Université de Technologie de Belfort-Montbéliard

MV54 Réalité Virtuelle et Augmentée

TP – Contrôleurs Oculus Touch dans OpenXR

Unity 2020.3 LTS

FISE Informatique

Sébastien CHEVRIAU
sebastien.chevriau@utbm.fr

Table des matières

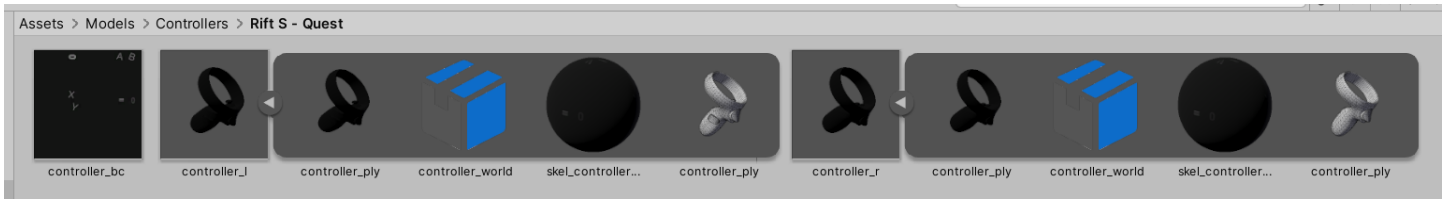
1. Introduction

L'objectif de document est d'apprendre à utiliser les contrôleurs Oculus Touch dans une configuration OpenXR.

2. Récupération des modèles 3D

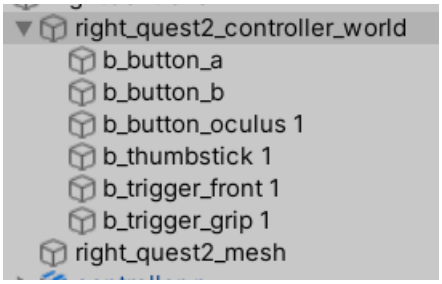
Les modèles 3D des contrôleurs « **Rift S / Quest 1** » et « **Quest 2** » sont disponibles sur le site dédié aux développeurs d'Oculus : [Oculus Developer Center | Downloads]
Vous les retrouverez également sur Moodle si besoin.

Pour le modèle « **Rift S / Quest 1** », vous trouverez un fichier FBX par contrôleur (gauche et droite) et un fichier texture.



Pour le modèle « **Quest 2** », vous trouverez 3 fichiers FBX contenant les 2 contrôleurs avec des maillages de qualités différentes. Ces FBX embarquent les textures nécessaires.

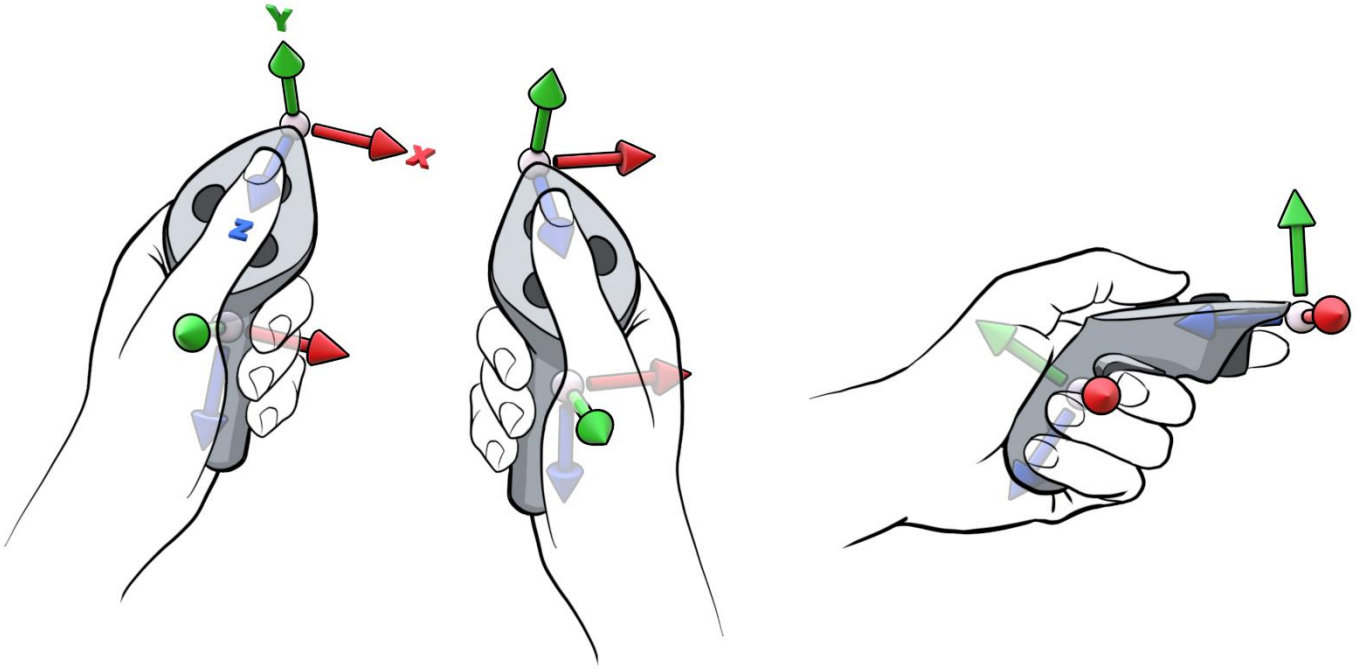
Ce sont, pour les 2 versions, des maillages déformables permettant d'animer les différents boutons et triggers.



3. Les référentiels dans OpenXR

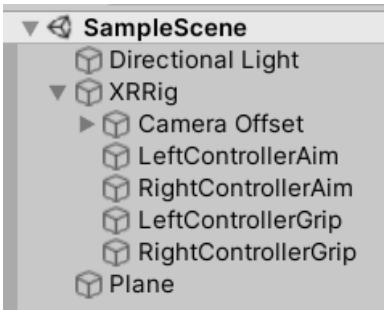
OpenXR implémente deux référentiels pour un contrôleur :

- Le « **Grip** » correspondant à une position à l'intérieur de la main et dans l'axe de préhension
- Le « **Aim** » correspondant à une position légèrement devant la manette et dans l'axe de pointage de celle-ci

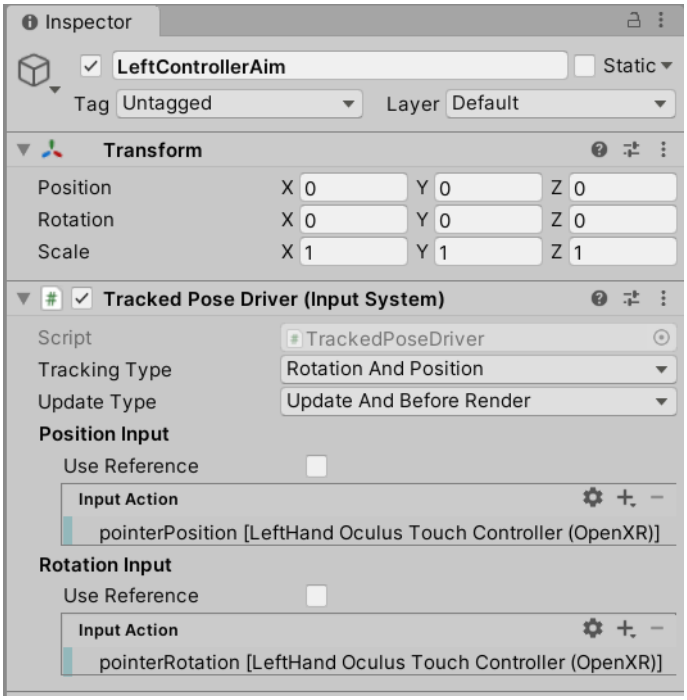


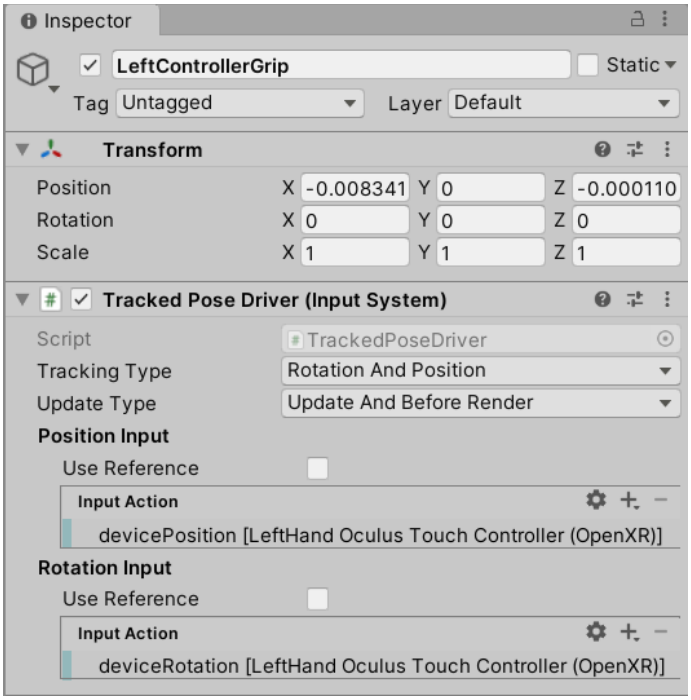
4. Configuration du suivi

Ajouter **4 GameObjects Empty** : 1 par contrôleur et référentiel. Les définir en tant qu'enfants du GameObject « **XRRig** ».



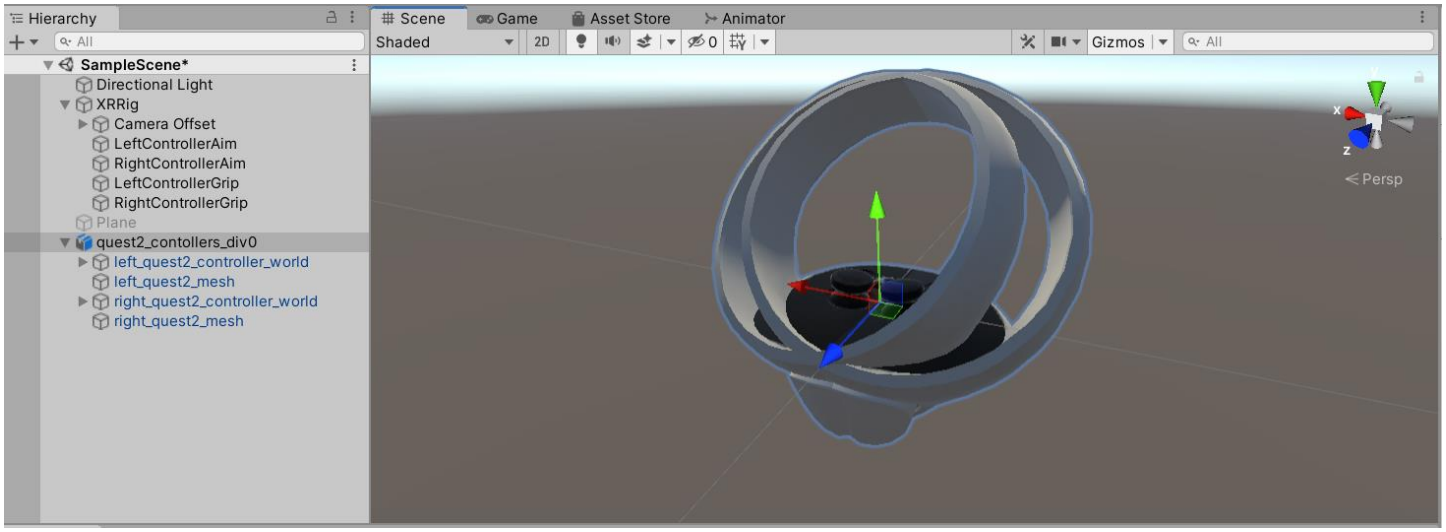
Ajouter un composant « **Tracked Pose Driver (Input System)** » à chacun. Définir les actions d'entrée associées



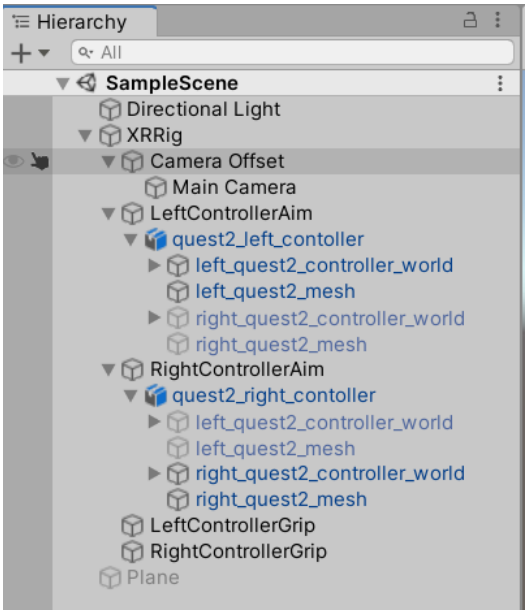


5. Placement des modèles 3D

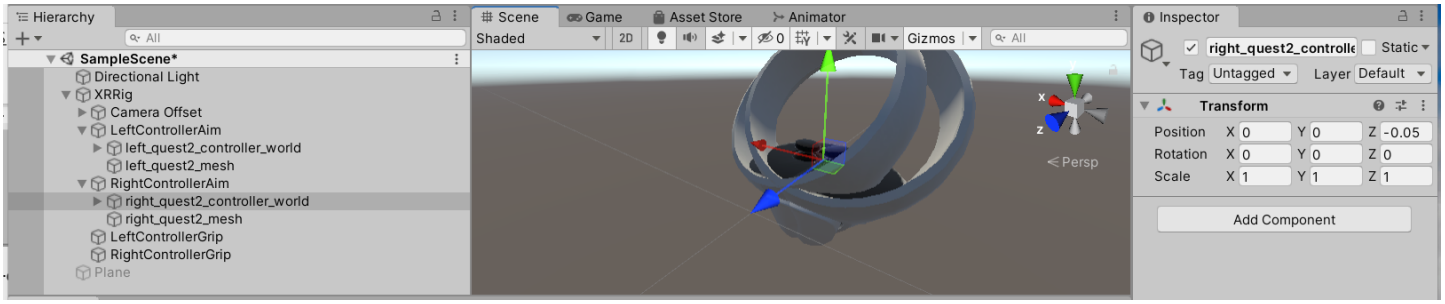
Le placement des modèles 3D des contrôleurs va dépendre du référentiel dans lequel ils ont été modélisés. Pour les 2 modèles vous remarquerez que leur référentiel se situe à peu près au centre des boutons pour le pouce. Il sera donc plus facile de les positionner sur le référentiel « **Aim** ».



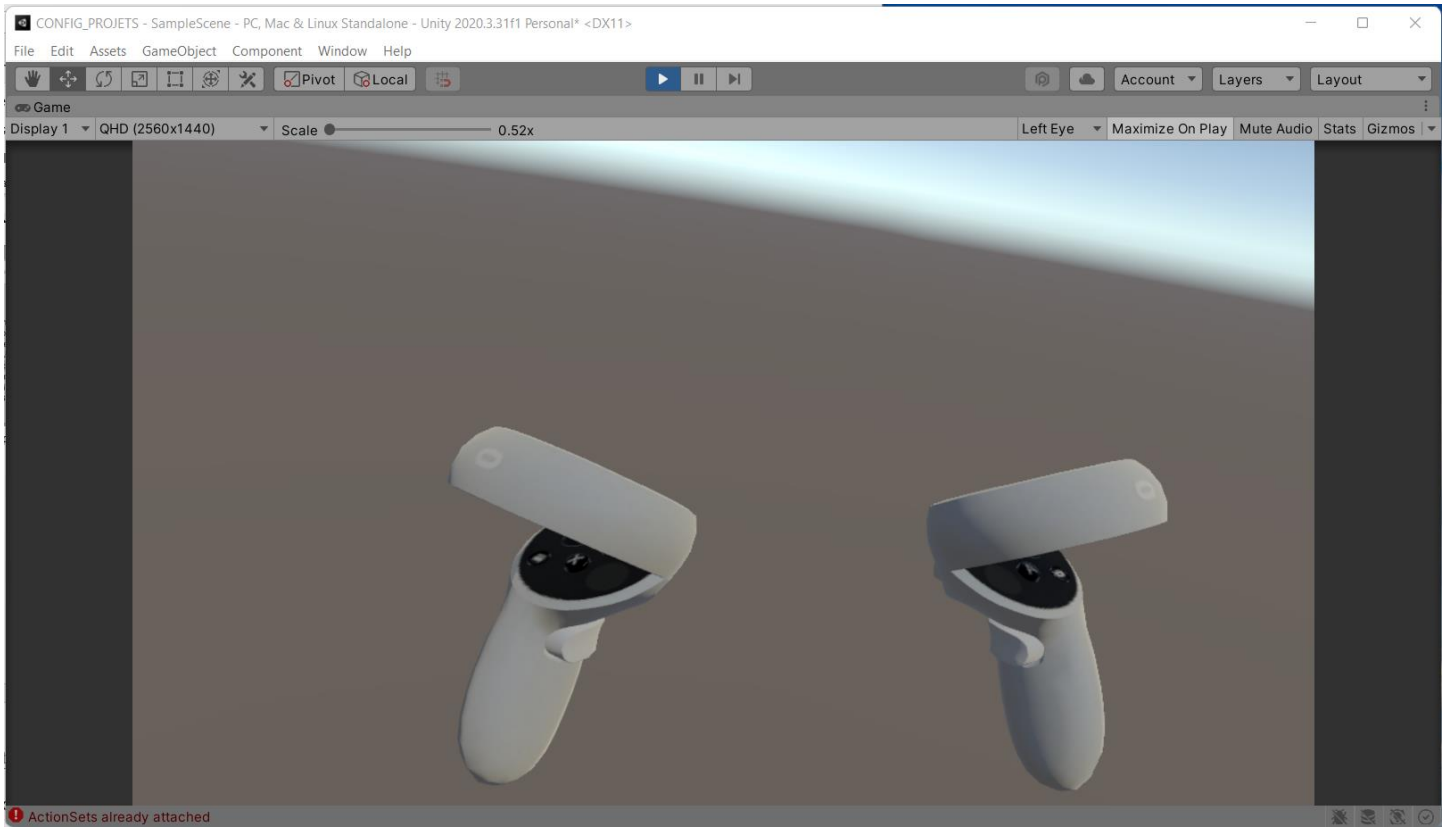
Déplacer chaque contrôleur en tant qu'enfant du référentiel Aim correspondant en position (0, 0, 0). Nous laissons la structure du FBX intacte pour faciliter l'animation des boutons plus tard.



Tester l'application. Vous remarquerez que le modèle virtuel ne semble pas au même endroit que le modèle réel. Cela est dû au fait que, comme dit précédemment, le référentiel Aim est placé **devant le contrôleur**. **Reculer** alors chaque modèle 3D de contrôleur **d'environ 5 cm** par rapport à son parent.



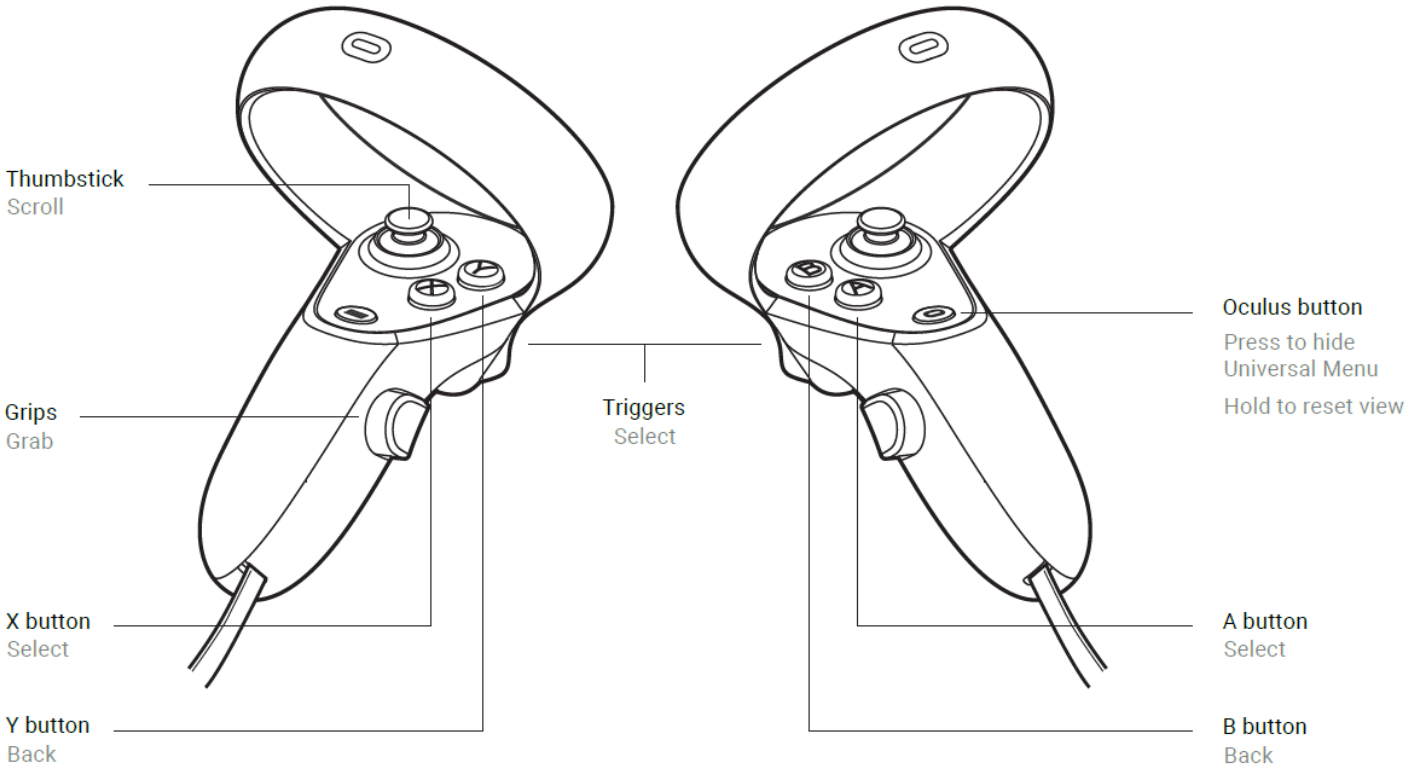
Tester votre application. Les contrôleurs virtuels devraient suivre les contrôleurs réels. Ajuster le décalage si besoin.



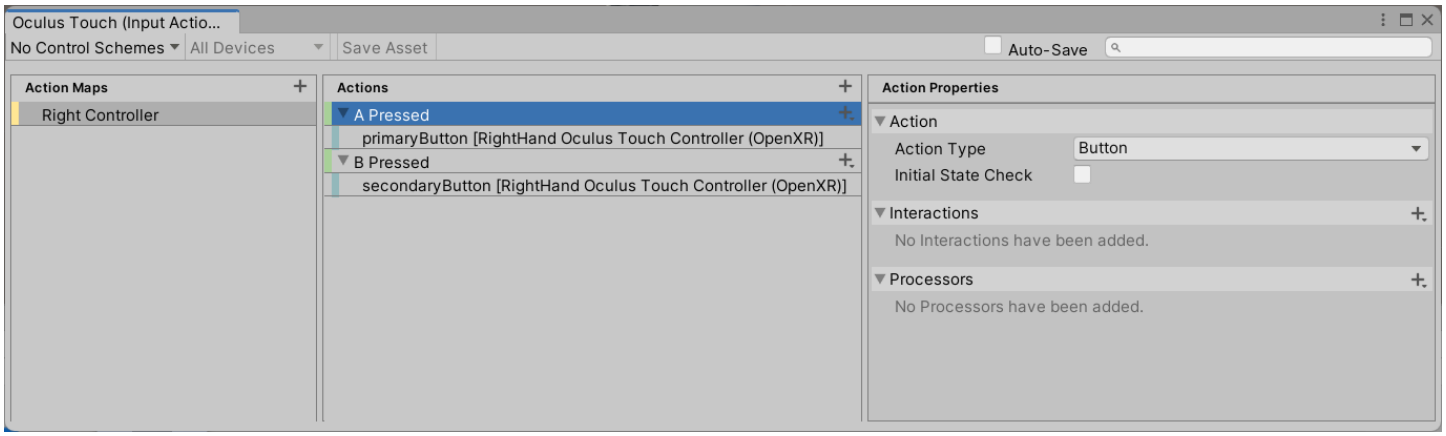
6. Récupérer les actions

Voici les différents types de données accessibles sur les contrôleurs Oculus Touch :

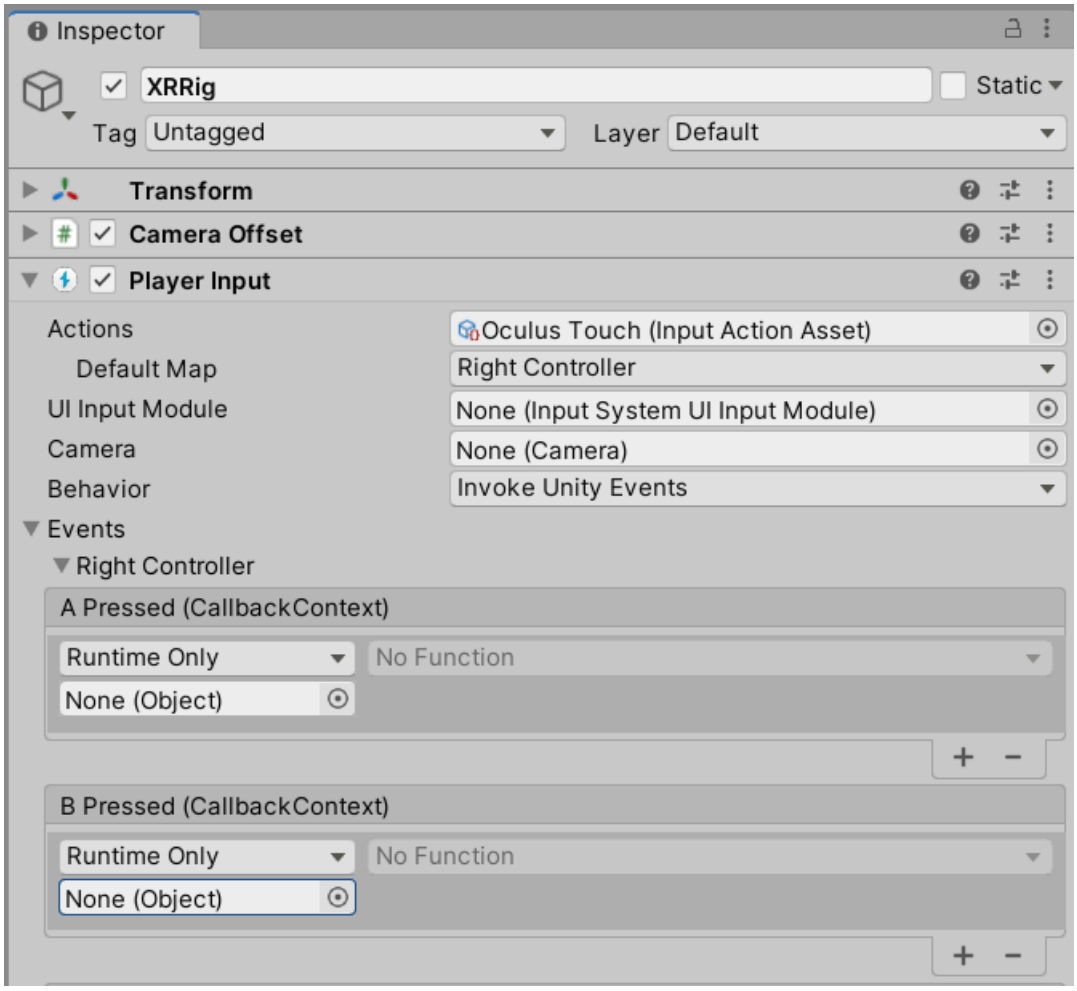
- Bouton (booléen) – réagissent lorsqu'ils sont pressés à fond :
  - A, B, X, Y
  - Menu
  - Thumbstick clicked
  - Triggers, Grips
- Touch (booléen) – réagissent lorsque l'on pose le doigt dessus :
  - A, B, X, Y
  - Thumbsticks
  - Triggers
- 2D Axis (Vector2) – de -1 à 1 :
  - Thumbsticks
- Analog (float) – de 0 à 1 :
  - Triggers
  - Grips



La captation des actions se fait via l'Input System. Créer un asset d'actions. Commençons par les 2 boutons de la manette droite (A et B).



Ajouter un composant « **Player Input** » au GameObject XRRig et choisir votre asset d'actions. Choisir un comportement « **Invoke Unity Events** ». Cela nous permettra d'associer des fonctions aux événements captés.

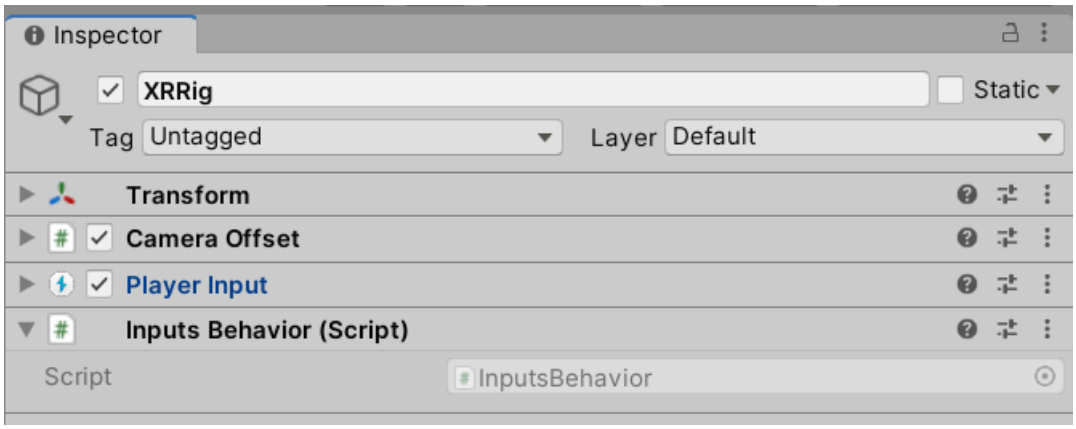


Créer un nouveau script pour gérer le traitement des actions.

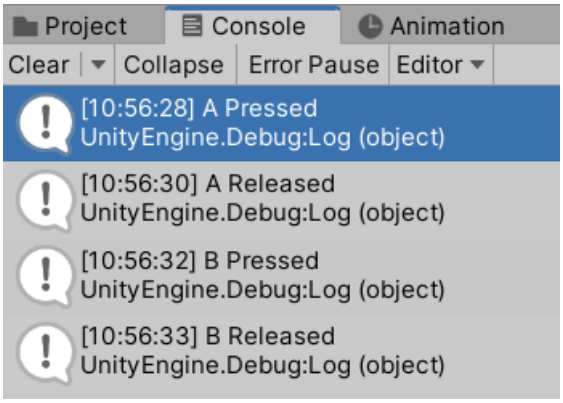
```
public class InputsBehavior : MonoBehaviour
{
    public void OnAPressed(InputAction.CallbackContext context)
    {
        if (context.started)
        {
            Debug.Log("A Pressed");
        }
        if (context.canceled)
        {
            Debug.Log("A Released");
        }
    }

    public void OnBPressed(InputAction.CallbackContext context)
    {
        if (context.started)
        {
            Debug.Log("B Pressed");
        }
        if (context.canceled)
        {
            Debug.Log("B Released");
        }
    }
}
```

Associer ce script au GameObject XRRig et associer chaque événement du composant Player Input

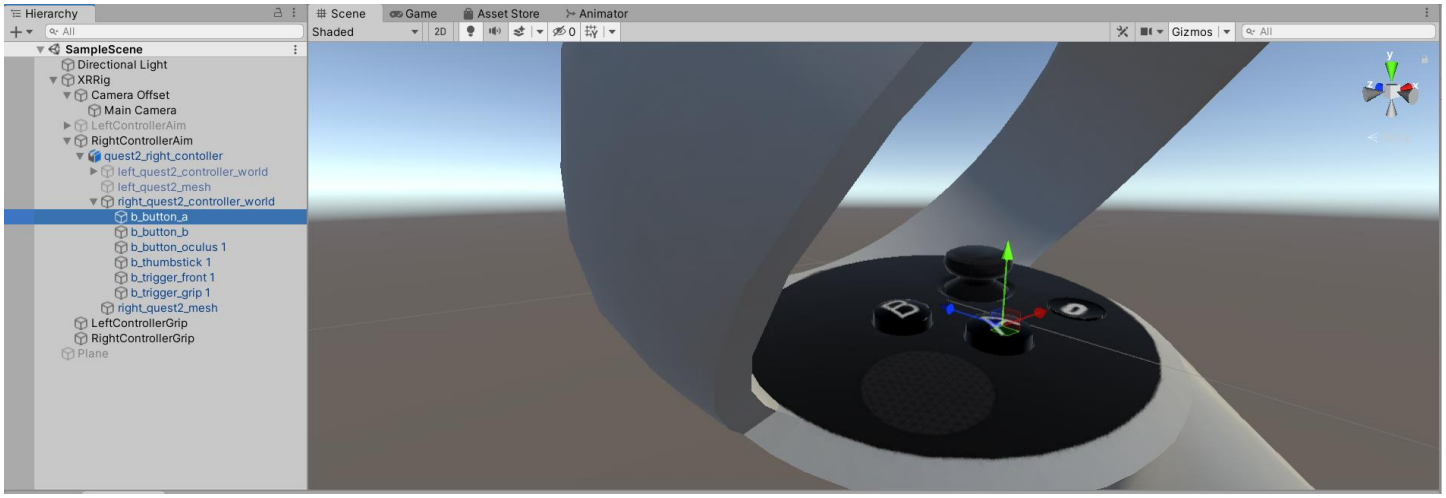


Tester. Vous devriez capter les événements Pressed et Released.

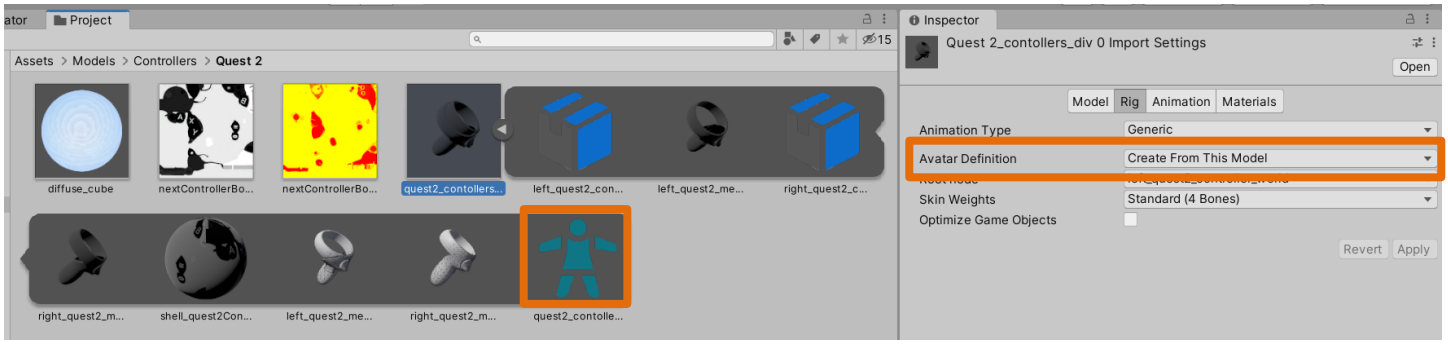


7. Animer les boutons des contrôleurs

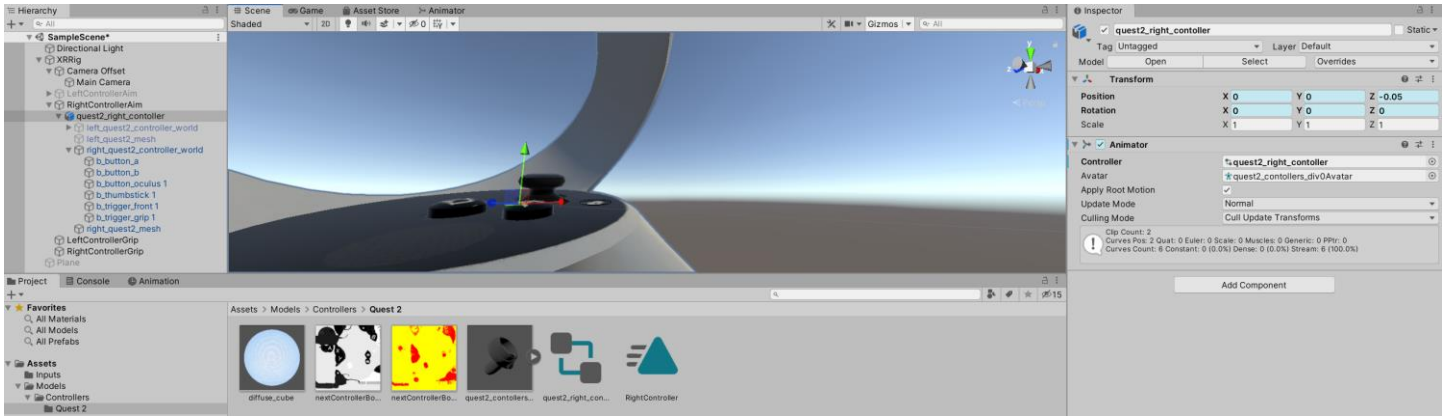
Les modèles fournis possèdent un maillage déformable au niveau des boutons et qui est géré par un squelette.



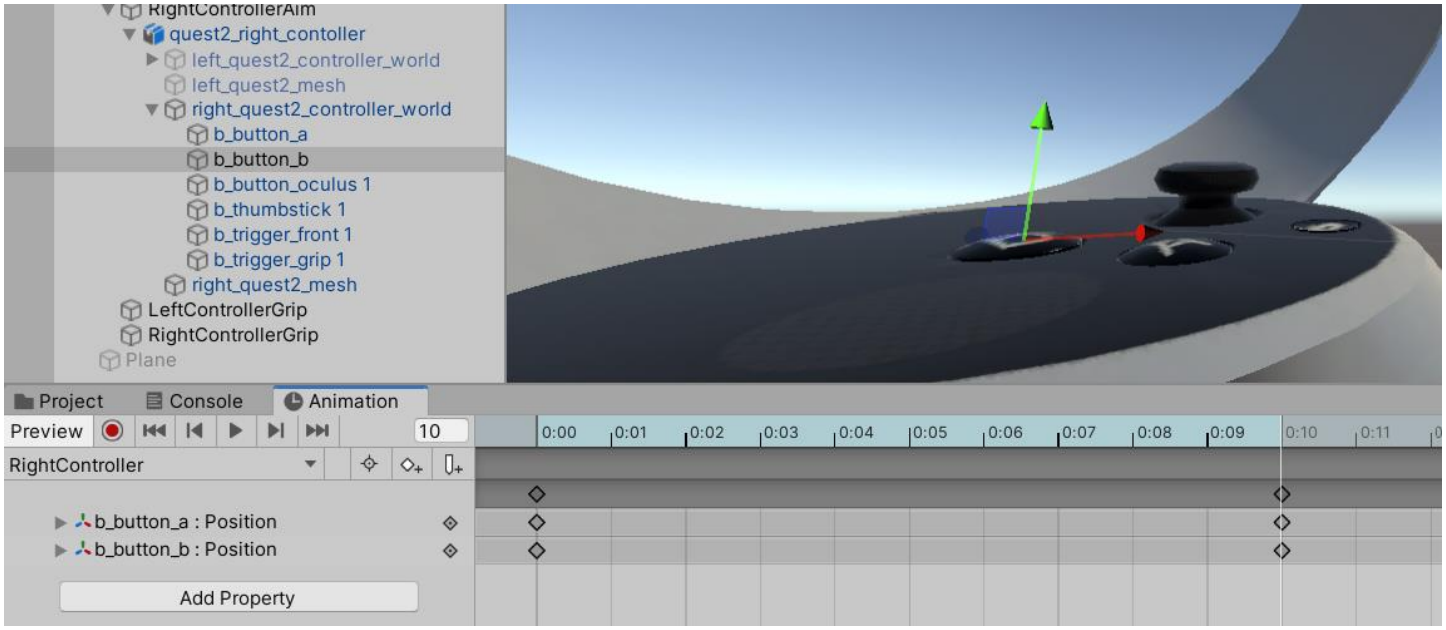
Voyons comment animer cela en fonction des actions. Il serait possible de faire un contrôleur d'animation et une animation par bouton, mais voyons comment optimiser cela. Nous n'allons créer qu'un seul contrôleur d'animation et qu'une seule animation pour l'ensemble, puis nous utiliserons les masques d'avatar pour filtrer chaque bouton.
Comment par ajouter un avatar depuis l'asset FBX.



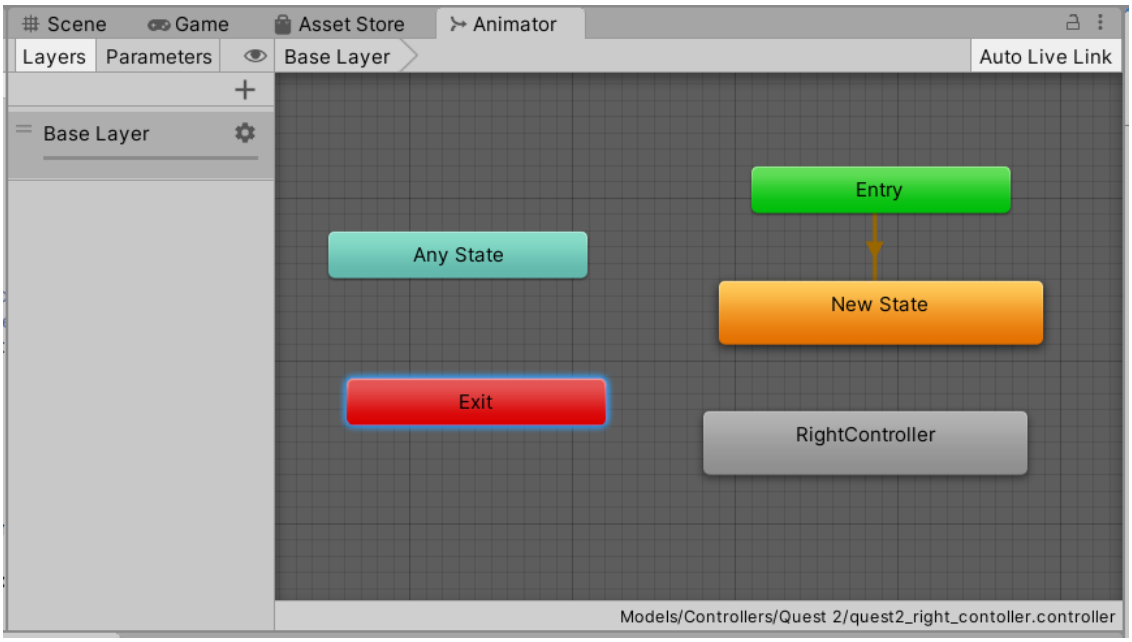
Ajouter un composant « **Animator** » au GameObject représentant la manette droite et créer une animation « **RightController** ».



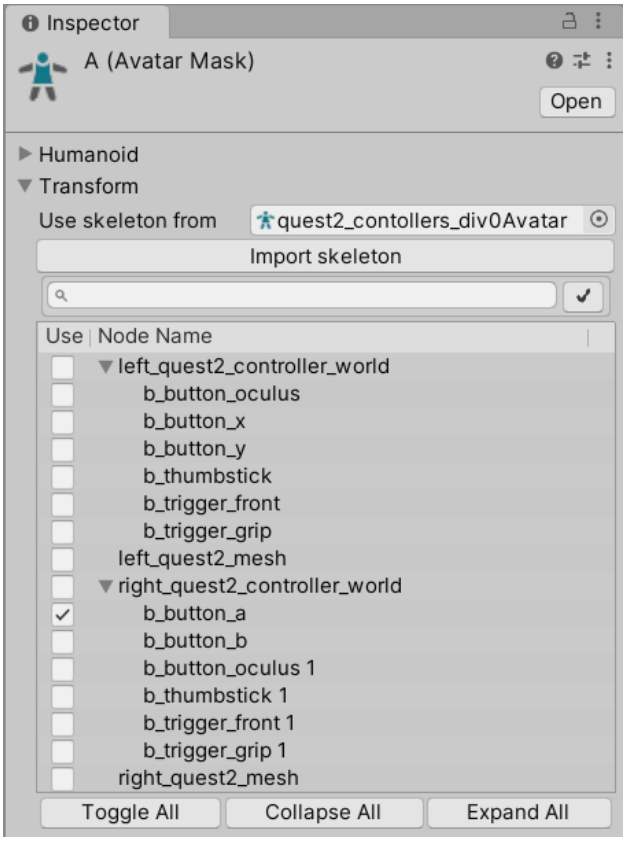
Dans cette animation, abaisser le bouton A et le bouton B sur environ 0,10 secondes.

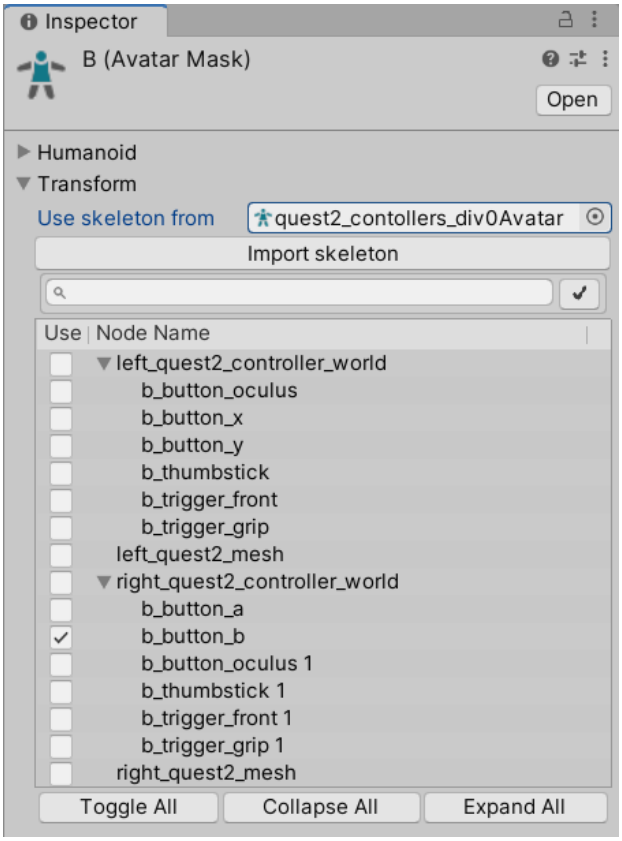


Dans le contrôleur d'animation, faire d'un état vide l'état par défaut du calque de base.

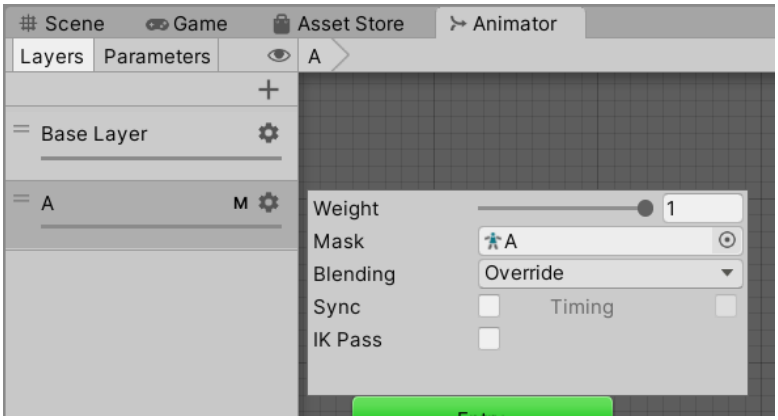


Créer 2 masques d’avatar chacun pour contrôler un des boutons.

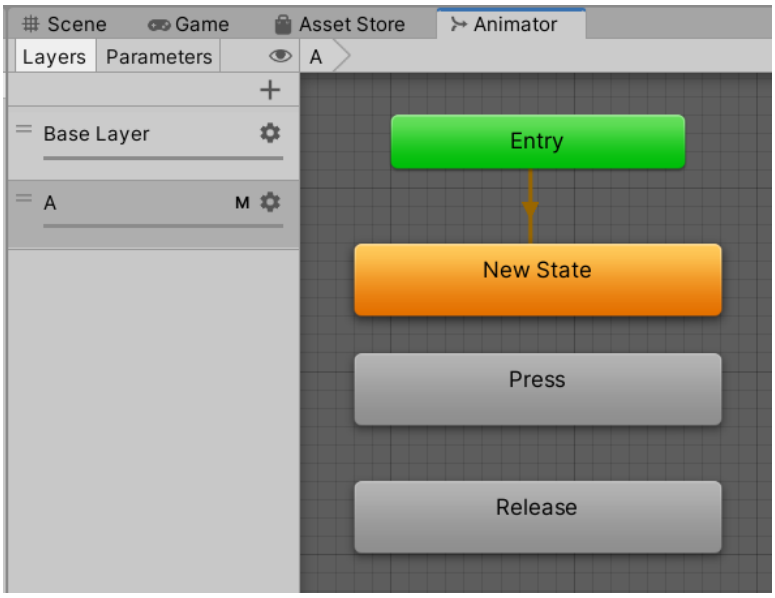




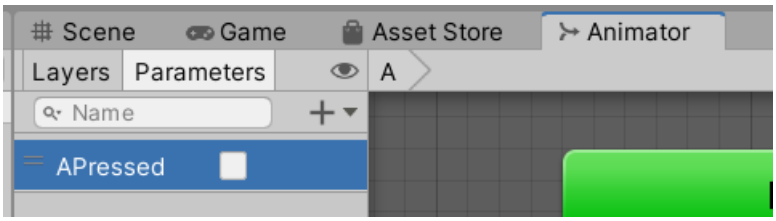
Créer un calque (**Layer**) pour le bouton A. Lui définir un poids de 1 et choisir le masque d’avatar correspondant.



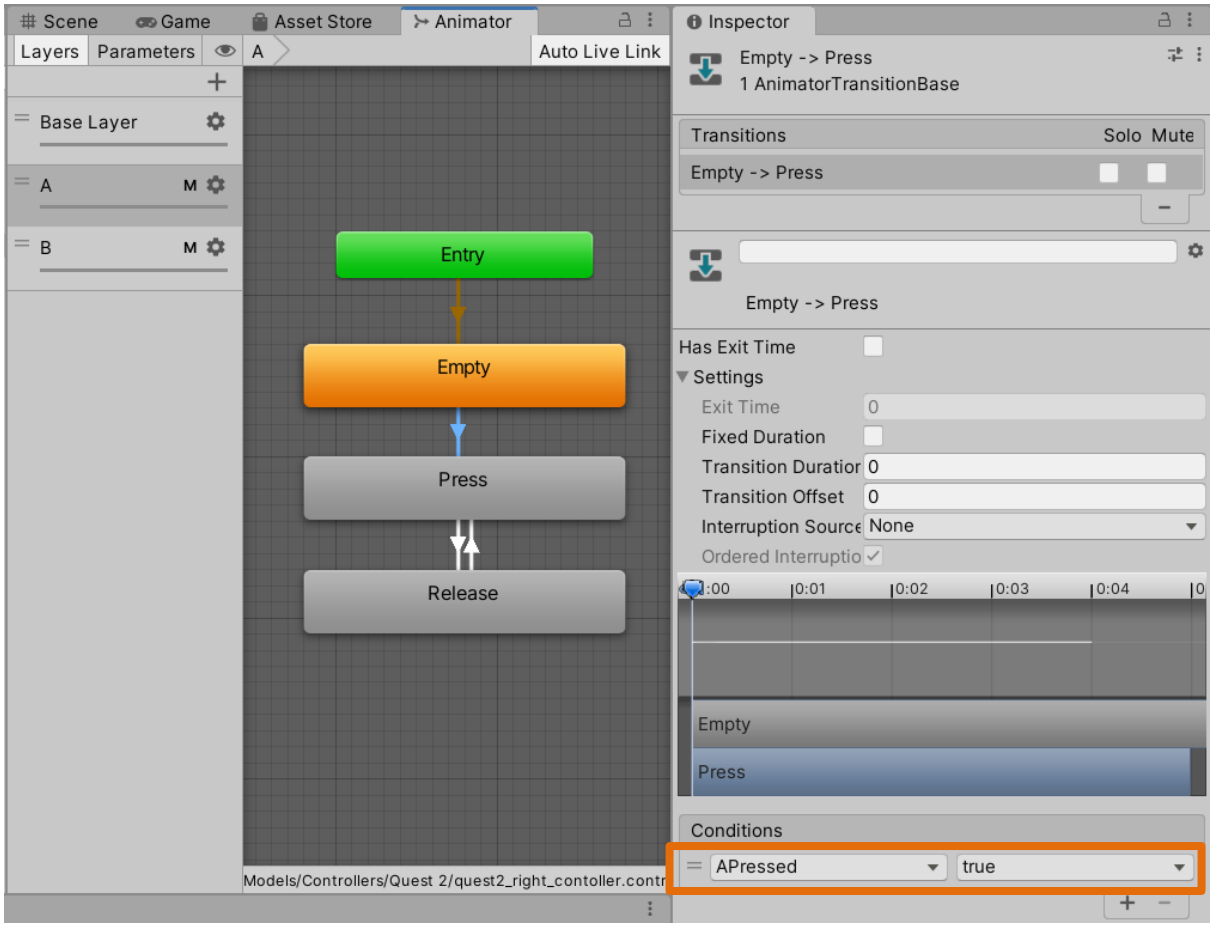
Configurer la machine à état suivante. L'état « **Press** » est associé à l'animation créée et l'état « **Release** » est associé à la même animation mais en vitesse **-1**.



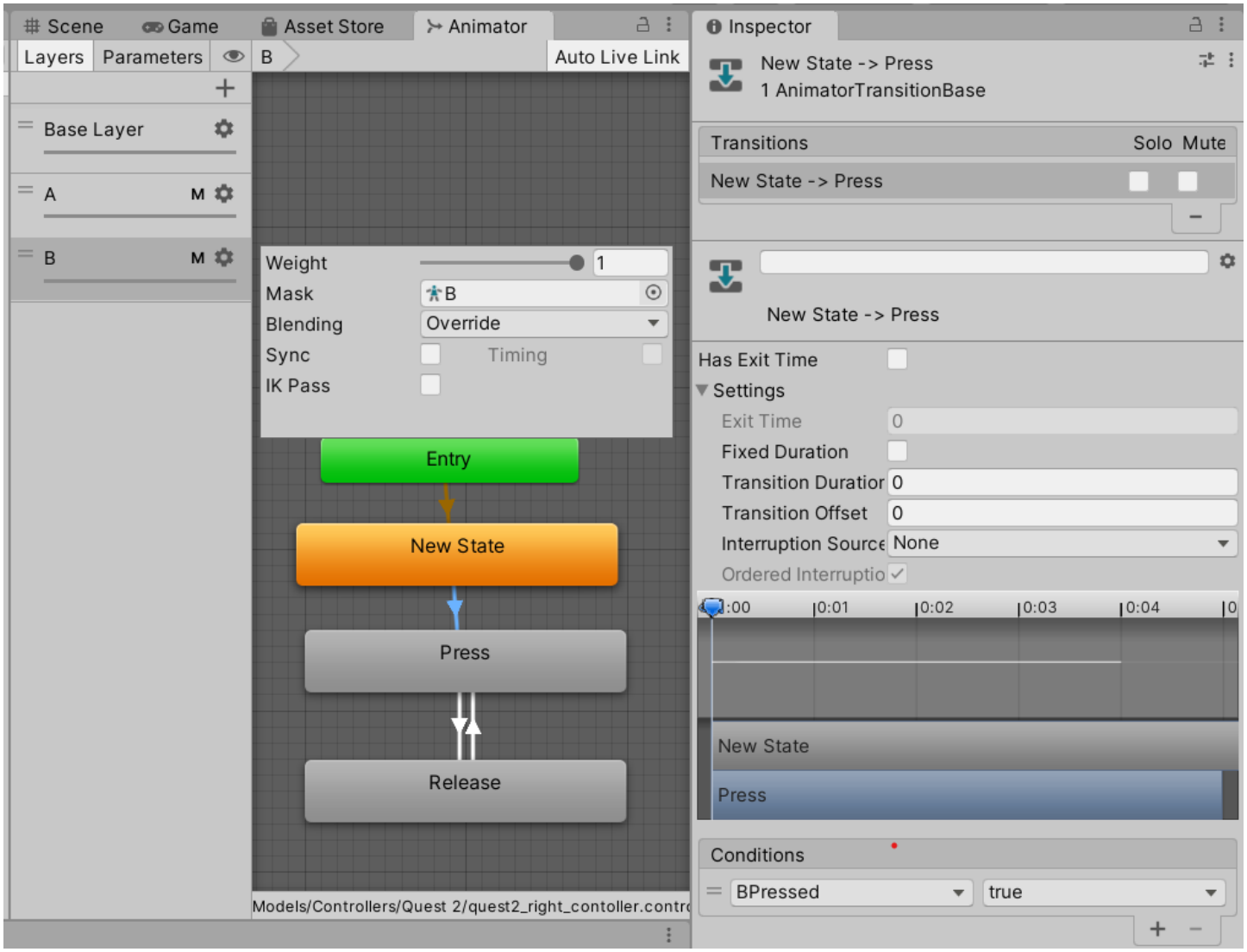
Créer un paramètre de type booléen.



Mettre en place des transitions basées sur ce paramètre.



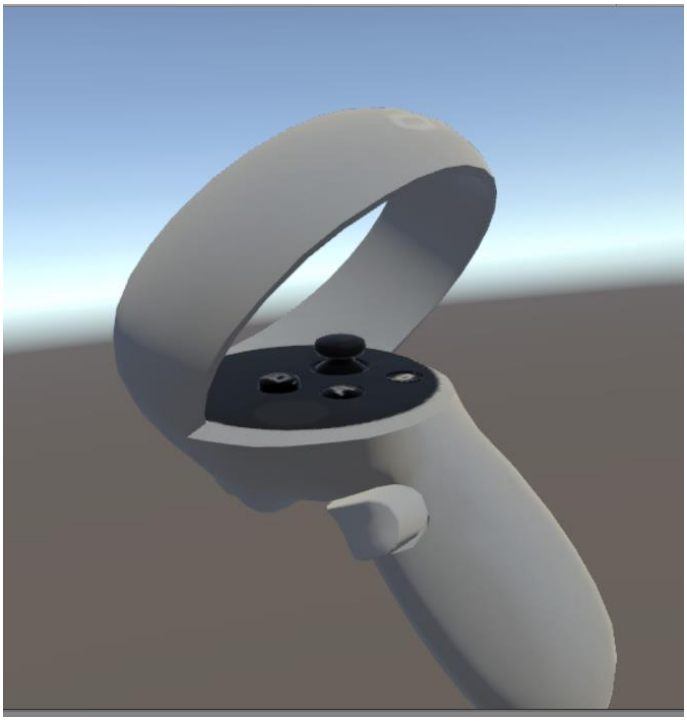
Faire de même pour le bouton B.

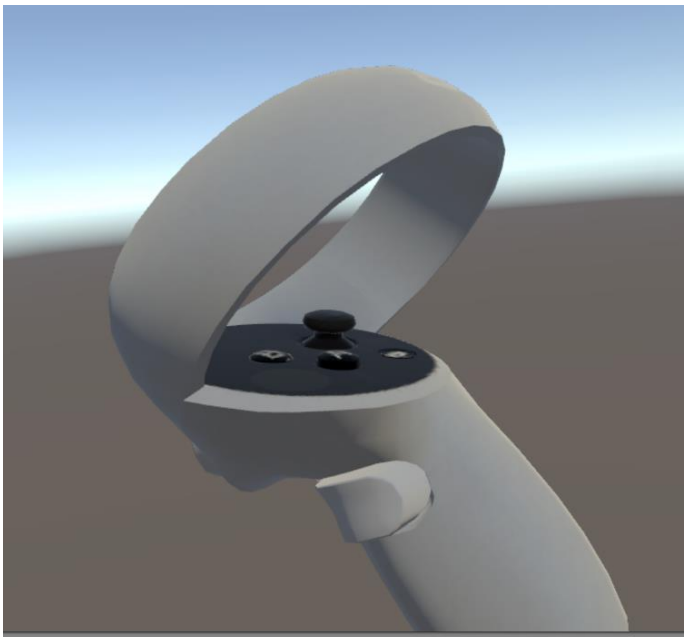


Adapter votre script pour contrôler les paramètres définis dans le contrôleur d'animation.

```
public void OnAPressed(InputAction.CallbackContext context)
{
    if (context.started)
    {
        Debug.Log("A Pressed");
        if (rightAnimator)
        {
            rightAnimator.SetBool("APressed", true);
        }
    }
    if (context.canceled)
    {
        Debug.Log("A Released");
        if (rightAnimator)
        {
            rightAnimator.SetBool("APressed", false);
        }
    }
}
```

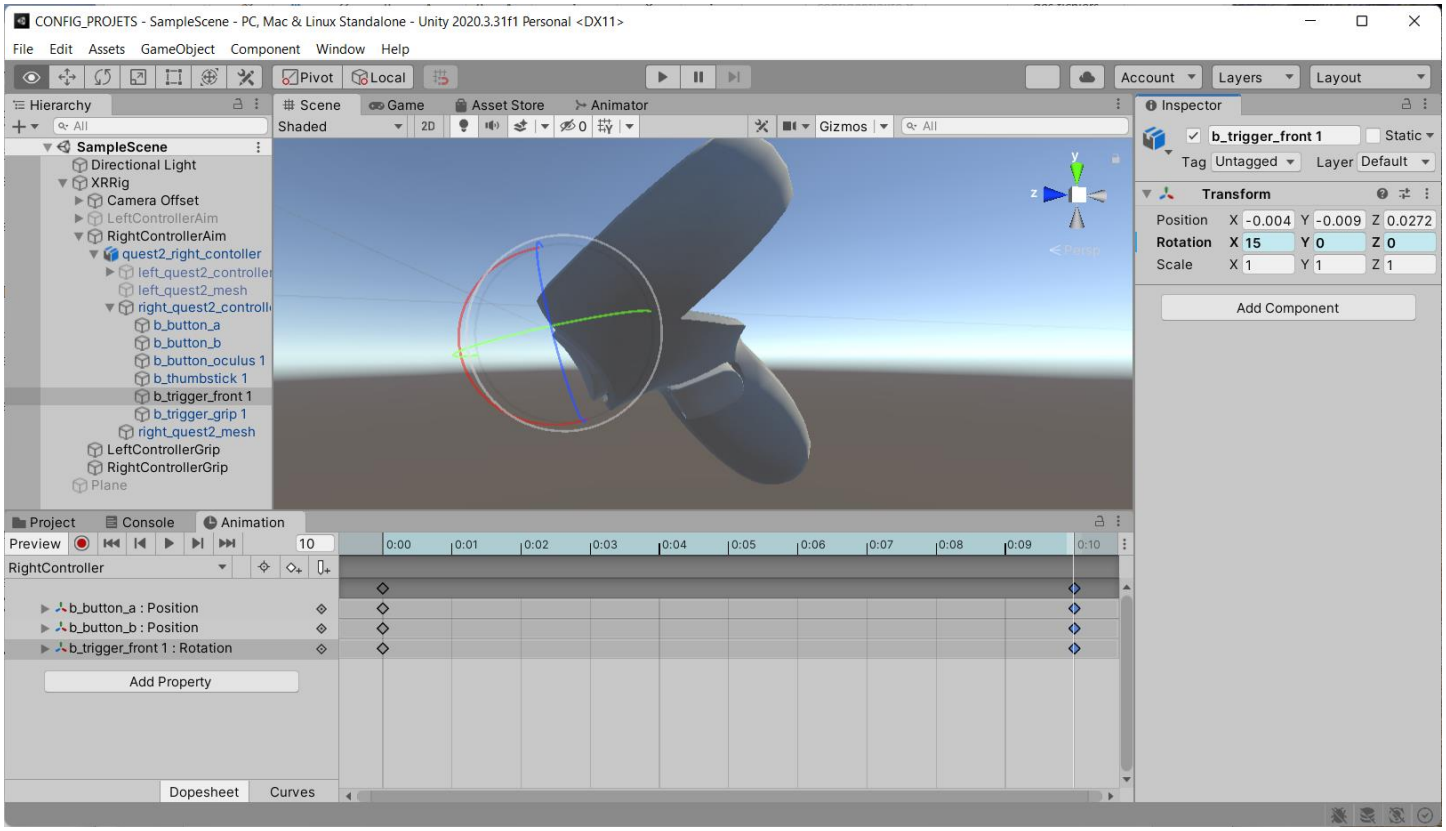
Tester. Vous devriez voir les animations des boutons lorsque vous les enfoncez.



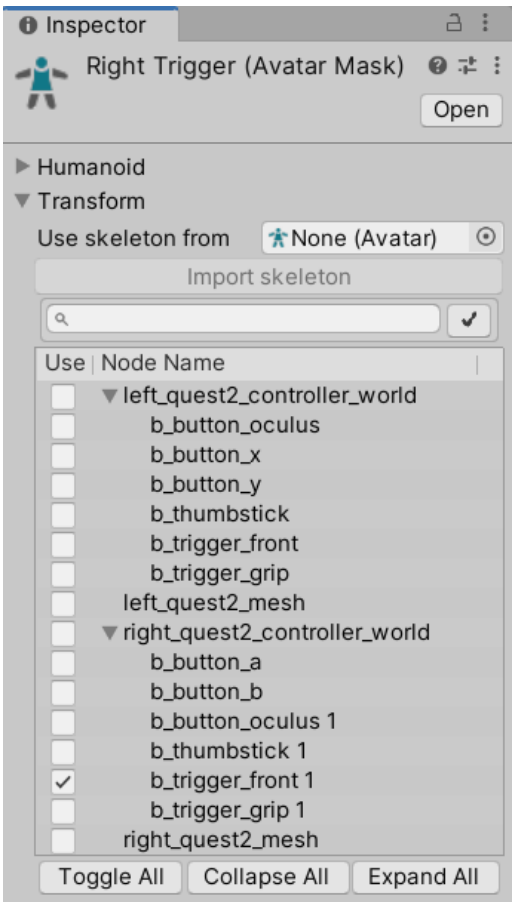


8. Animer les triggers des contrôleurs

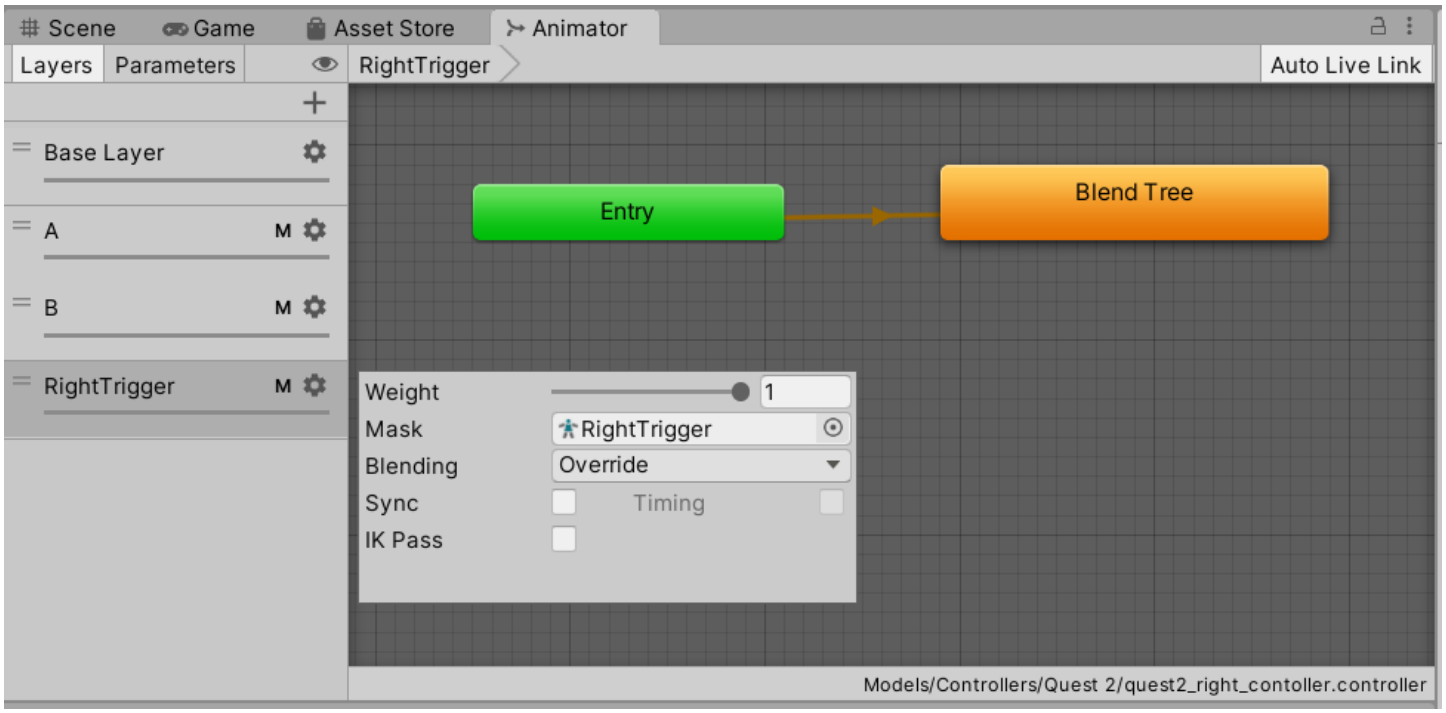
Modifier l'animation pour ajouter une animation sur le trigger.



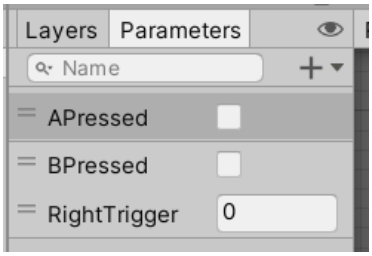
Créer un masque d'avatar pour cet objet.



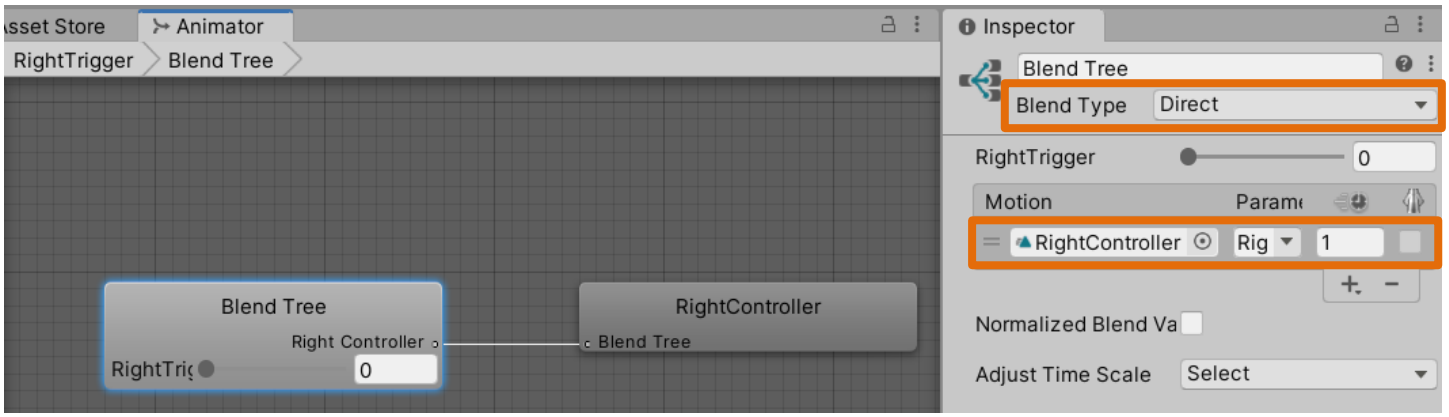
Dans le contrôleur d'animation, ajouter un nouveau calque utilisant ce masque d'avatar et ajouter un état de type « **Blend Tree** ».



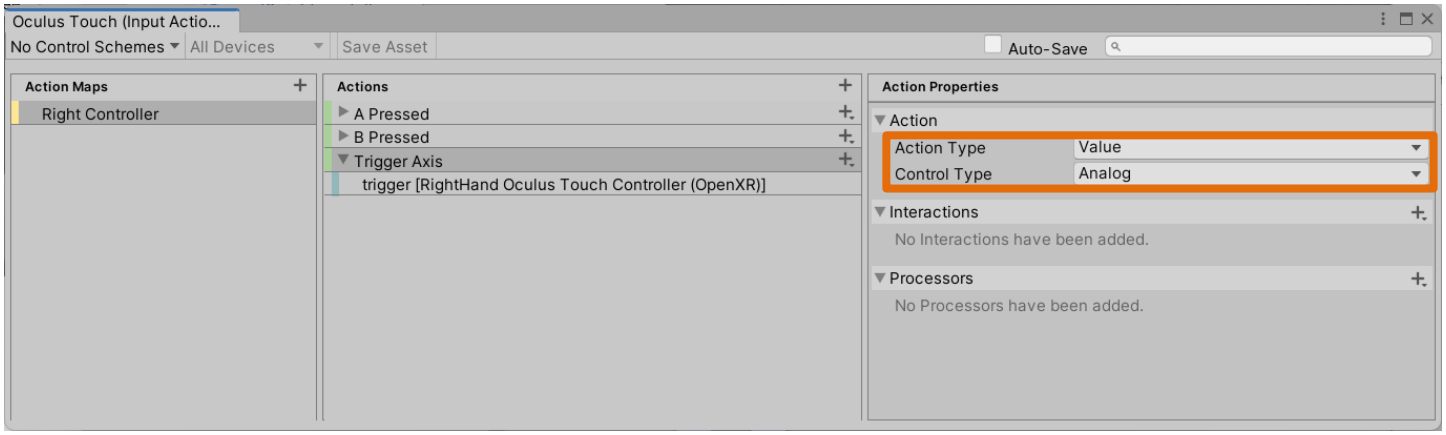
Cela a créé automatiquement un paramètre. Le renommer « **RightTrigger** ».



Editer l'état de type Blend Tree. Passer le « **Blend Type** » à « **Direct** » et sélectionner l'animation.



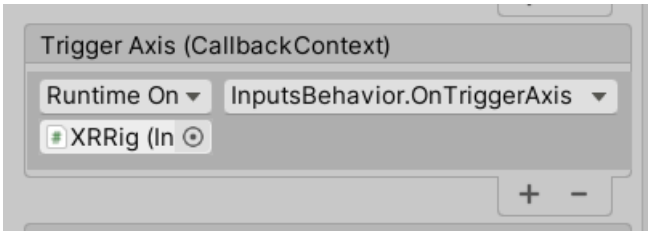
Ajouter une action de type « **Analog** » associée à la gâchette.



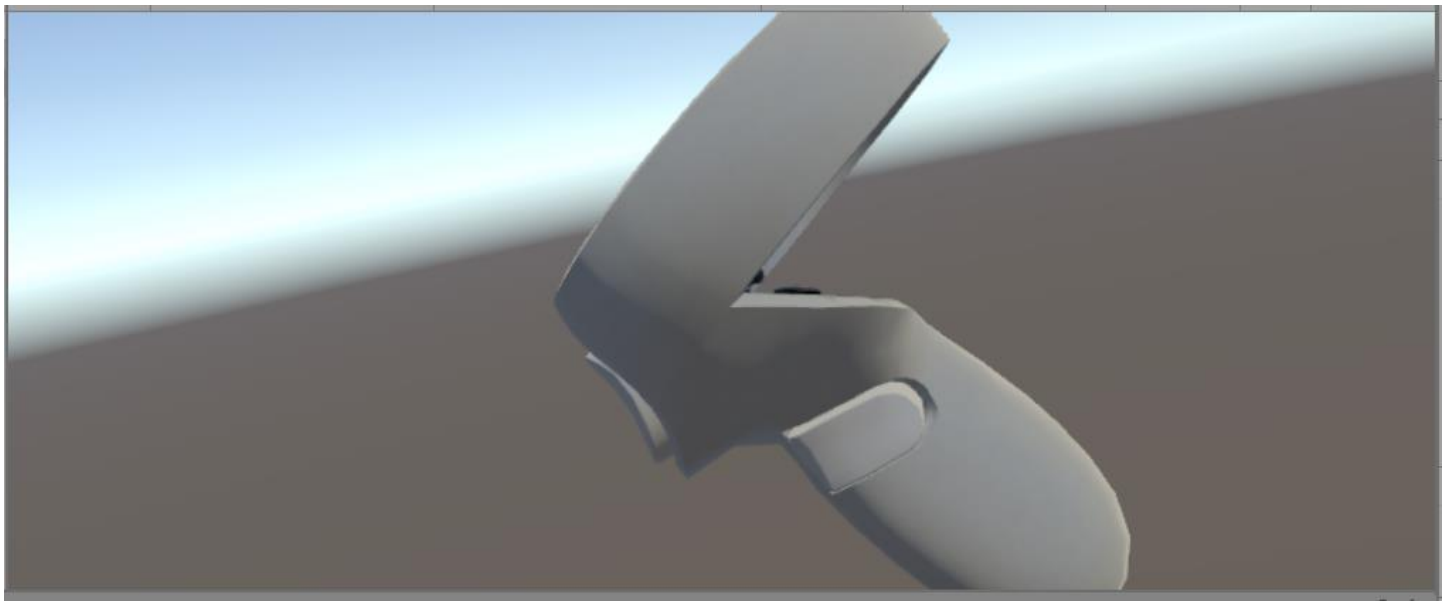
Ajouter une fonction contrôlant le paramètre « **RightTrigger** » du contrôleur d'animation.

```
public void OnTriggerAxis(InputAction.CallbackContext context)
{
    if (rightAnimator)
    {
        rightAnimator.SetFloat("RightTrigger", context.ReadValue<float>());
    }
}
```

Dans le composant « **Player Input** », associer cette fonction à l'évènement correspondant à votre action.

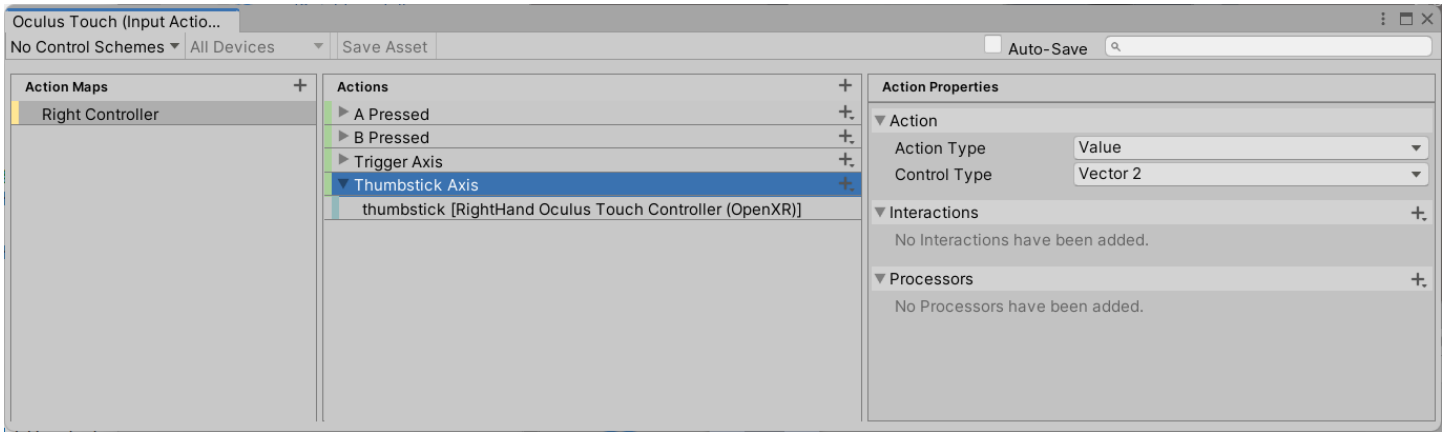


Tester. La gâchette virtuelle devrait suivre la gâchette réelle.



9. Animer les joysticks

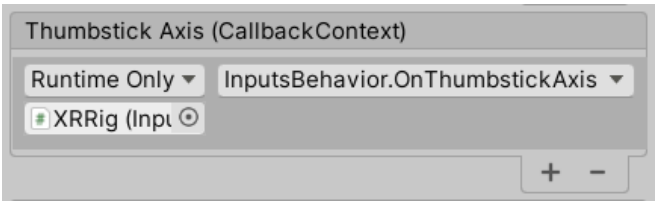
Pour les joysticks, nous allons utiliser une autre technique. Nous allons contrôler directement les angles d'Euler locaux. Créer une action de type « **Vector 2** » associée au joystick droit.



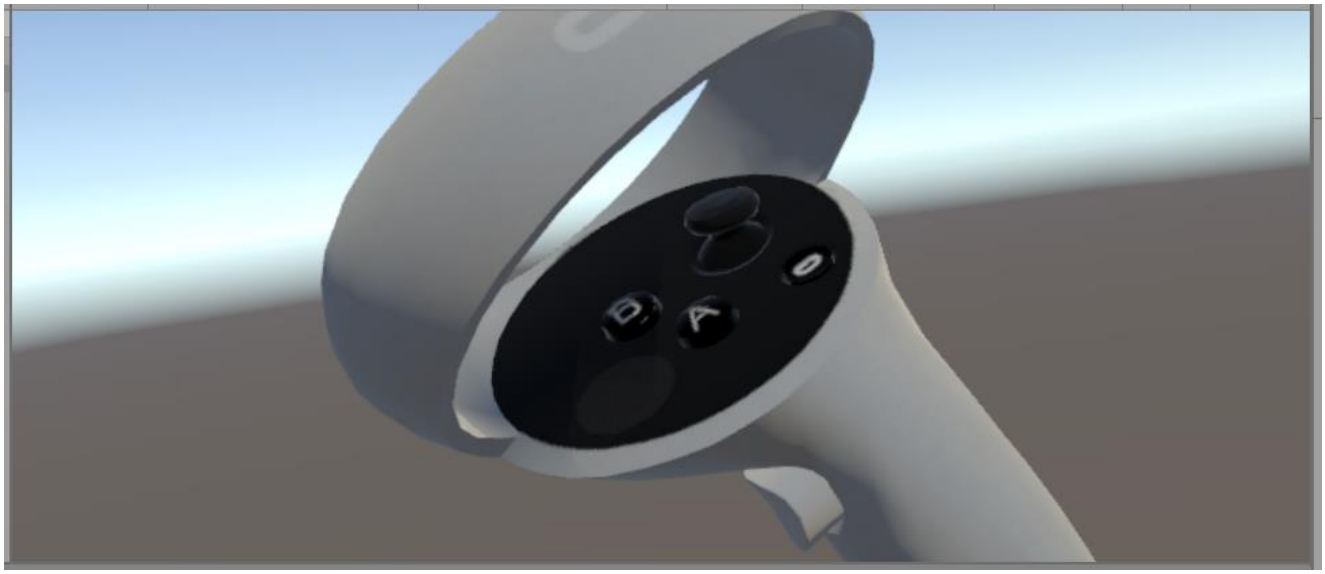
Ajouter une fonction qui contrôle les angles d'Euler du GameObject correspondant.

```
    public void OnThumbstickAxis(InputAction.CallbackContext context)
    {
        if (rightThumbstick)
        {
            Vector2 thumbstickValue = context.ReadValue<Vector2>();
            rightThumbstick.transform.localEulerAngles = new Vector3(thumbstickValue.y, 0, -
thumbstickValue.x) * 15f;
        }
    }
```

Associer cette fonction à l'évènement correspondant.

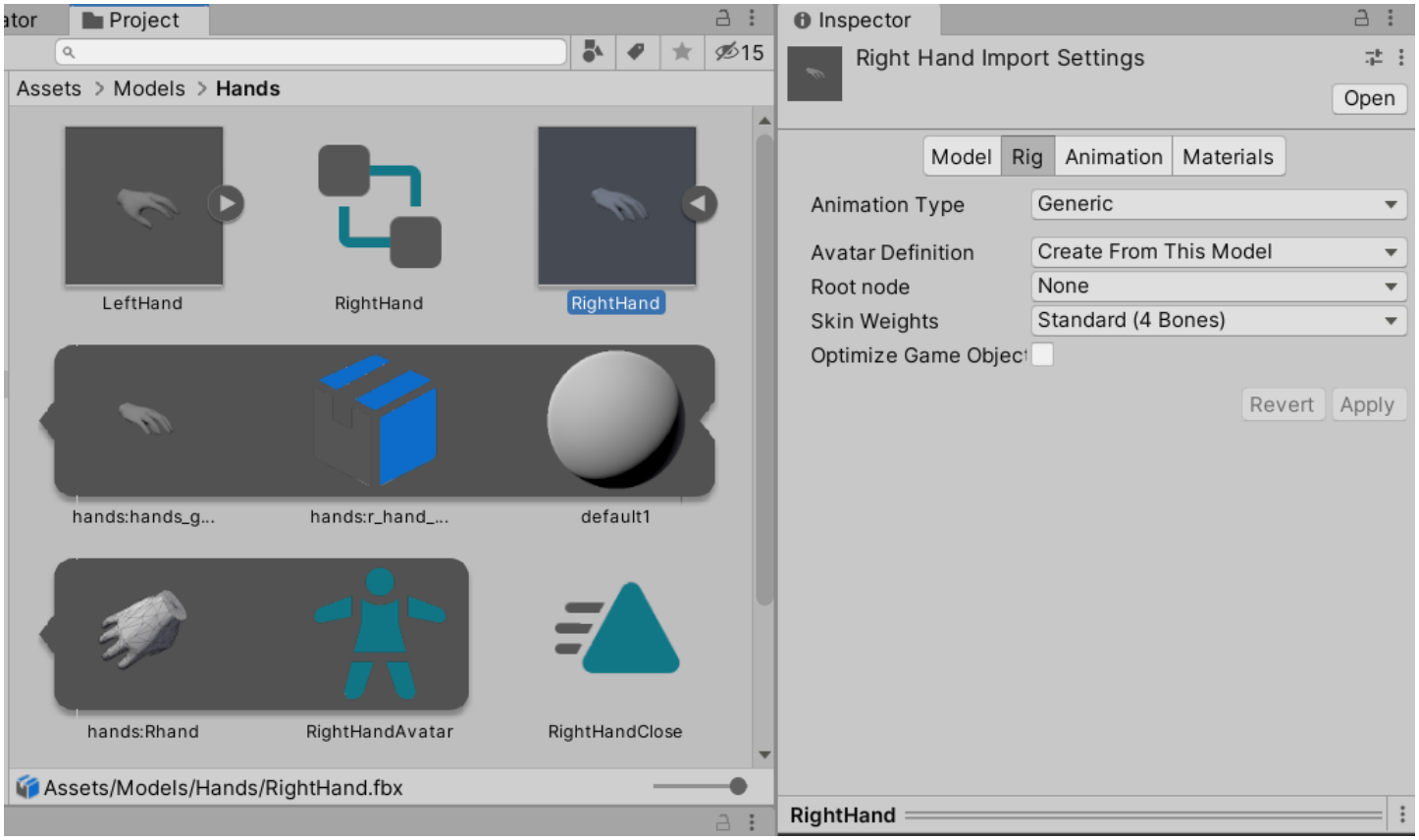


Tester. Le joystick virtuel devrait suivre le joystick réel.

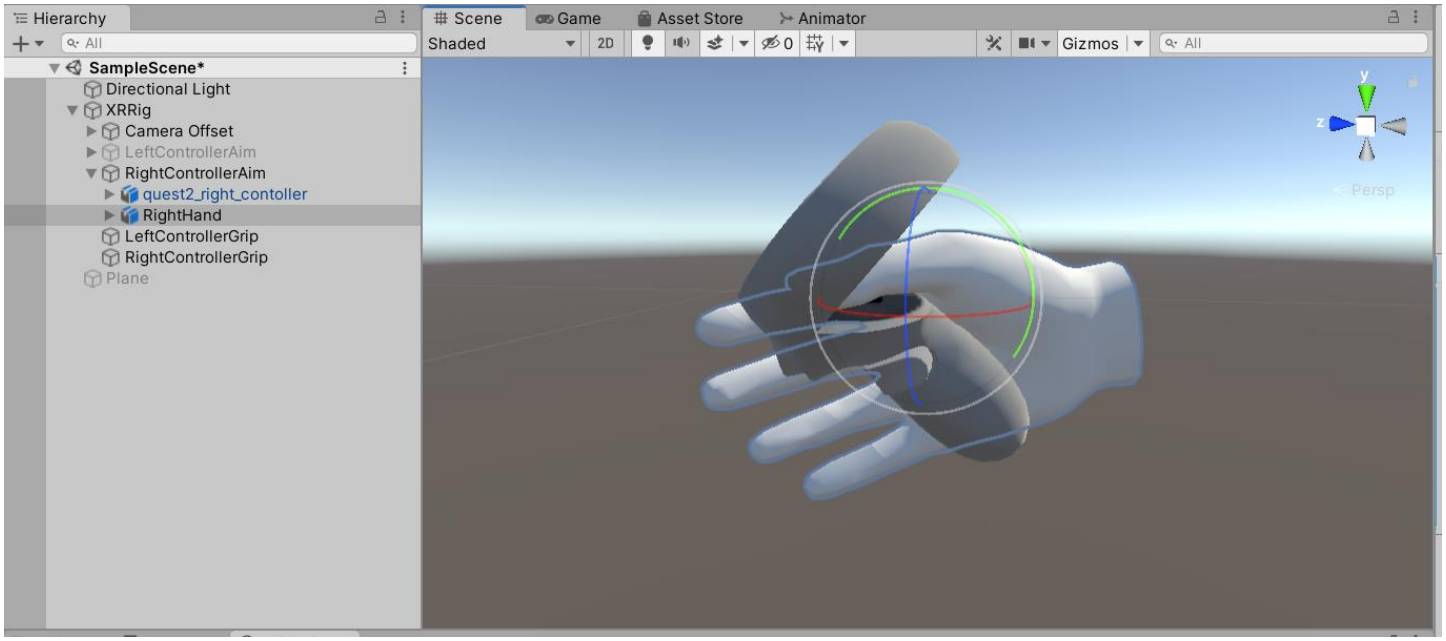


10. Animer des mains virtuelles

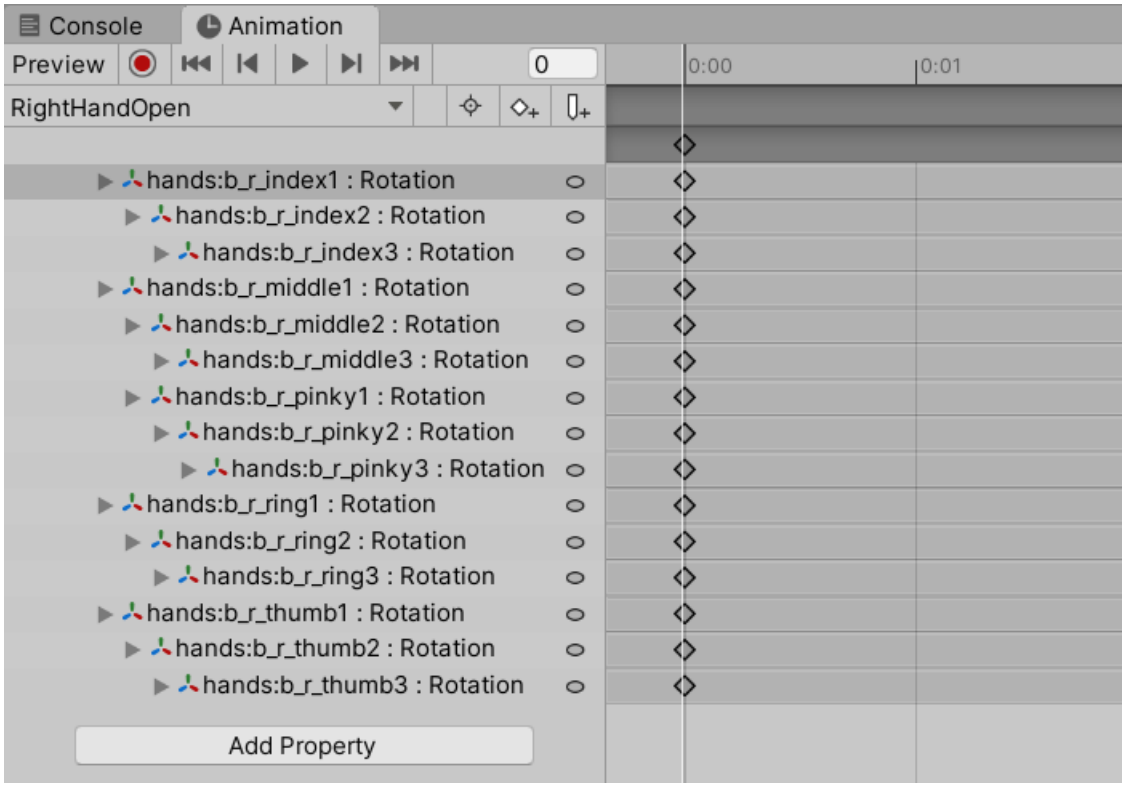
Une autre façon d'animer est de définir des poses, c'est à dire des animations avec juste une clé. Voyons cela avec l'animation de mains virtuelles. D'abord importer les modèles FBX de mains et générer les avatars correspondants.



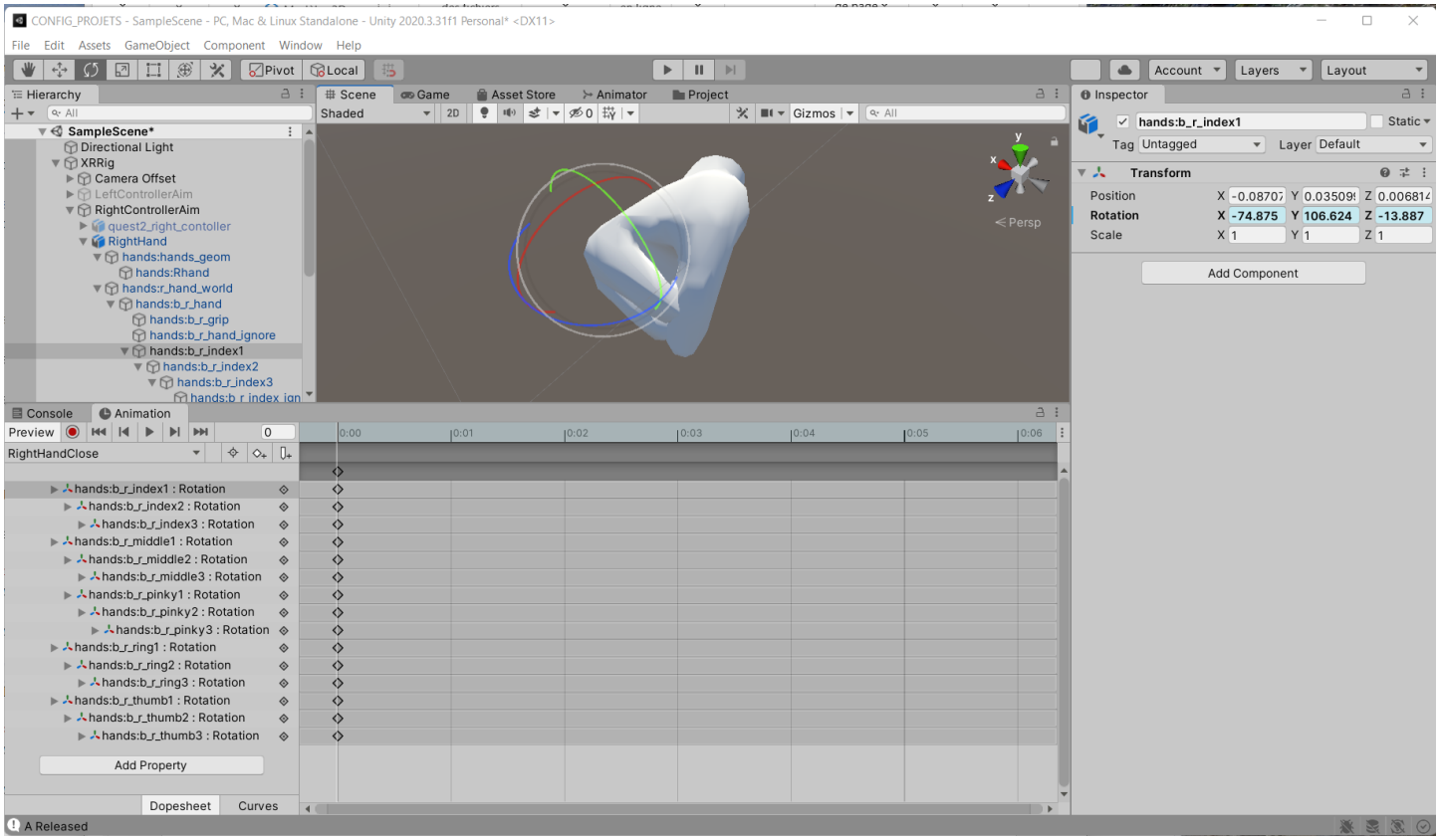
Placer la main dans la hiérarchie du contrôleur.



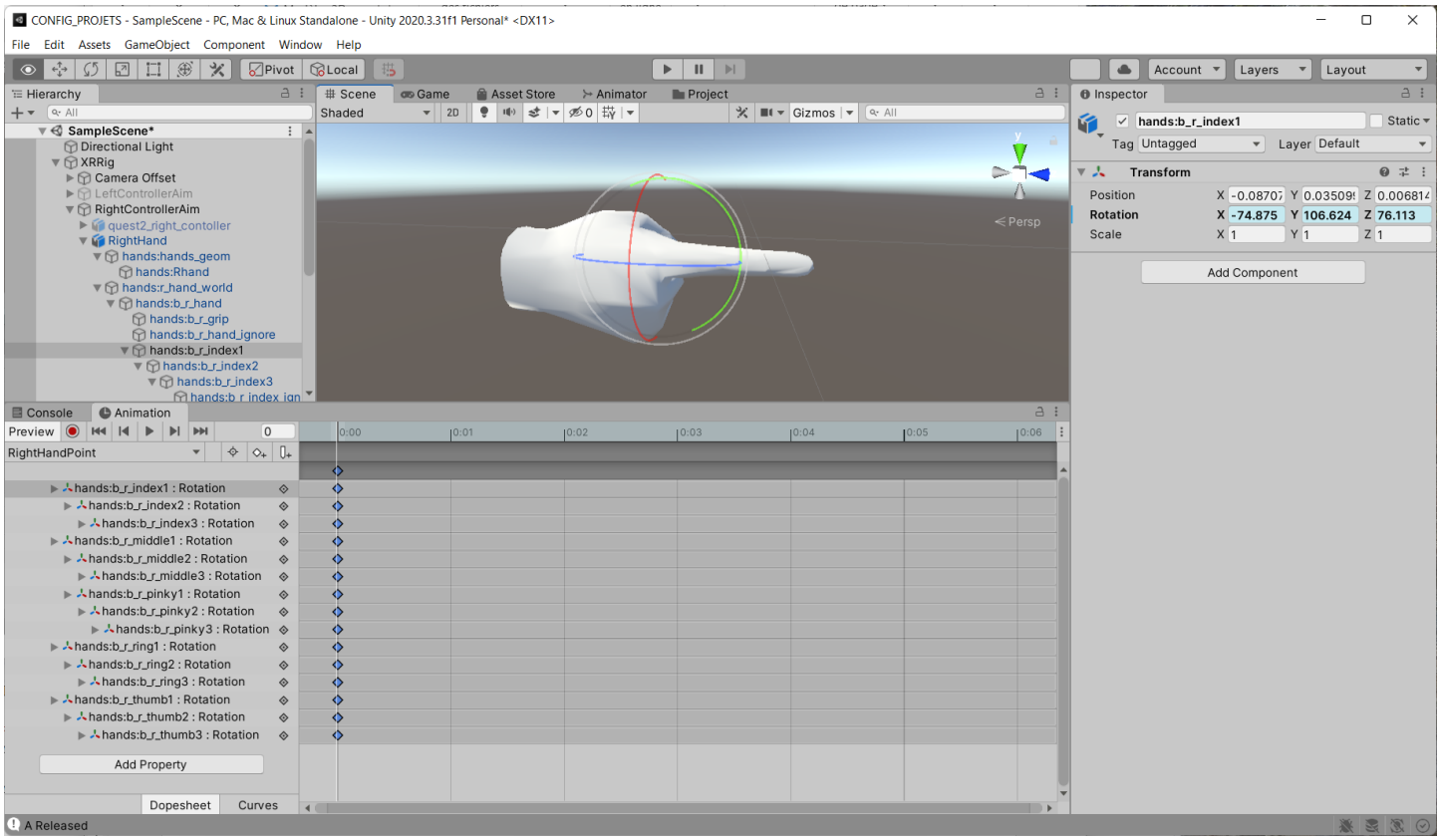
Commencer par créer une première animation « **Opened** ». Ajouter toutes les phalanges et ne laisser qu'une seule clé.



Créer une deuxième animation « **Closed** » et **copier/coller** les données de l'animation « **Opened** ». Ajuster les clés pour faire une main fermée.

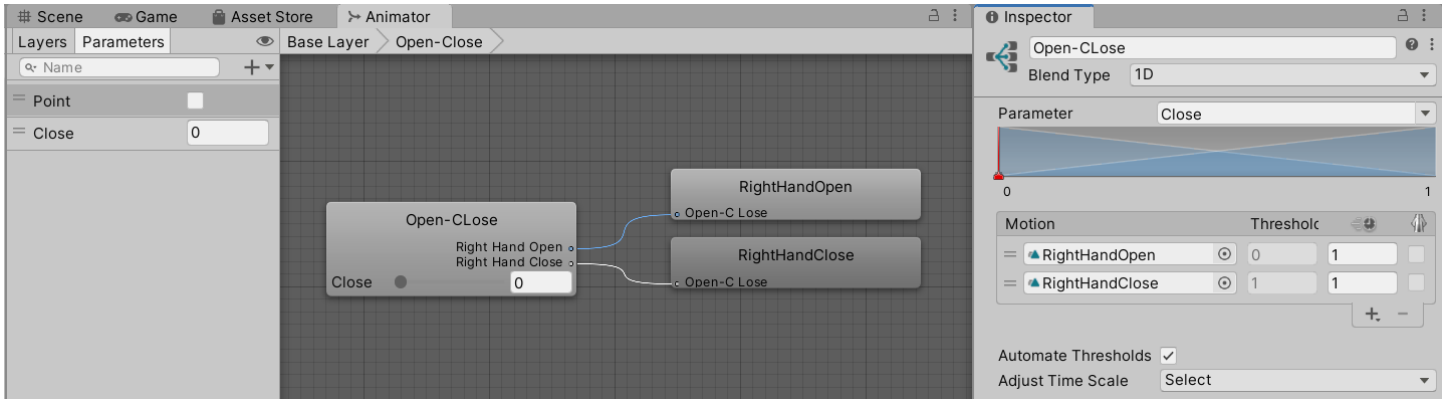


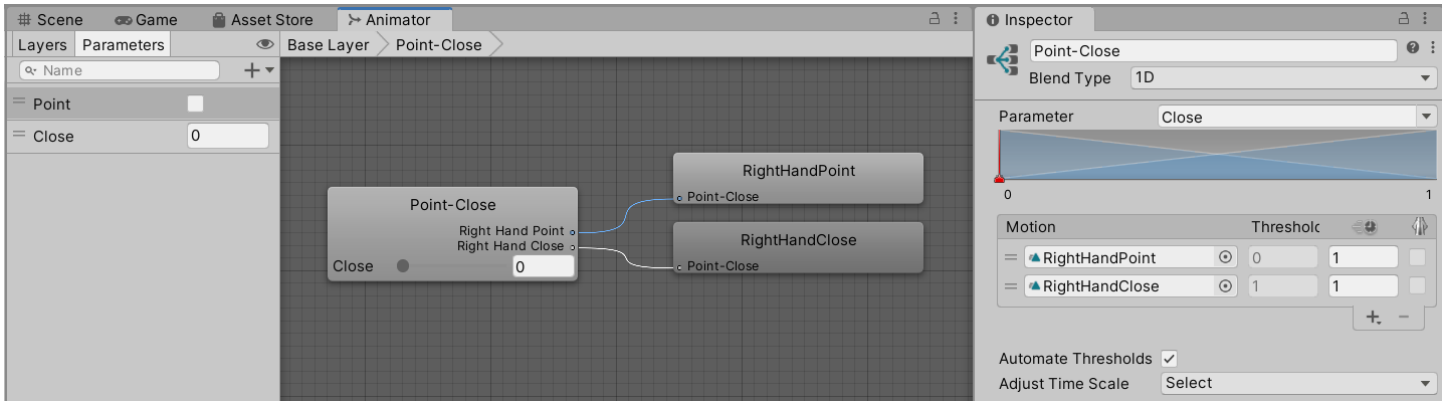
Créer une troisième animation « **Pointing** » et **copier/coller** les données de l'animation « **Closed** ». Ajuster l'index pour qu'il pointe.



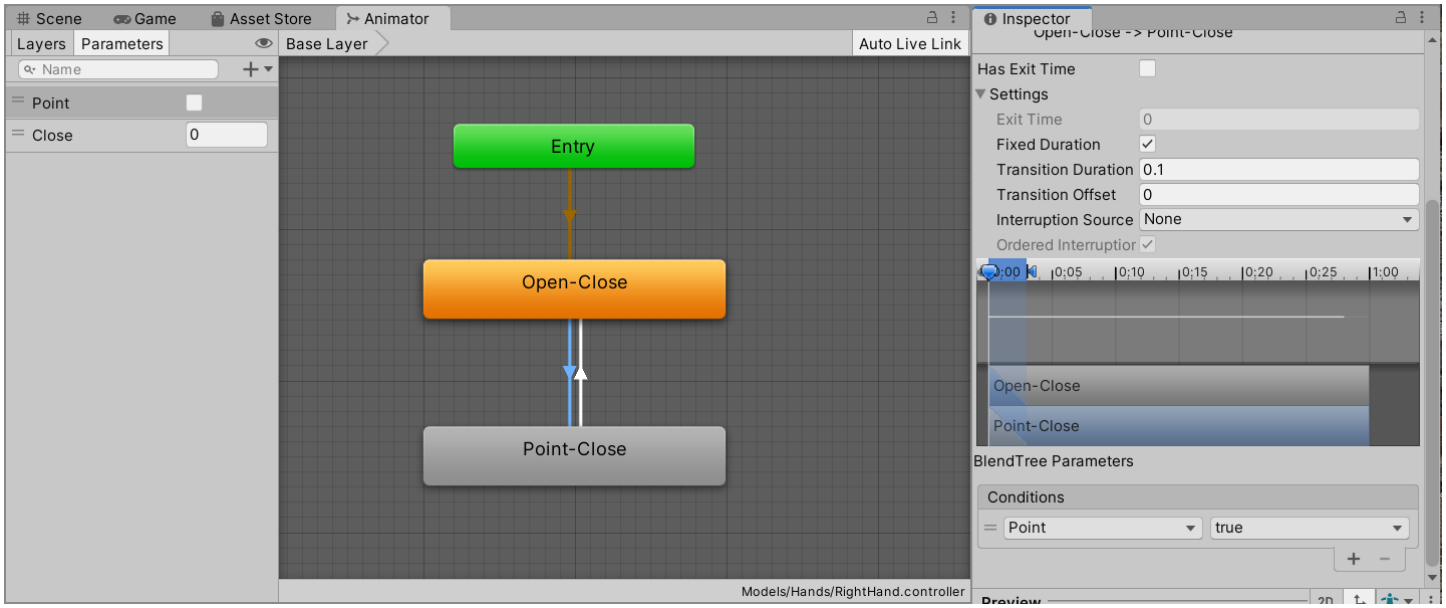
Nous allons donner la priorité à la pose « **Closed** ». Dans le contrôleur d'animation, ajouter deux états de type Blend Tree :

- Un définissant une animation de « **Opened** » vers « **Closed** »
- Un définissant une animation de « **Pointing** » vers « **Closed** »



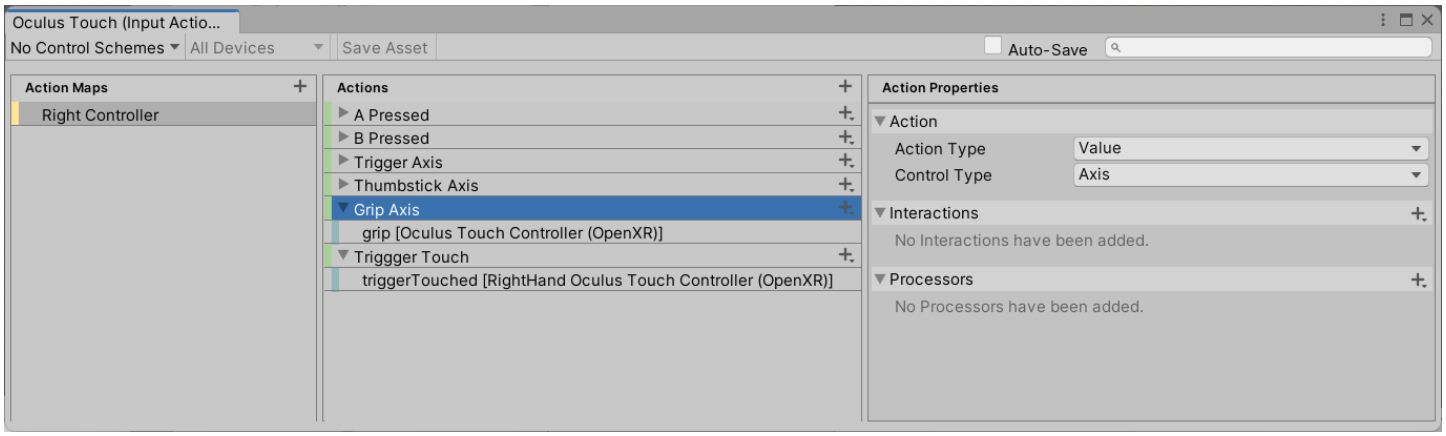


Dans le calque principal définir que le passage d'un Blend Tree à l'autre se fait via la valeur d'un booléen.



Définir deux actions :

- Une correspondant à l'appui sur le grip (Axis)
- Une correspondant au toucher du trigger (button)



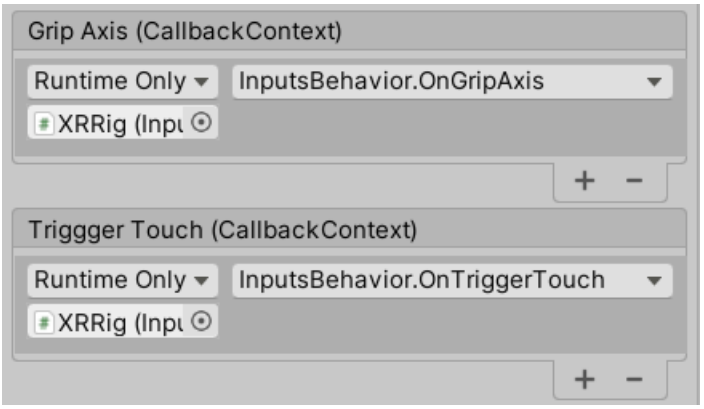
Ajouter deux fonctions :

- Une pour contrôler la valeur de fermeture
- Une pour contrôler le fait de pointer

```
public void OnGripAxis(InputAction.CallbackContext context)
{
    if (rightHandAnimator)
    {
        rightHandAnimator.SetFloat("Close", context.ReadValue<float>());
    }
}

public void OnTriggerTouch(InputAction.CallbackContext context)
{
    if (context.started)
    {
        if (rightHandAnimator)
        {
            rightHandAnimator.SetBool("Point", false);
        }
    }
    if (context.canceled)
    {
        if (rightHandAnimator)
        {
            rightHandAnimator.SetBool("Point", true);
        }
    }
}
```

Associer ces fonctions aux évènements.



Tester. Vous devriez pouvoir alterner entre les 3 postures.