alhomepage

1. Introduction

InGame Dialogue est un système de gestion rapide de dialogues apparaissant en sous-titres pendant les phases de *gameplay*. Ce système est pensé pour être intégré dans chaque scène, seulement s'il est requis. Les répliques s'enchaînent de façon linéaire et il n'y a pas d'embranchements.

Il requiert un ensemble de **fichiers** à installer : son **noyau de scripts** (*MonoBehaviour*, *ScriptableObject*, classes de types), les **interfaces utilisateur pour l'éditeur Unity** (*Editor*), les **éléments d'UI pour le produit final** (*Prefabs*), une scène d'exemple.

Il dépend du package *TextMeshPro* pour les textes d'UI et leur mise en forme.

Une fois les fichiers installés et selon les besoins, on créera autant de fichiers **InGameDialogue** qu'il est nécessaire.

Le principe est le suivant. Des objets déclenchent une fonction du gestionnaire **InGameDialogueManager** en lui passant une référence de fichier **InGameDialogue**. S'il n'y a pas de condition interdisant l'affichage du dialogue, alors les **répliques** s'affichent les unes après les autres, selon un **délai** personnalisé pour chacune.

Les fichiers les plus importants sont **mis en forme** par **scripts d'éditeur** pour rendre le travail d'édition confortable. Passer le curseur sur les noms fait apparaître la plupart du temps **une bulle d'information**. Les noms qui apparaissent en édition peuvent être différents des noms de variable réels, ceci à des fins de lisibilité. J'utilise dans le présent document les noms tels qu'ils apparaissent dans *Unity*.

Le **menu** d'*Unity* se voit augmenté de l'entrée suivante : **Tools > Reliquia > Dialogue In Game**. On y trouvera des liens sélectionnant les fichiers d'aide.

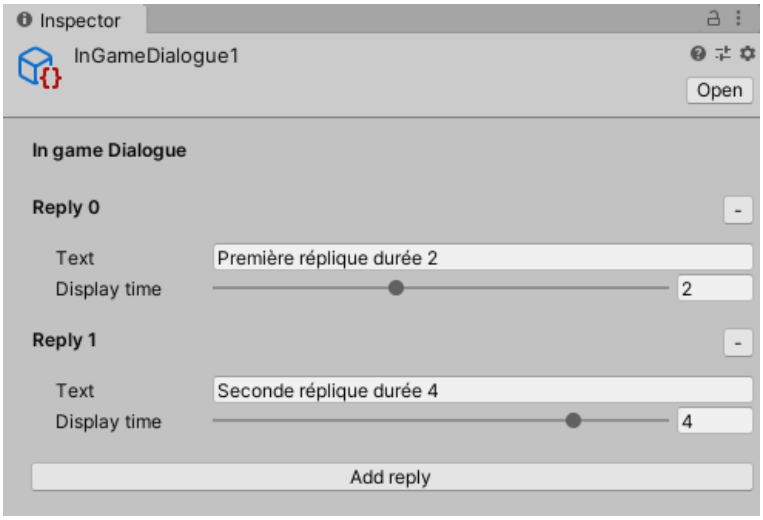
La scène de test fournie a ce chemin : **Alexandre_SceneTest/DialogueTest.unity**.

2. InGameDialogue

Un **dialogue *in-game*** est un **ensemble de données** que l'on renseigne dans un fichier et qui est affiché dans le texte d'UI prévu.

Un dialogue « en jeu » est un fichier *ScriptableObject* du type **InGameDialogue** que l'on crée dans ***Project* puis clic droit > Create > ScriptableObjects > InGameDialogue > InGameDialogue**.

Ce fichier contient les **répliques** qui vont s'afficher à l'écran. Pour ajouter une réplique, cliquer sur le bouton **Add reply**. Pour supprimer une réplique, cliquer sur le bouton « - ».



Chaque réplique dure un certain **temps** avant que la suivante ne s'affiche ou que le dialogue ne se termine. Or, une réplique peut contenir plus de texte qu'une autre et le temps de lecture nécessaire se verrait donc rallongé. Pour permettre à l'utilisateur d'avoir le temps de lire le texte, et bien que la séquence soit de *gameplay* et que l'attention à la lecture soit moindre que dans une boîte de dialogue, alors le temps d'affichage à l'écran est paramétrable. L'intervalle est de 0 à 5 secondes.

Pour paramétrer le temps d'affichage, déplacer la glissière (*slider*) ou bien entrer directement une valeur dans le champ. Le nombre est décimal et accepte des nombres entiers.

On créera autant de dialogues de ce type qu'il est nécessaire. Ces données étant des *Assets*, alors elles sont stockées sur le disque dur.

Le texte peut être de différentes longueurs. Au-delà d'un certain nombre de caractères, plus rien ne sera affiché dans la zone UI prévue.

Les champs de saisie se redimensionnent selon la longueur du texte. Ils peuvent accueillir une mise en forme au moyen d'une **feuille de style** (voir le chapitre idoine).

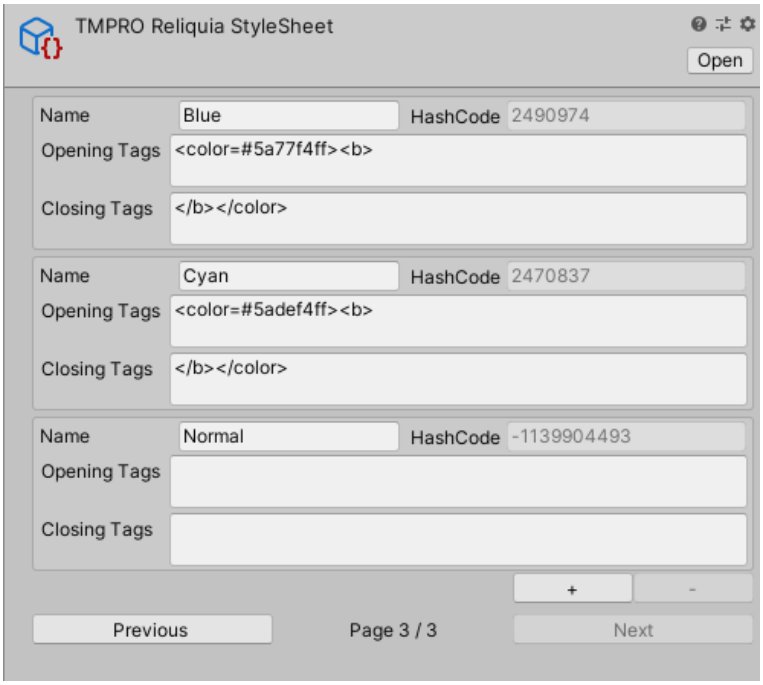
3. Feuille de style

Ce chapitre reprend les contenus de la notice du **Dialogue System**. En effet, les deux systèmes de dialogue partagent les mêmes ressources **TextMeshPro**.

DialogueColorsRef n'est pas requis car ce fichier est spécifique de **Dialogue System**.

Pour éditer les **styles du texte** de **réplique**, un fichier est mis à disposition : un fichier *ScriptableObject* des styles *TextMeshPro* nommé **TMPRO Reliquia StyleSheet**.

Ce fichier *ScriptableObject* se trouve dans le *Project*. Il est une copie du fichier fourni par défaut par *TextMeshPro*. *TextMeshPro* utilisera les styles de ce fichier spécifique du projet. Pour fonctionner dans la version d'Unity utilisée pour *Reliquia* (la version 2019.3), ce fichier doit être renseigné dans **menu Edit > Project Settings… > TextMesh Pro > Settings** ; pour cette version d'Unity et de *TextMeshPro*, la référence à ce fichier s'effectue par la classe (pour d'autres versions, *TextMeshPro* autorise d'autres accès).



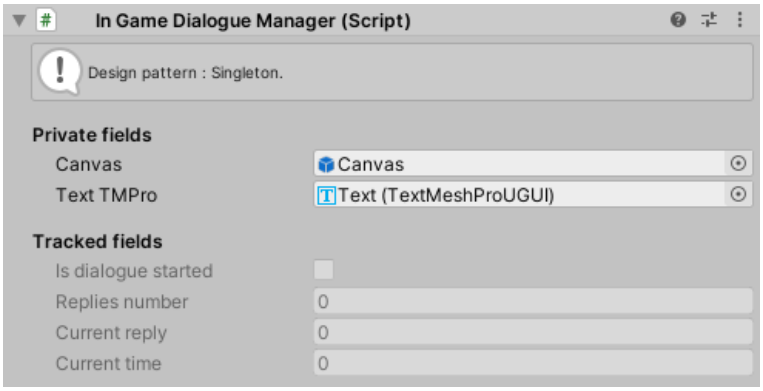
Ce fichier contient une liste de **styles**. Sa structure dépend de *TextMeshPro* et il convient de la conserver. Ces styles présentent un **Name** (nom), un numéro **HashCode** (identifiant unique), des **Openings Tags** (balises d'ouverture) et des **Closing Tags** (balises de fermeture). Par exemple, le champ prévu pour la couleur **Blue** présente les balises suivantes : <color=#5a77f4ff> et </color>. Le principe est celui du langage HTML. Noter que la définition du style peut présenter tout autant de balises que souhaitées (voir la documentation du *package* pour en connaître la liste).

Pour qu'un texte de réplique présente un mot en bleu, on écrira par exemple : « J'aime le <color=#5a77f4ff>bleu</color>. » *TextMeshPro* interprétera la chaîne pour afficher le texte comme il convient.

Noter que le style **Normal** ne présente pas de balisage afin de prendre les valeurs par défaut du composant *TextMeshPro* utilisé dans le *Canvas*, ceci afin de laisser la liberté de colorer un texte *via* le composant.

Pour plus d'informations sur la feuille de styles *TextMeshPro*, consulter la documentation officielle.

4. Gestionnaire



Pour traiter les dialogues *in-game*, il faut instancier dans la scène le *prefab* **InGameDialogueManager.** Une telle instance est nécessaire pour que le système de dialogue fonctionne.

Ce *prefab* contient le composant **InGameDialogueManager** servant à la gestion des dialogues et son arborescence présente un *Canvas* comprenant tous les objets de l'interface graphique.

Une fois dans la scène, il doit être accompagné d'un *EventSystem*. Si ce dernier n'est pas présent, penser à l'ajouter ainsi : **clic droit dans *Hierarchy* > UI > Event System**.

Le composant **InGameDialogueManager** est un **Singleton**, c'est-à-dire que l'ajouter à la scène fait ainsi de cette instance la référence unique, ce qui convient aux besoins du système de dialogue *in-game*.

Le composant présente un certain nombre de champs **déjà renseignés**. Ces champs déjà renseignés pointent vers des **références déjà disponibles** par le *Prefab* en tant qu'*Asset*. Il n'est pas nécessaire de les changer.

Sous l'étiquette **Tracked fields**, apparaissent des champs non éditables à des fins de suivi et *debug*. Les valeurs de ces champs changent selon qu'un dialogue a démarré ou non et selon le contenu du dialogue.

InGameDialogueManager fournit des **méthodes publiques** utilisables à tout moment. On appelle ces méthodes sur le modèle : **InGameDialogueManager.Instance.NomMéthode(paramètres)**. Voyons ces fonctions.

StartDialogue() est appelée pour qu'**InGameDialogueManager** lance la lecture d'un dialogue à l'écran. Elle requiert de lui passer un fichier de dialogue en paramètres.

StopCurrentDialogue() est appelée pour **interrompre** un dialogue qui est déjà en cours d'affichage. Cette fonction est pensée afin que le développeur puisse librement décider si le dialogue doit s'arrêter ou non selon ses besoins.

Enfin, la propriété **IsDialogueStarted**, publique en lecture seule, permet de connaître l'état du dialogue : est-il lancé ou non ? Cette propriété retourne une valeur booléenne.

5. Déclencher un dialogue

Contrairement au **Dialogue System**, le présent programme ne propose **pas de déclencheur**. En effet, les situations pour déclencher un dialogue *in-game* sont bien plus variées et nombreuses que pour lancer la boîte de dialogue et il serait impossible d'en dresser une liste exhaustive. Par conséquent, le développeur codera les conditions et utilisera les fonctions publiques du gestionnaire **InGameDialogueManager** selon ses besoins et les spécificités de chaque situation.

6. Rapport à Dialogue System

Un autre type de dialogue est créé pour le jeu : le **Dialogue System** prenant en charge la boîte de dialogue. Les deux systèmes sont en concurrence bien que leur objectif soit différent.

En effet, si une boîte de dialogue est affichée et présente déjà du contenu de dialogue, alors il convient de ne pas superposer de texte qui viendrait perturber la compréhension de l'écran et nuire à l'expérience.

Par conséquent, **DialogueManager** teste si **InGameDialogueManager** a un dialogue lancé et, si c'est le cas, stoppe ce dialogue pour lancer le sien. Ce choix est justifié par l'importance des boîtes de dialogue sur les événements textuels *in-game*.

Inversement, un dialogue en jeu ne se lance pas tant que la boîte de dialogue est encore à l'écran.

Ainsi, il n'y a pas de superposition possible entre les deux systèmes.