

Seront décrites dans ce document les étapes suivies pour arriver au projet VirtualESIEA résultant sur la branche vr_projet sur GitHub. A noter que les phases de tests sur les branches feature respectives de chaque membre ne seront pas détaillées.

Virtual'ESIEA

Partie VR

Documentation

Matthieu COQUELIN, Paul-Amaury KLEIN,
Stella THAMMAVONG

Table des matières

I) Configuration de la scène.....	2
1) Mise en place de la scène et du player.....	2
2) Ajout d'un pointeur au player	2
3) Création d'un timer.....	3
II) Déplacements	4
1) Téléportation	4
2) Création de marqueurs	6
III) Interactions.....	7
1) Clic sur le bouton d'un canvas.....	7
2) Interagir avec un bouton sur un canvas	8
3) Vidéos.....	10
a) Création de « l'écran » où la vidéo sera lue.....	10
b) Contrôle de la lecture vidéo	13
4) Textes	16
5) Sons	18

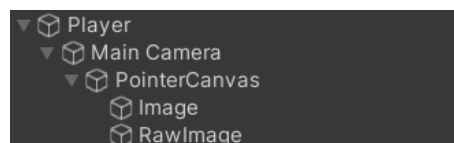
I) Configuration de la scène

1) Mise en place de la scène et du player

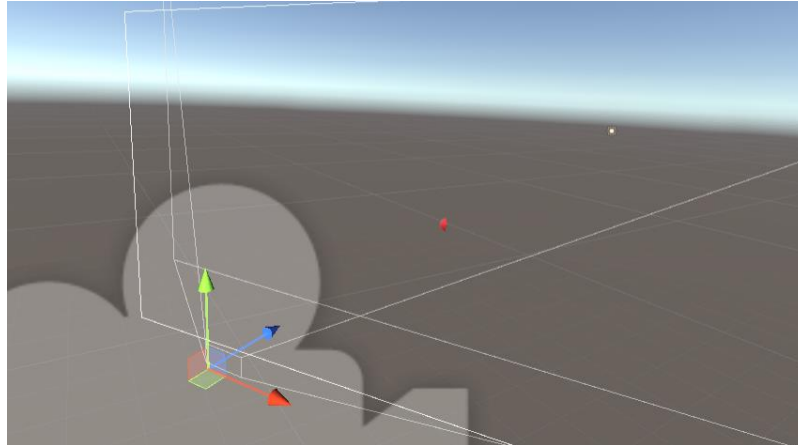
- Installez votre projet avec le tutoriel de Google : <https://developers.google.com/cardboard/develop/unity/quickstart>
- Supprimez la caméra dans votre scène
- Copiez le player du sample et collez le dans votre nouvelle scène (la nouvelle caméra contiendra un script CameraPointer qui appelle les méthodes OnPointerenter() et OnPointerExit() qui seront utilisées dans tout le projet).
- Ajoutez le script CardboardStartup à un objet fixe (par sa durée de vie) telle que la map.

2) Ajout d'un pointeur au player

- Créez sur Paint 3D (ou équivalent) un cercle de 16x16px plein qui servira de pointeur d'une part de moyen de visualiser le timer (que l'on créera par la suite) d'autre part. Sauvegardez-le et importez-le sous Unity.
- Changez le type de la texture précédemment importée en « Sprite (2D and UI) ».
- Créez un canvas, que l'on mettra en enfant de la Main Camera, qui contiendra une image et une rawImage (on y reviendra par la suite) de sorte à avoir cette arborescence d'éléments pour le player :



- Supprimez le Component « Graphic Raycaster » du canvas (pour que ce dernier ne gêne pas le script « CameraPointer » de la caméra)
- Passez le canvas en « World Space » et attribuez-lui la Main Camera
- Donnez la position suivante au canvas (0, 0, 1) de sorte à que ce dernier soit centré sur le regard de l'utilisateur (caméra).
- Pour l'image, son image source sera le sprite précédemment créé, son type sera « filled » et assurez-vous que les dimensions sont bien de 16x16px. Elle servira à visualiser le chargement du timer.
- Pour la rawImage, donnez-lui la texture correspondant au Sprite de l'image, de la même façon redimensionnez la texture en 16x16px et modifiez son scale de la sorte : (0.2540115, 0.2540115, 0.2540115). Elle servira de pointeur.



3) Création d'un timer

- Ajoutez un script à la caméra (ou autre gameObject) que l'on appellera « Timer ». Ouvrez-le puis remplacez les lignes existantes par les suivantes :

```
using UnityEngine;
using UnityEngine.UI;

public class Timer : MonoBehaviour
{
    ///<summary>
    /// Gaze filling image
    ///</summary>
    [SerializeField]
    private Image m_timerCharger;

    ///<summary>
    /// Timer duration in second
    ///</summary>
    [SerializeField]
    private float m_activeDuration;
    private float m_remainingTime;
    private bool m_activeTimer;

    private void Start()
    {
        m_timerCharger.fillAmount = 0.0f;
        m_remainingTime = 0.0f;
    }

    ///<summary>
    /// Check if timer is finished or not
    ///</summary>
    public bool IsActive
    {
        get { return m_remainingTime <= 0 ? true : false; }
    }

    ///<summary>
    /// Each frame implement the timer by decreasing m_activeDuration value
    ///</summary>
    void Update()
    {

```

```

    {
    if (m_remainingTime > 0 && m_activeTimer)
        {
            m_remainingTime -= Time.deltaTime;
            m_timerCharger.fillAmount = m_activeDuration - m_remainingTime;
        }
    else
        NullTimerCharger();
    }

    ///<summary>
    /// Activate the timer
    ///</summary>
    publicvoid Activate()
    {
        m_activeTimer = true;
        m_remainingTime = m_activeDuration;
    }

    ///<summary>
    /// Put the fill image to 0
    ///</summary>
    publicvoid NullTimerCharger()
    {
        m_activeTimer = false;
        m_timerCharger.fillAmount = 0.0f;
    }
}

```

Le but de ce script est d'activer un timer d'une certaine durée que l'on pourra aisément modifier et de contrôler le sprite qui suit notre regard afin de connaître l'état du chargement du timer. A chaque action (exemple : téléportation par marqueur) on utilisera le timer.

- Sous Unity pensez bien à modifier la durée du timer et de lui passer l'image du sprite.



II) Déplacements

Le but sera de placer un script de déplacement sur des marqueurs assez « plats » en hauteur et assez petit (pas trop pour qu'il soit visible) pour pouvoir se téléporter dessus.

1) Téléportation

- Créez un script « Teleportation » et remplacez les lignes existantes par les suivantes :

```

using UnityEngine;

publicclassTeleportation : MonoBehaviour
{
    ///<summary>
    /// Is use to know if we are looking an interactable gameObject
    ///</summary>

```

```

private bool m_watch;

///

```

```

        m_timer.NullTimerCharger();
        m_actualPosition = m_player.position;
        TeleportPlayer(m_watch);
    }

    ///<summary>
    /// Define if the new position will be the same or not
    ///</summary>
    ///<paramname="gazedAt"></param>
    private void TeleportPlayer(bool gazedAt)
    {
        m_player.position = gazedAt ? m_newPosition : m_actualPosition;
    }
}

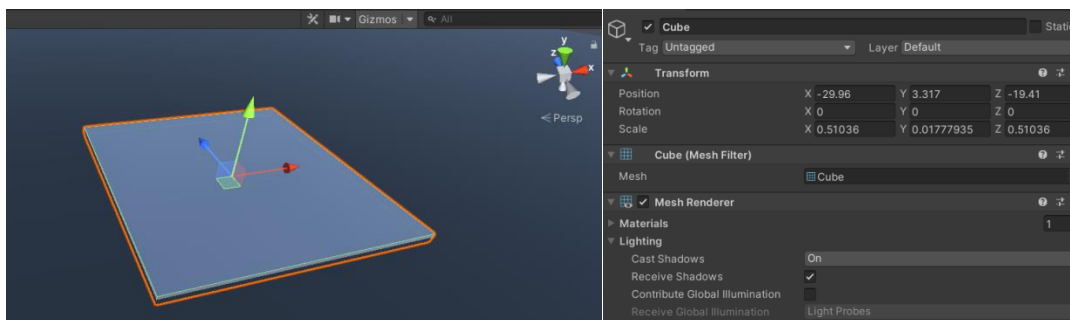
```

Les méthodes `OnPointerEnter()` et `OnPointerExit()` sont en permanence appelées par le script situé sur la caméra pour savoir si on regarde ou non le marqueur de notre script. L'état du pointeur et la position de l'utilisateur (`m_player`) sont ajustés en fonction de si on regarde ou non le marqueur.

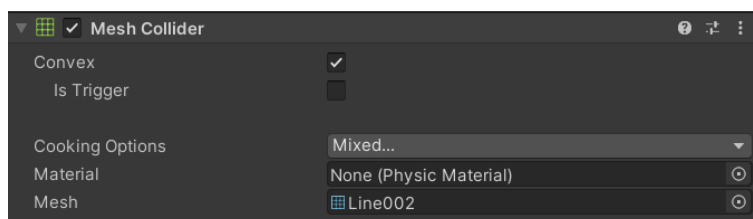
2) Création de marqueurs

Cette étape explique comment créer des marqueurs mais peut être passée si vous comptez utiliser les marqueurs déjà présents dans le projet nommés Footmarks.

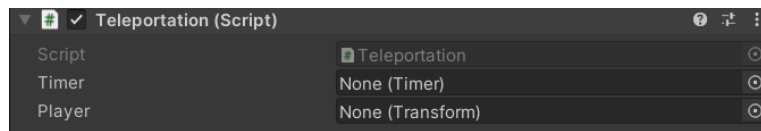
- Créez un `gameObject` plat (cube ou autre), diminuez son scale en Y de sorte à ce qu'il ne fasse que quelques cm et placez son centre au niveau du sol (il faut garder en tête que c'est par rapport au centre de ce `gameObjet`, que l'utilisateur se téléportera) Pour que le marqueur soit suffisamment visible il faudrait qu'en X et en Z il fasse 50 cm



- Assurez-vous que le `gameObject` ait un collider (box collider ou autre) dans ses composants sinon rajoutez-lui un mesh collider et cochez la case `convex`.



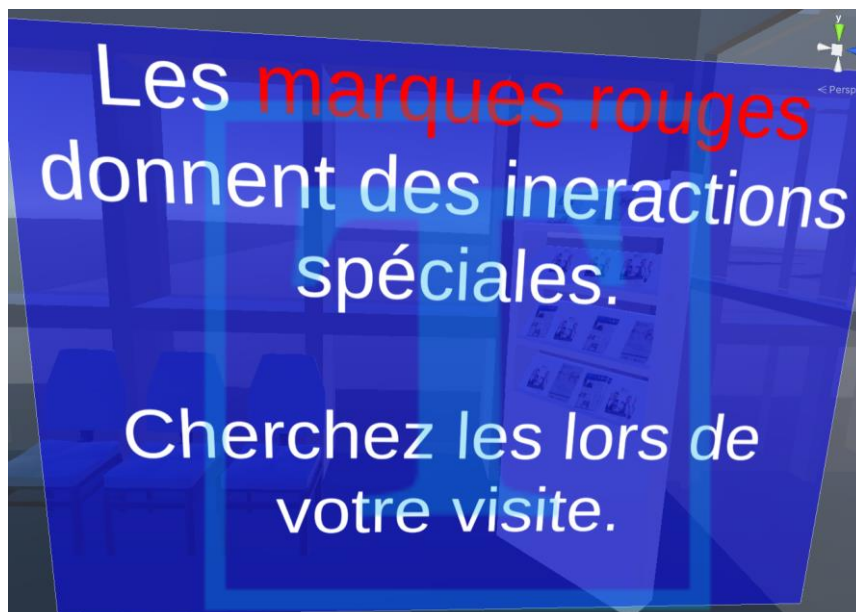
- Rajoutez à ce `gameObject` le script `Teleportation`.
- Créez un prefab à partir de ce `gameObject`.
- Vous pouvez désormais placer ce prefab un peu partout dans votre scène pour vous déplacer (ne pas oublier de passer le player et le timer au script).



Dans le projet, certains marqueurs ont une couleur différente (rouge) ils permettent de prévenir l'utilisateur qu'une action va arriver ici.



Un texte a été placé pour le prévenir en fonction :



III) Interactions

1) Clic sur le bouton d'un canvas

Comme pour le script de téléportation on retrouve les méthodes `OnPointerEnter()` et `OnPointerExit()` qui serviront à activer l'image de notre timer et à invoquer une méthode par clic sur bouton (avec un `UnityEvent`).

- Créez un script « `ButtonInteractor` » et remplacez son contenu par les lignes suivantes :

```
using UnityEngine;
```



```

using UnityEngine.Events;

public class ButtonInteractor : MonoBehaviour
{
    ///<summary>
    /// Is use to know if we are looking an interactable gameObject
    ///</summary>
    private bool m_watch;
    [SerializeField]
    private Timer m_timer;
    ///<summary>
    /// Call a public method by interacting with the button
    ///</summary>
    [SerializeField]
    private UnityEvent click;

    void Start()
    {
        m_watch = false;
    }

    ///<summary>
    /// If timer is finished and user is looking at a button a method is called by
    /// UnityEvent
    ///</summary>
    void Update()
    {
        if (m_timer.IsActive && m_watch == true)
            click.Invoke();
    }

    ///<summary>
    /// Called by cameraPointer script to know if we are looking at a button
    ///</summary>
    public void OnPointerEnter()
    {
        m_watch = true;
        m_timer.Activate();
    }

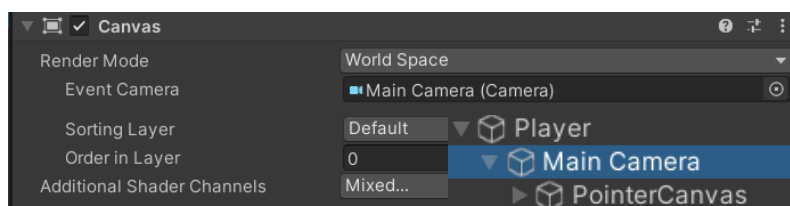
    ///<summary>
    /// Called by cameraPointer script to know if we are not looking at a button
    ///</summary>
    public void OnPointerExit()
    {
        m_watch = false;
        m_timer.NullTimerCharger();
    }
}

```

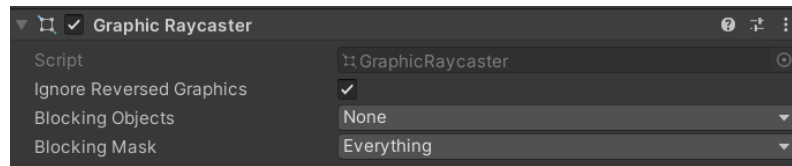
Un 2ème script pourra être créé en parallèle pour contenir des méthodes « public void » qui seront appelées lorsque le timer a fini de se charger sur le bouton.

2) Interagir avec un bouton sur un canvas

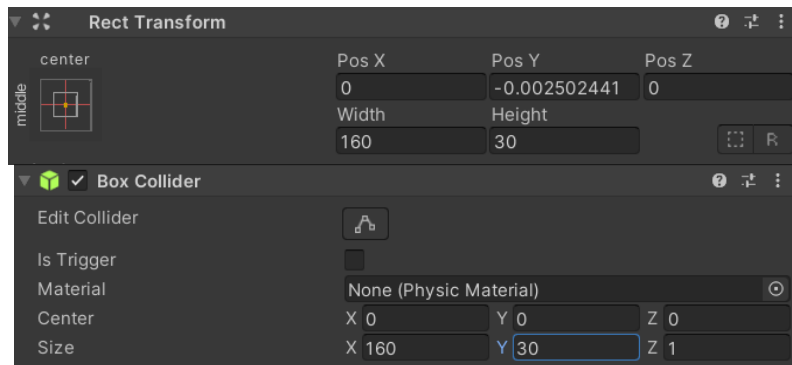
- Créez un canvas, mettez son Render Mode en World Space et passez-lui la caméra



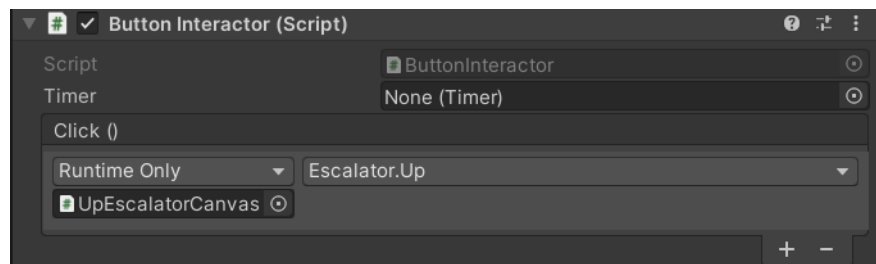
- Changez, au besoin, sa position et sa taille comme vous le souhaitez.
- Assurez-vous que le canvas ait bien un Graphic Raycaster component.



- Ajoutez un bouton et donnez-lui un box collider qui devra faire la taille du bouton (donnez les dimensions du bouton au box collider).



- Ajoutez-lui le script « ButtonInteractor » et définissez la méthode qui sera appelée lorsqu'on regardera le bouton (fin du timer)



Voici un exemple de script qui a été créé à part et mis sur le canvas du bouton

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Escalator : MonoBehaviour
{
    [SerializeField]
    private Transform m_player;

    [SerializeField]
    private Transform m_upFootMarks;

    [SerializeField]
    private Transform m_downFootMarks;

    // Update is called once per frame
    void Update()
    {

```

```

    }

    public void Up()
    {
        m_player.position = new Vector3(m_upFootMarks.position.x,
        m_upFootMarks.position.y + 1.6f, m_upFootMarks.position.z);
    }

    public void Down()
    {
        m_player.position = new Vector3(m_downFootMarks.position.x,
        m_downFootMarks.position.y + 1.6f, m_downFootMarks.position.z);
    }
}

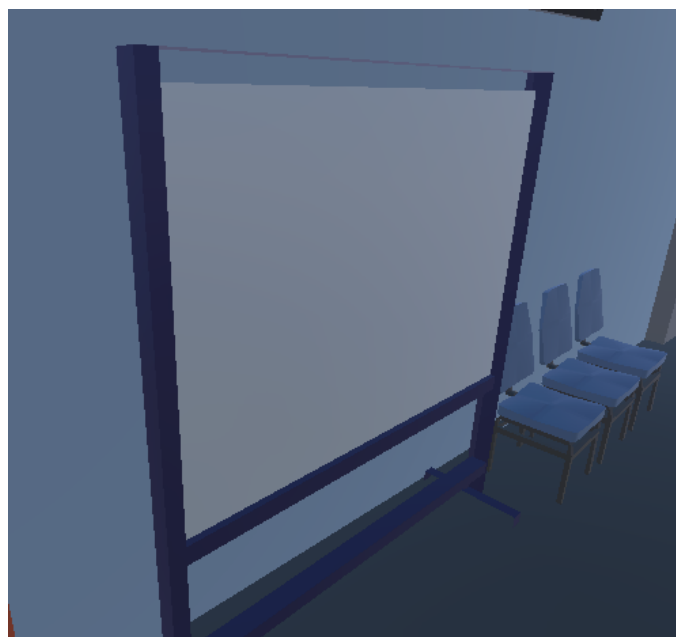
```

Ainsi il est possible de créer une méthode en « public void » qui sera appelée dès que l'on appuie sur le bouton.

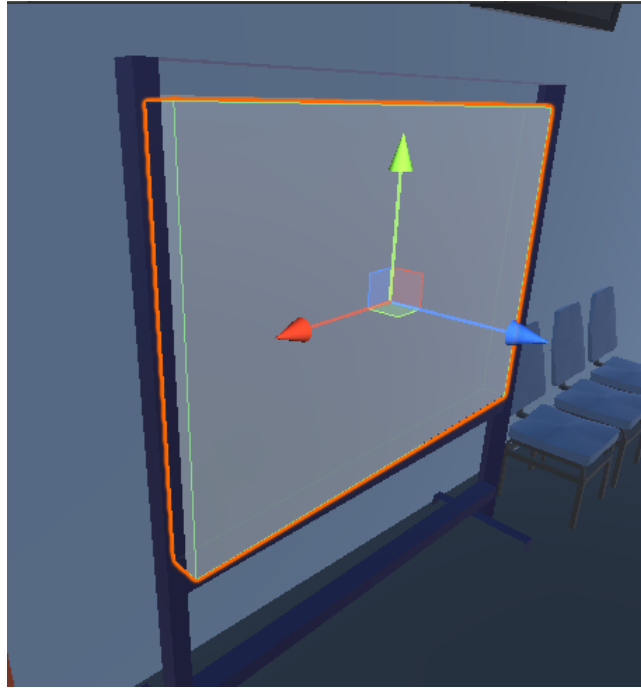
3) Vidéos

a) Création de « l'écran » où la vidéo sera lue

- Créez un cube (3D Object) et redimensionnez-le de la bonne taille. Placez-le à la bonne position, sur un écran virtuel de votre scène si vous le souhaitez. Veillez à ce que l'épaisseur ne soit pas trop petite car cela peut poser des problèmes pour lire la vidéo.

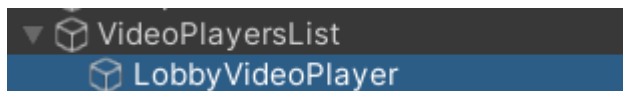


Environnement sans notre cube faisant office « d'écran »



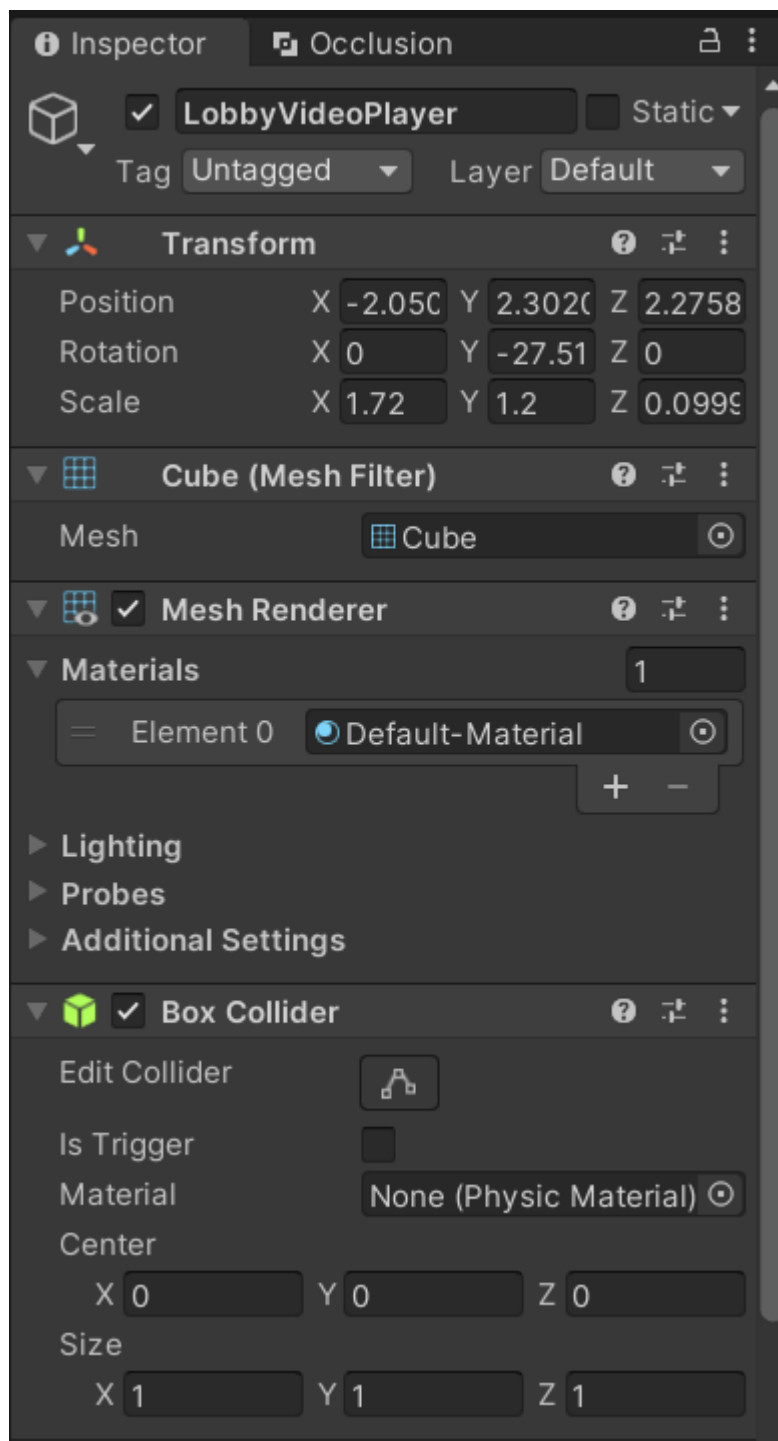
Avec notre cube ajouté par-dessus

- Donnez-lui un nom adéquat, pour cet exemple il s'appelle LobbyVideoPlayer

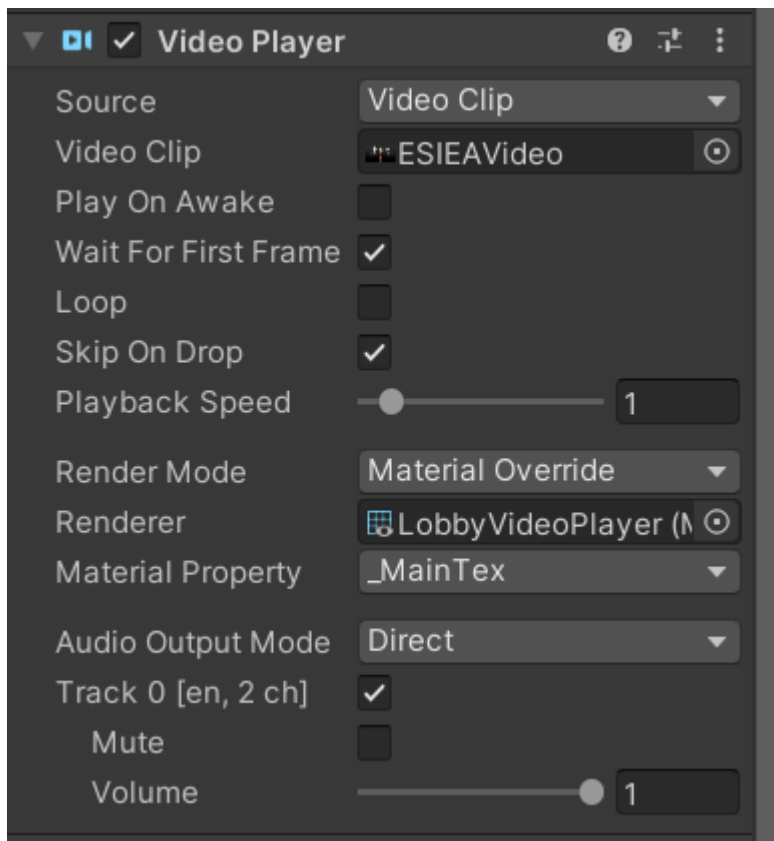


Vous pouvez organiser tous vos « écrans » dans une liste dans l'arborescence de la scène.

Votre LobbyVideoPlayer devrait ressembler pour l'instant au Screenshot suivant avec votre Transform adapté à votre scène :



- Ajoutez-lui ensuite un component VideoPlayer



- Sélectionnez Video Clip comme source après avoir ajouté la vidéo de votre choix dans les assets du projet. Sélectionnez la vidéo dans Video Clip.
- Décochez Play On Awake pour contrôler la lecture de la vidéo depuis un script, et donc qu'elle ne se lance pas toute seule dès le début.
- Cochez Loop si la vidéo doit être lue en boucle (mais peut être arrêtée depuis un script)
- Laissez les autres paramètres par défaut tels qu'ils sont.

b) Contrôle de la lecture vidéo

Pour contrôler la lecture de cette vidéo et de ce lecteur vidéo, créez un script VideoPlayerScript avec le contenu suivant :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Video;

public class VideoPlayerScript : MonoBehaviour
{
    ///<summary>
    /// //Script to manage a video player component from a footmark
    ///</summary>

    //Video Screen which contains the VideoPlayer object to manage
    [SerializeField]
    private GameObject videoScreen = null;

    //Transform of the player of the scene
    [SerializeField]
    private Transform m_player = null;
```

```

//VideoPlayer object to manage
private VideoPlayer videoPlayer = null;

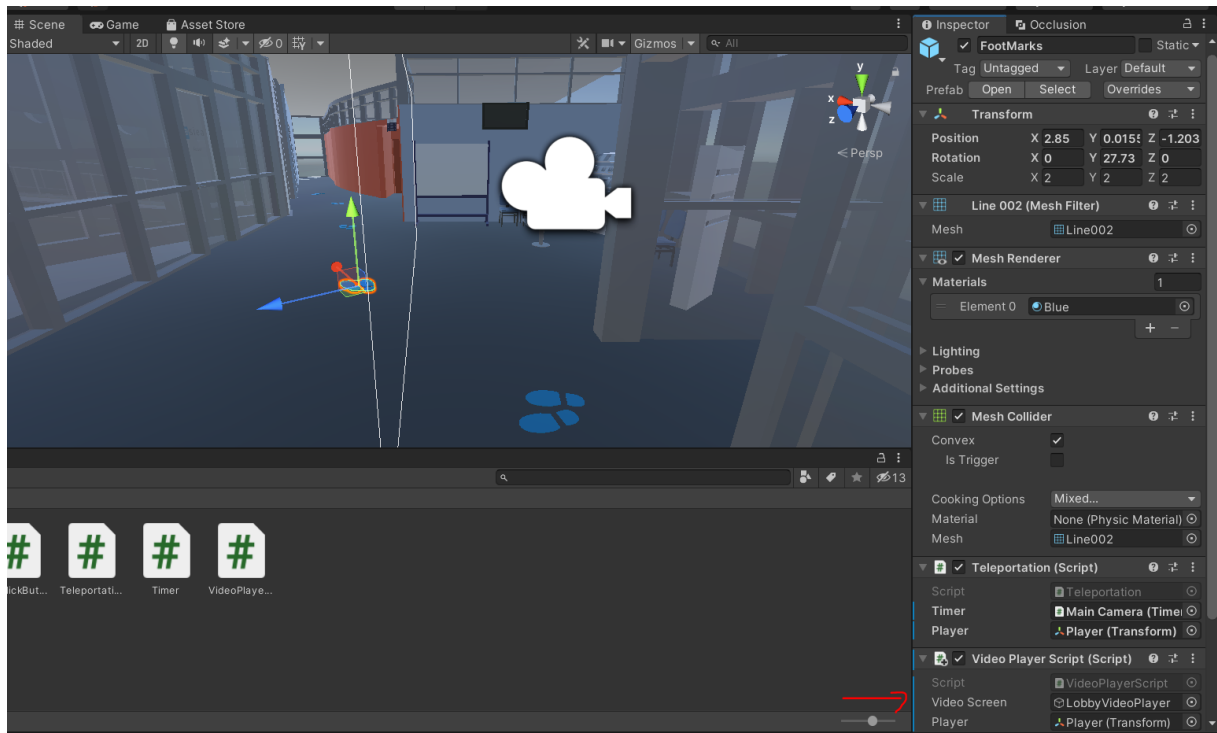
voidStart()
{
    videoPlayer = videoScreen.GetComponent<UnityEngine.Video.VideoPlayer>();
}

///

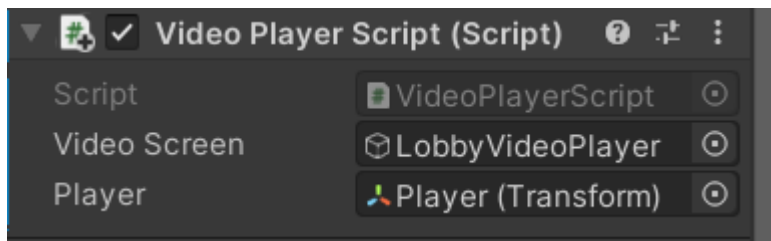
```

Ici, le script a été écrit pour être compatible avec notre système de déplacement sur marqueurs par téléportation, mais il peut être adapté pour que les fonctions stopVideo() et startVideo() soient appelées d'autres façons.

- Ajoutez le script en component du marqueur que vous voulez utiliser pour lancer la vidéo



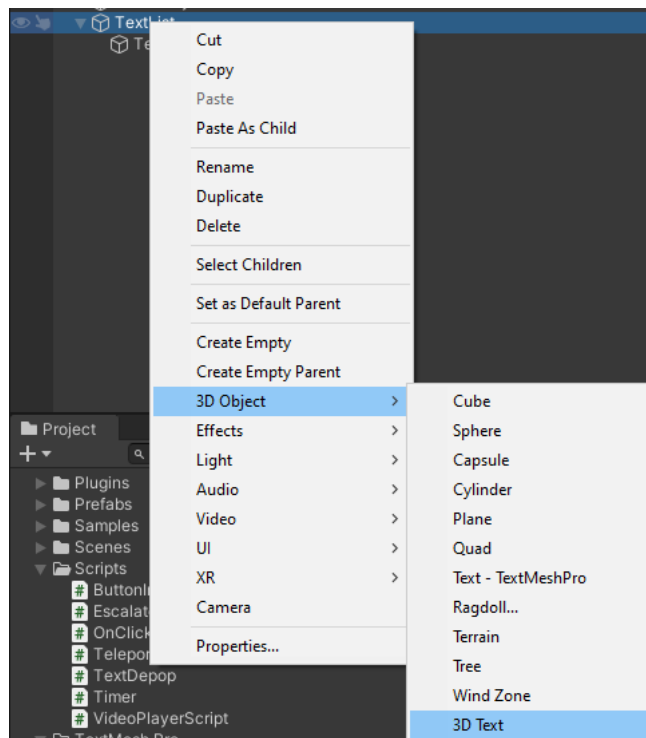
- Après avoir ajouté le script, sélectionnez l'écran créé à l'étape 1 pour le champ VideoScreen et le Player de la scène dans le champ correspondant.



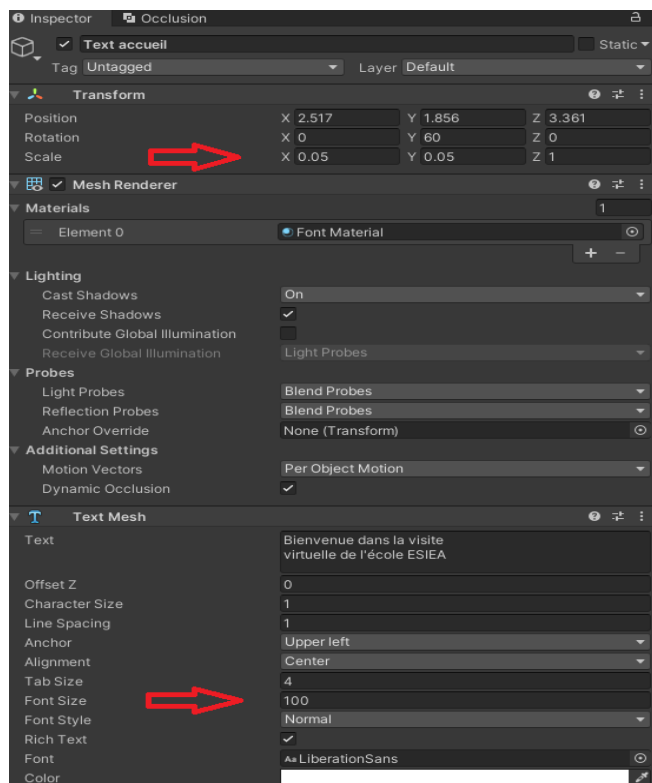
Une fois le projet compilé et lancé, la vidéo se lancera quand le player se rendra sur le marqueur où le script a été ajouté et s'arrêtera lorsqu'on la quittera.

4) Textes

Pour ajouter du texte, créer un 3D Object et cliquer sur 3D Text.



Le texte qui apparaîtra sera un peu flou. Pour l'enlever, il faut augmenter le « Font size » et diminuer le « Scale ». Voici un exemple pour ce qui concerne le message d'accueil :



Vous pouvez bien-sûr modifier la position du texte soit avec les flèches représentant les 3 axes principaux dans la fenêtre Scene, ou alors par le biais de la fenêtre « Inspector » du texte et de modifier la position X, Y et Z.

(Facultatif) Si vous souhaitez faire disparaître votre texte à partir d'un certain temps, créez un script en C# puis insérez ces lignes :

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class TextDepop : MonoBehaviour
6  {
7      public float time = 5; //Seconds to read the text
8
9      void Start()
10     {
11         Destroy(gameObject, time);
12     }
13     // Update is called once per frame
14     void Update()
15     {
16     }
17 }
18
19
```


Dans la fenêtre « Inspector » de votre texte, tout en bas de la fenêtre, appuyez sur « Add Component » -> « Scripts » -> (Le nom de votre script).

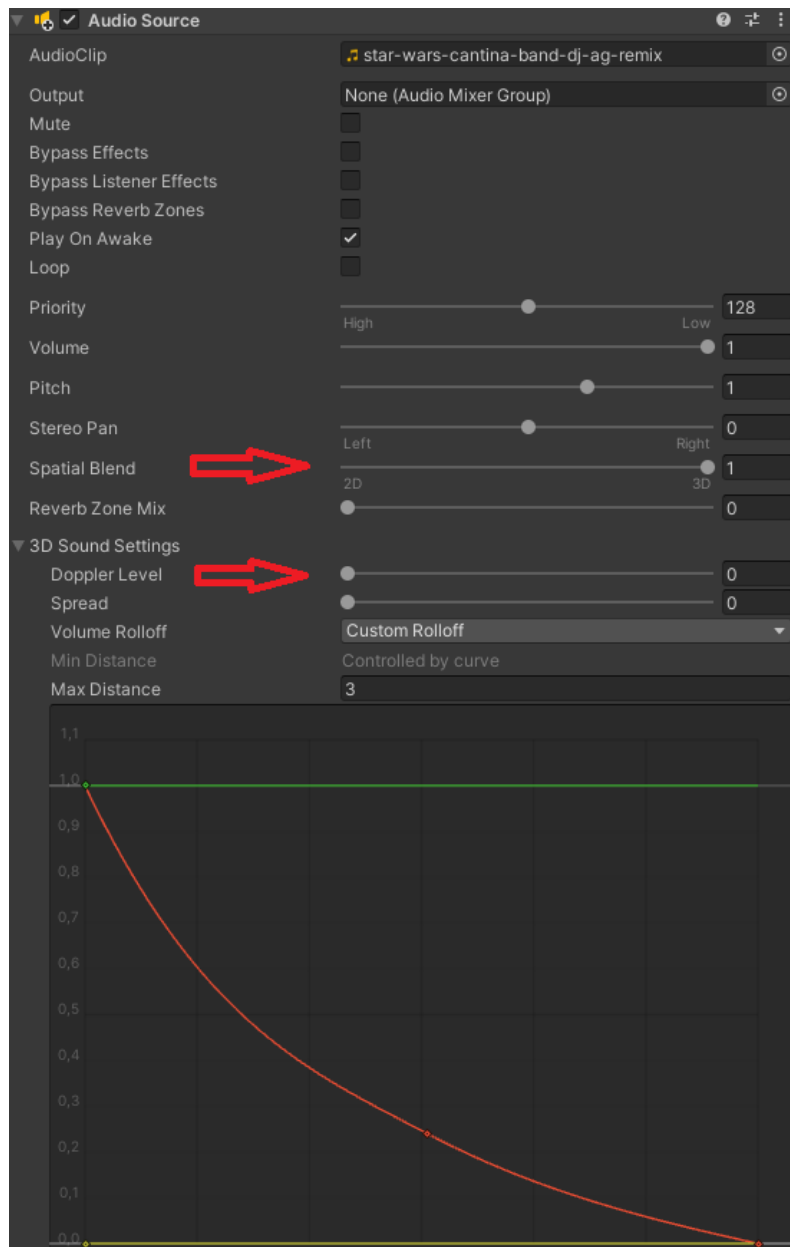
Ainsi, la fonction « Destroy » va agir sur le gameObject, à savoir votre texte, et le fera disparaître à partir d'un certain temps, ici 5 secondes.

5) Sons

Pour ajouter du son, prenez d'abord la musique ou le bruit que vous souhaitez et insérez-le dans les assets, afin qu'Unity puisse reconnaître le fichier.

Ensuite, sélectionnez un objet ou une forme, allez dans sa fenêtre « Inspector », puis tout en bas, cliquez sur « Add Component » -> « Audio » -> « Audio Source ».

Pour insérer la musique ou le bruit que vous avez ajouté précédemment, dans le component Audio Source, cliquez sur le symbole  dans la partie « Audio Clip » et choisissez votre son. Cependant, votre son sera entendu dans toute l'école. Pour que ce ne soit plus le cas, les 2 paramètres les plus importants à modifier sont le « Spatial Blend » qui faut mettre à 1 pour avoir un son en 3D, mais surtout, en déroulant la partie « 3D Sound Settings », il faut que le Doppler Level soit à 0, sinon vous arriverez quand même à entendre le son que vous avez inséré où que vous soyez.



A partir de là, vous pouvez modifier la distance maximale pour entendre le son, dont vous pouvez voir la limite dans la fenêtre « Scene ». Vous pouvez aussi modifier la courbe rouge, qui représente l'atténuation du volume du son par rapport à la distance.