Documentation

AR-enberg



CANTON Alexandre DANG Thaï-Son DELAURENS Adrien DIEUDONNE Kenneth KUCUKAL Burak

Table des matières

[Interface utilisateur 3](#_Toc7300076)

[Layout 3](#_Toc7300077)

[Scene Menu 3](#_Toc7300078)

[Scene Game 6](#_Toc7300079)

[Scene Test 8](#_Toc7300080)

[Prefabs 8](#_Toc7300081)

[Scripts 10](#_Toc7300082)

[Géolocalisation 11](#_Toc7300083)

[Création des mires et des objets 3D 12](#_Toc7300084)

[Création de la base de données Vuforia 12](#_Toc7300085)

[Création des ImageTarget 14](#_Toc7300086)

[Interactions avec les objets 15](#_Toc7300087)

[Rotation des objets 15](#_Toc7300088)

[Emission de lumière 15](#_Toc7300089)

[LightMatching 19](#_Toc7300090)

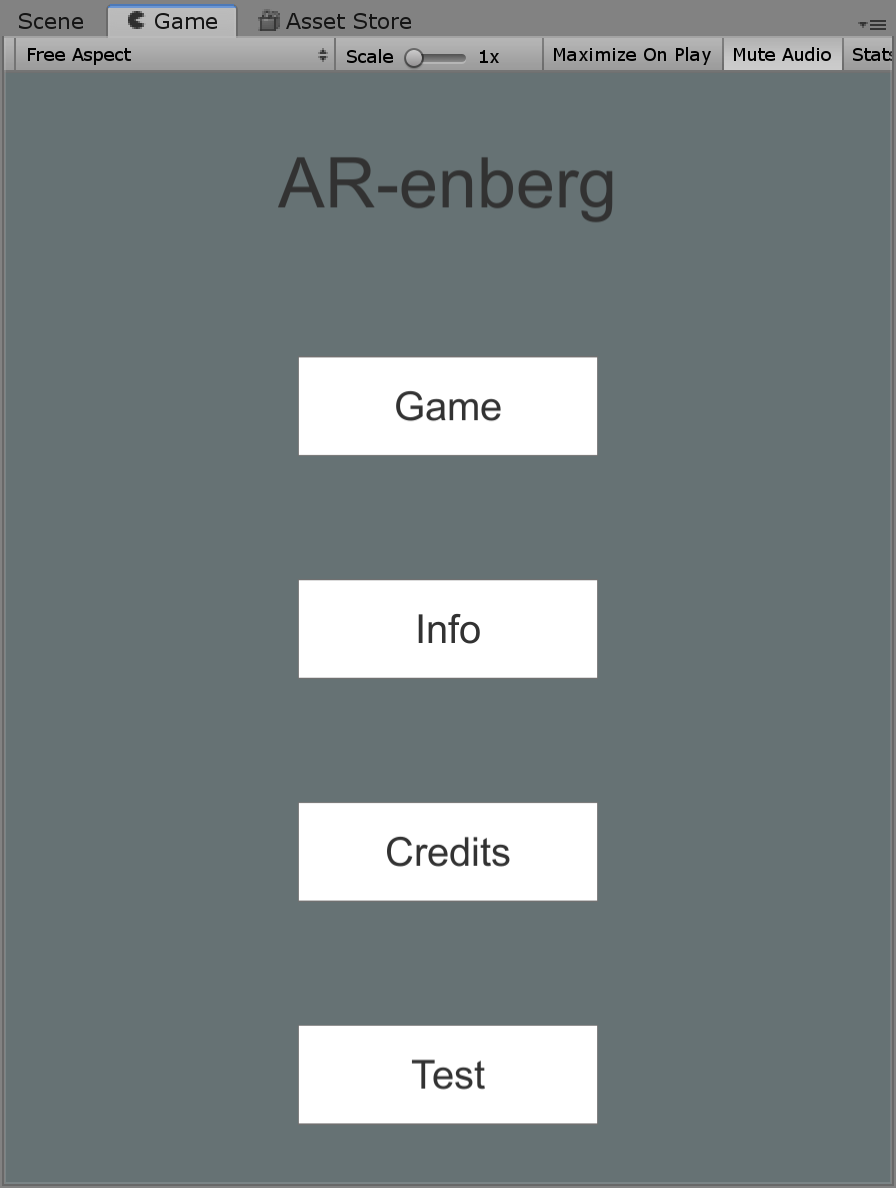
# Interface utilisateur

## Layout

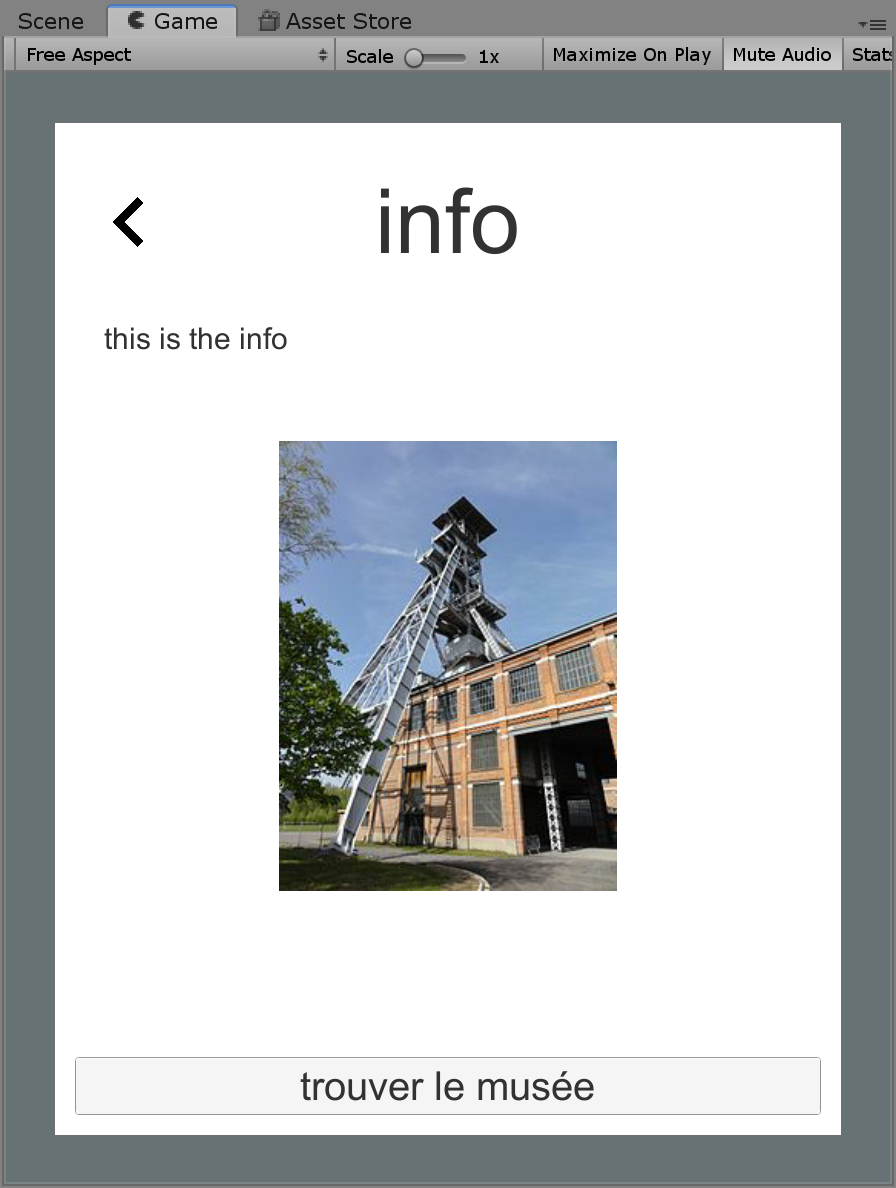
L'application a été séparé en 3 scènes, une pour les menus simples, puis une par menu nécessitant de l’AR, permettant de garder une rapidité de chargement en ne chargeant que ce qui est nécessaire.

### Scene Menu

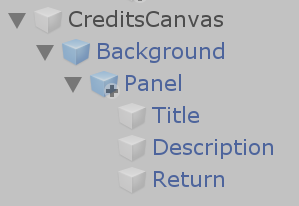
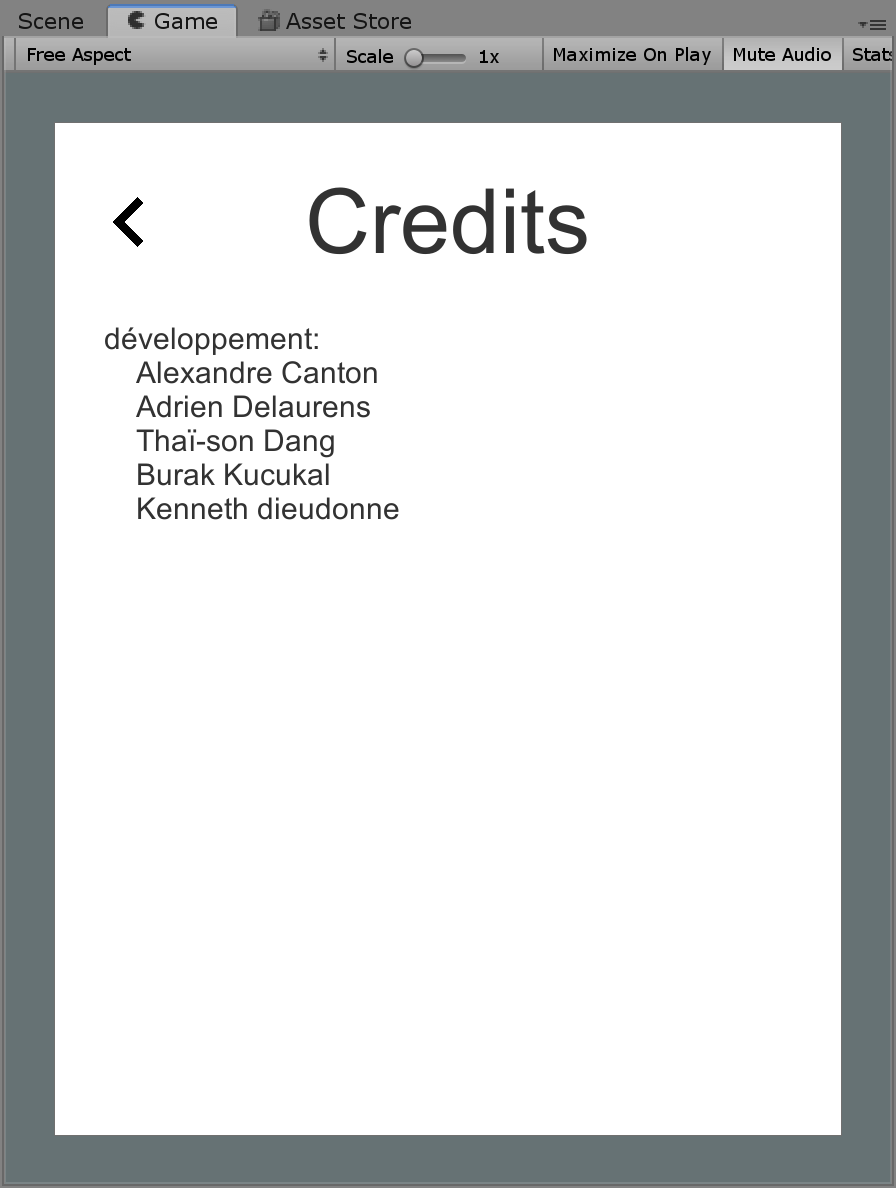
* Title canvas :



* Info canvas :

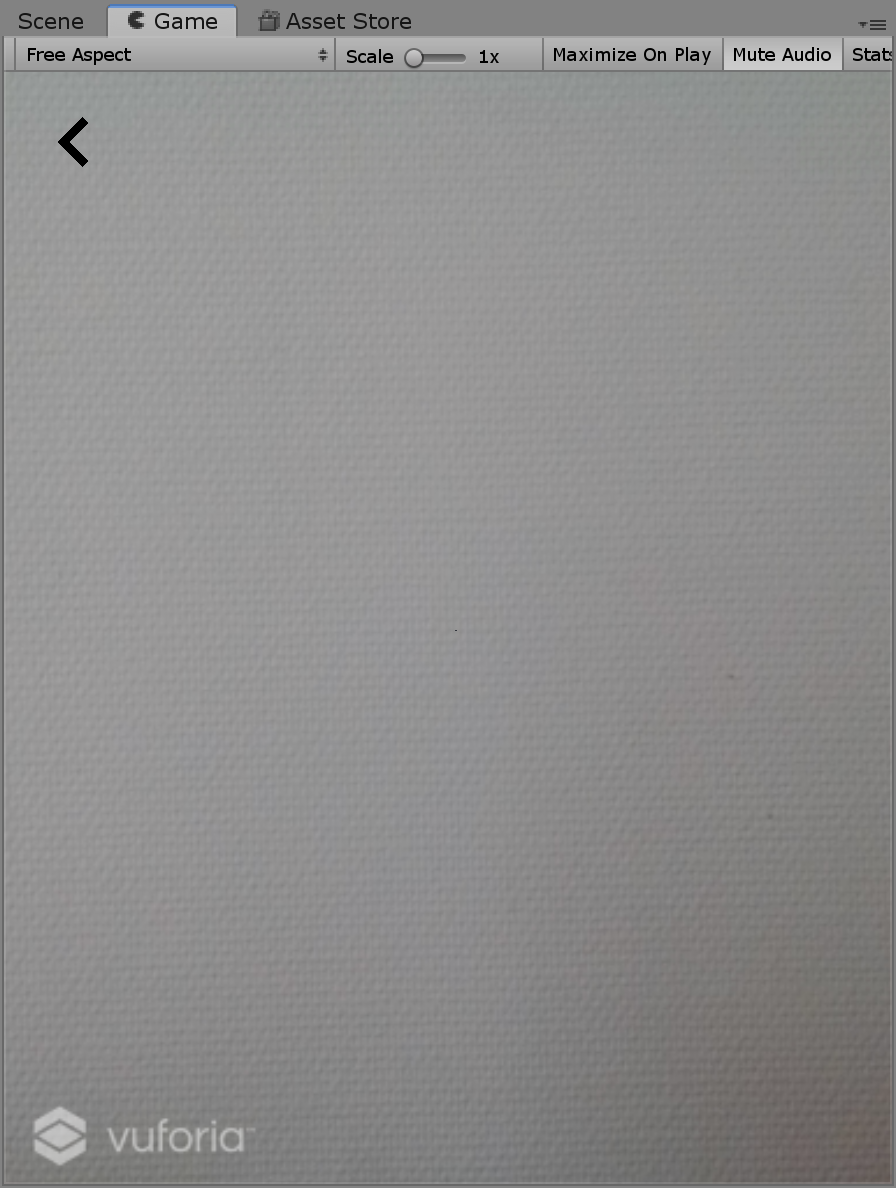


* Credits canvas :

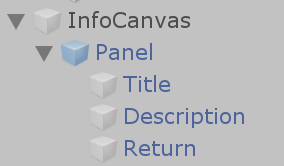


### Scene Game

* Game canvas :

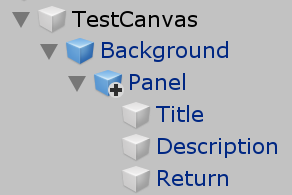
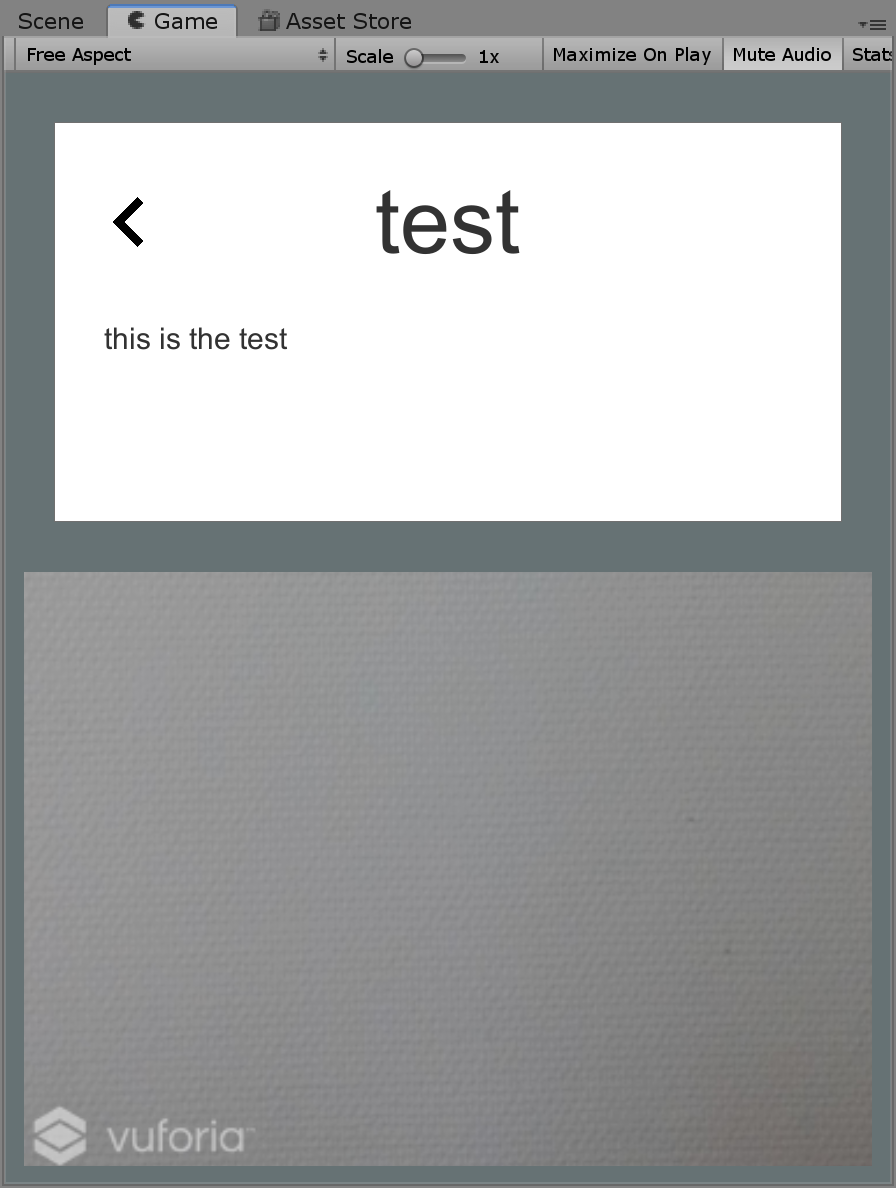


* Info canvas :



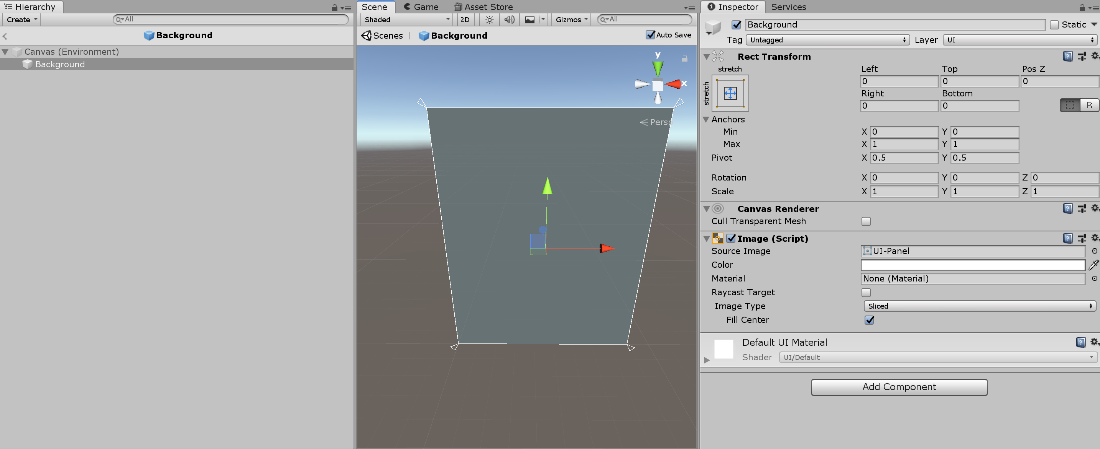
### Scene Test

* Test canvas :



## Prefabs

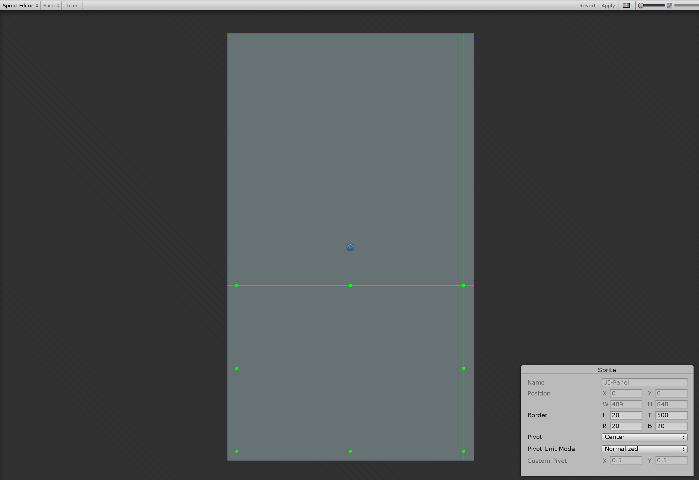
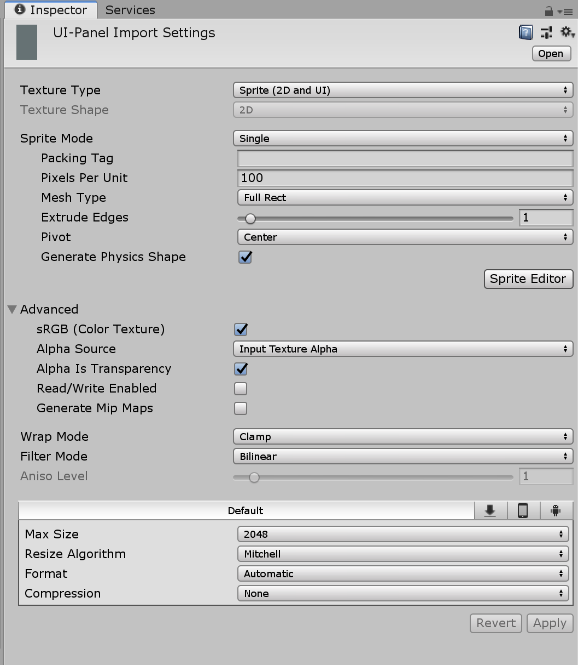
* Background :



Background, l'arrière-plan sur les menus,

Désactiver “fill center” permet de laisser un trou au milieu (utilisé dans la scene test)

Les dimensions du trou sont définies par le découpage du sprite (sprite editor, voir ci-dessous)

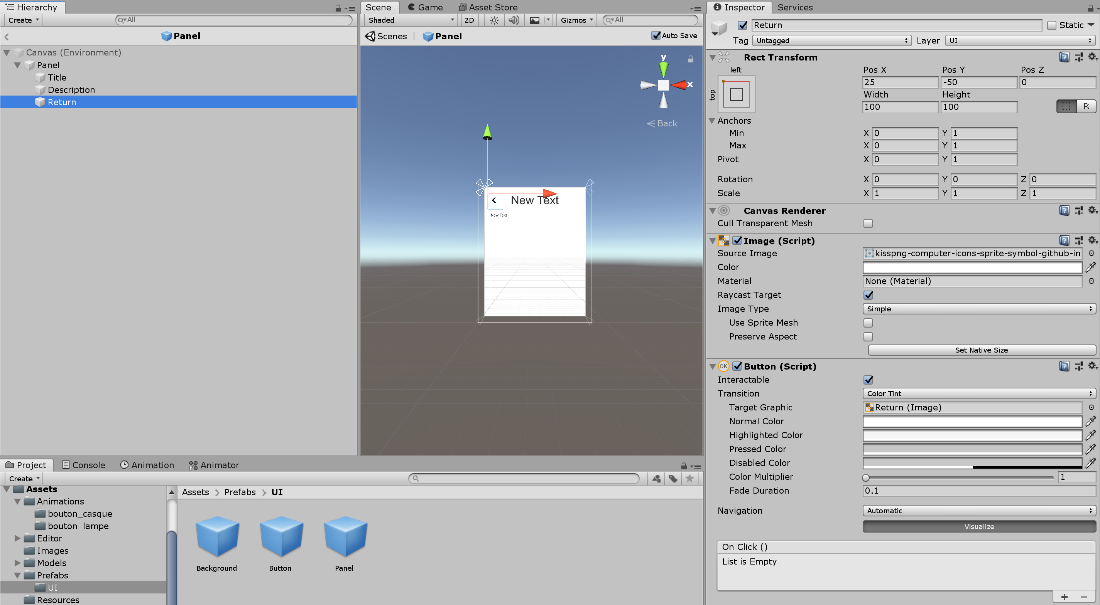


* Button :



Button définis le look de tous les boutons dans le menu titre leur fonction est défini dans OnClick (en bas à droite dans l’image, en bas de l’Inspecteur)

* Panel :



Panel, utilisé dans les scènes :

Menu (info canvas, credits canvas)

ARscene (info canvas)

Test (test canvas)

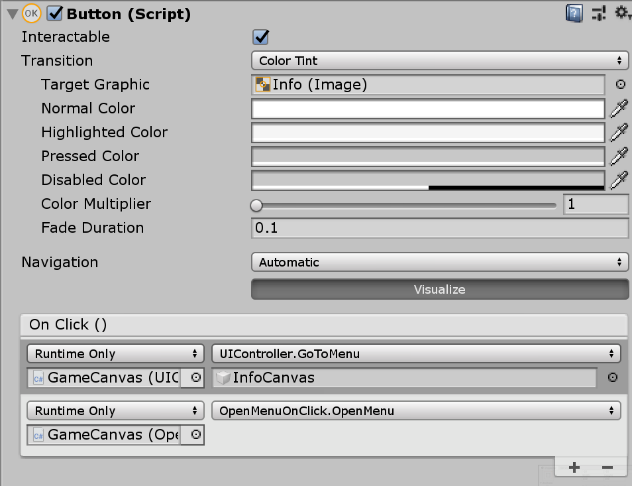
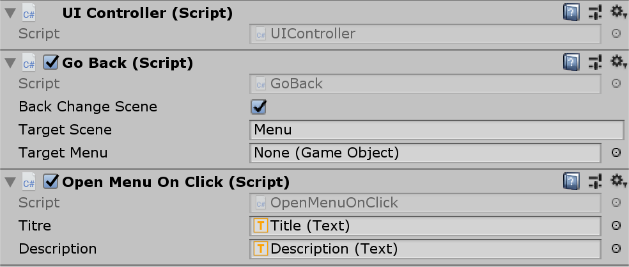
Il permet de définir un look commun pour tous ces menus avec un panneau blanc, un titre, une description, et un bouton retour.

## Scripts

Les scripts de navigation (et *OpenMenuOnClick*) sont attachés au canvas du menu correspondant, ledit canvas est ensuite attaché à la zone OnClick des boutons dans l’inspecteur pour utiliser les fonctions voulues.

Example :

game canvas et le bouton info:



Description de scripts :

OpenMenuOnClick.cs :

Permet d’afficher la bonne description en fonction de l’objet détecté.

ObjectConstants.cs :

Contient les textes à afficher pour les descriptions des objets.

Info.cs :

Structure des informations stockées dans ObjectConstants et utilisées dans OpenMenuOnClick.

# Géolocalisation

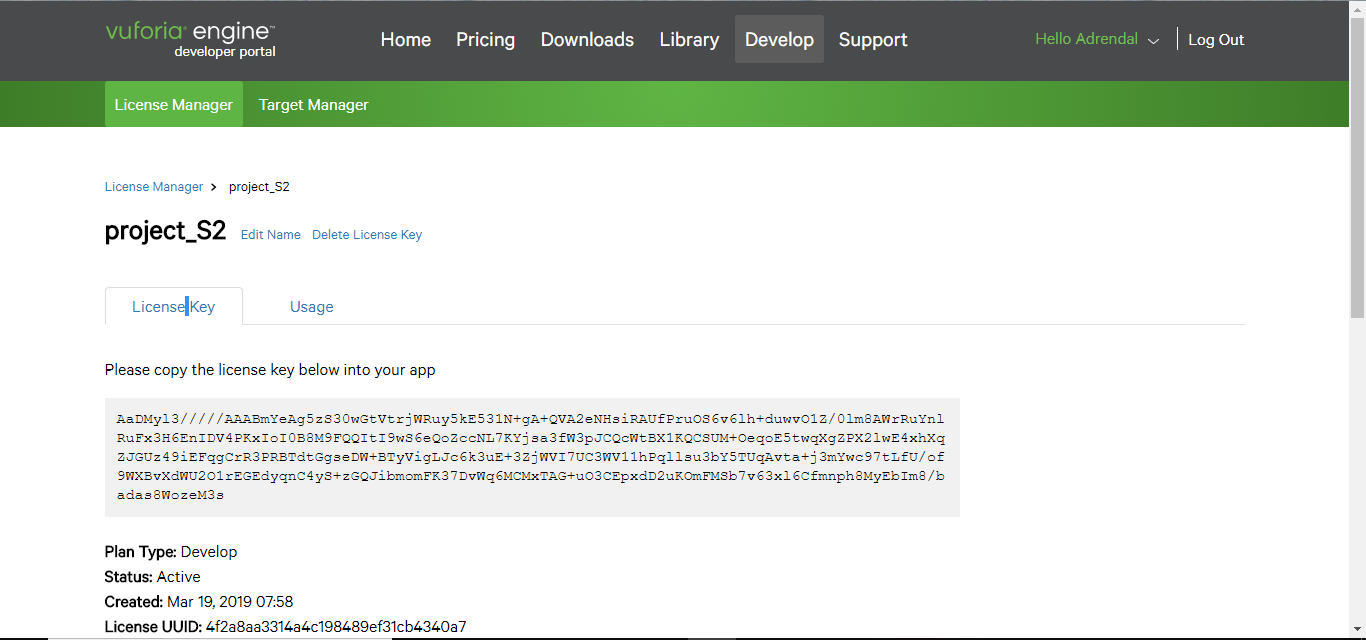
Lorsque l’utilisateur se retrouve dans la scène menu “Info canvas”, un bouton lui permet déterminer le trajet pour se rendre à la mine d’Arenberg depuis sa position.

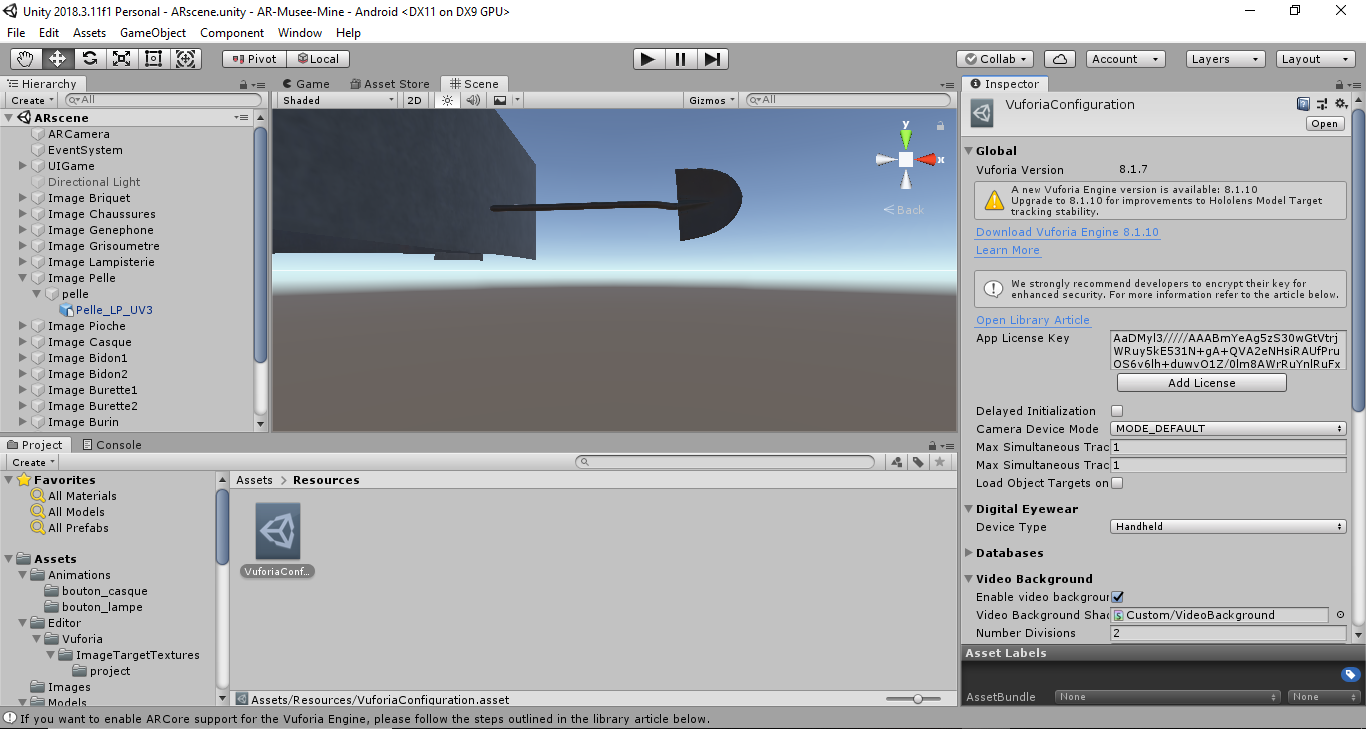
Pour cela, le script *OpenGoogleMaps* attaché au bouton demande la permission d’accès à la géolocalisation du téléphone. Une fois cette permission accordée, le script récupère la latitude et la longitude de l’utilisateur et redirige ce dernier vers l’application “Google Maps” avec un itinéraire tracé entre sa position et celle de la mine d’Arenberg.

# Création des mires et des objets 3D

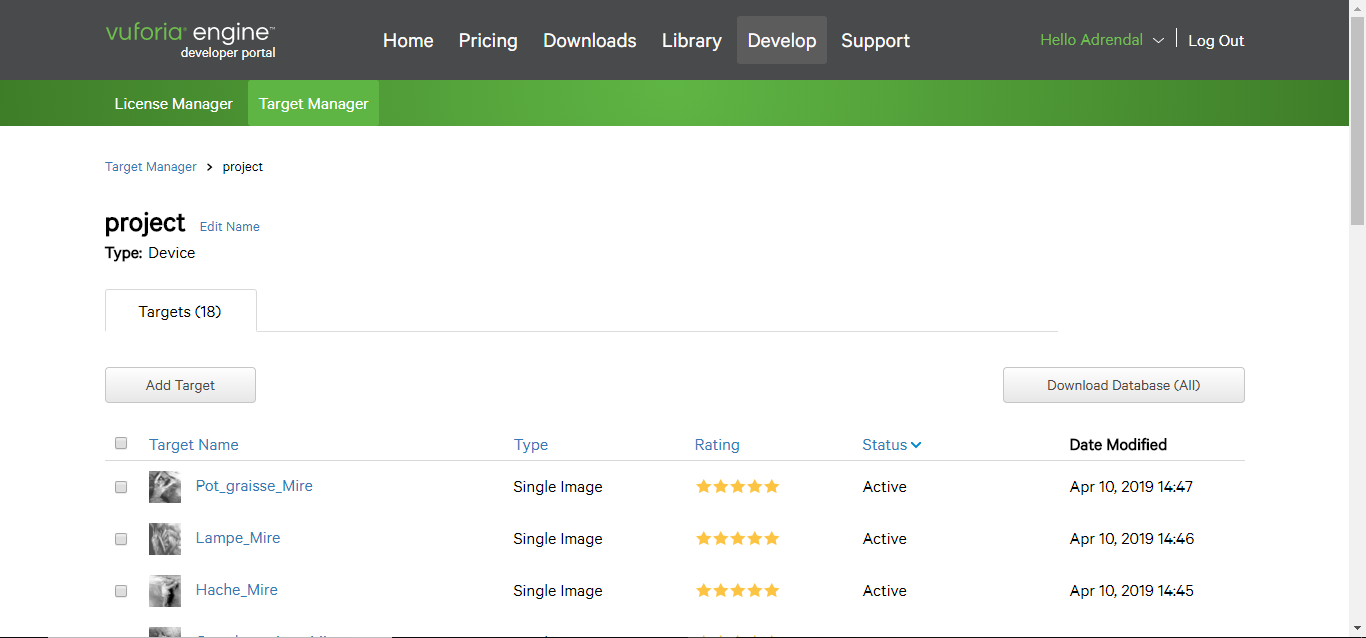
## Création de la base de données Vuforia

Pour faire de la réalité augmentée avec Vuforia, il faut associer chaque objet 3D à faire apparaître à une ImageTarget, à laquelle on joint une mire. Les mires ne peuvent pas être importées directement dans le projet et doivent être stockées dans une base de données. Pour ce faire, il faut tout d’abord créer un compte développeur sur le portail Vuforia en utilisant le lien <https://developer.vuforia.com/vui/auth/register>. Ceci nous permet dans un premier temps d’obtenir une license key.



Cette clé alphanumérique va permettre d’accéder à la base de données Vuforia que nous allons créer par la suite. La clé doit ensuite être renseignée sur Unity dans le fichier Vuforiaconfig (situé dans notre projet>>Assets >>ressources) dans la section “App license key”.

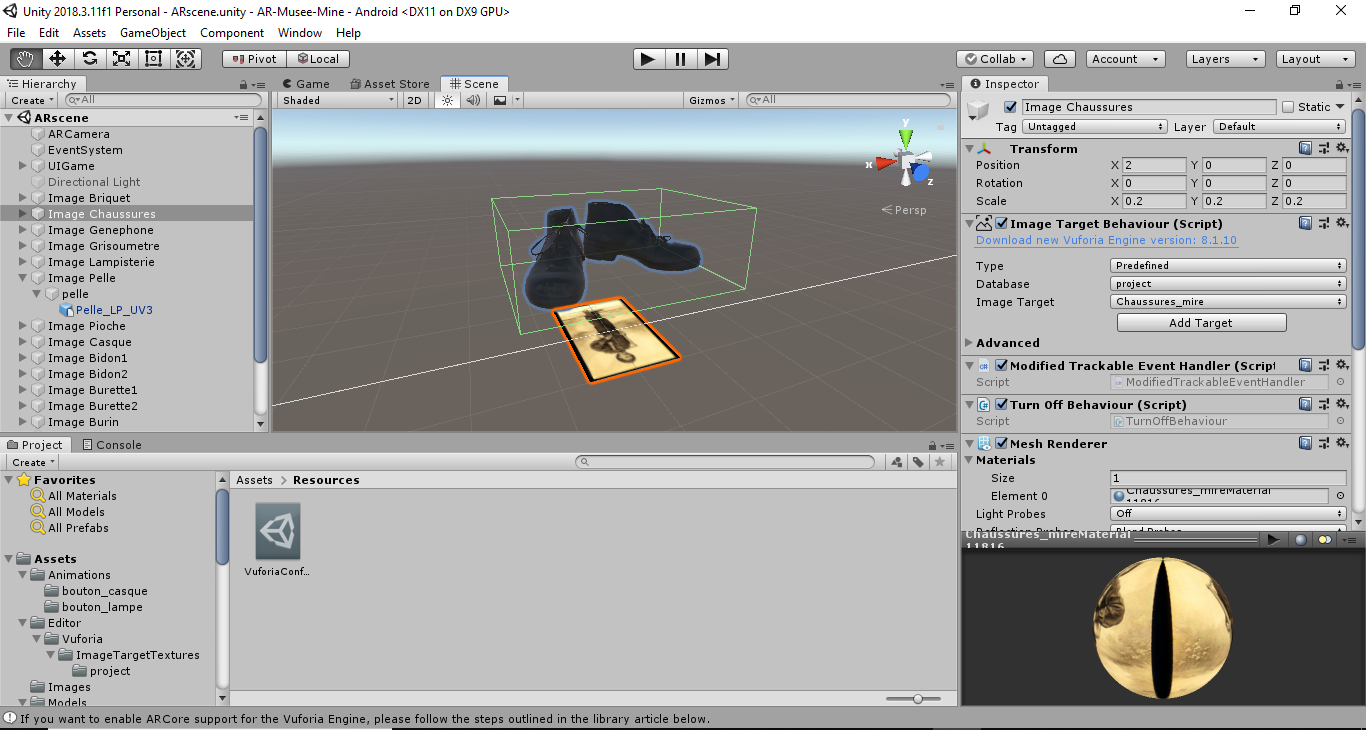
Il nous faut maintenant créer la base de données. Nous devons donc retourner sur le site de Vuforia, se connecter à notre compte, aller dans l’onglet “Target Manager” et cliquer sur *Add Database*. Après avoir donné un nom à cette base de données, nous avons la possibilité d’y ajouter des images en cliquant sur le bouton *Add target* et à téléverser les images que nous voulons ajouter à la base de données, puis de cliquer sur *Download database* et choisir le format compatible avec l’éditeur Unity.



Une fois la base de données créée, il suffit de l’importer dans les assets de notre projet Unity.

## Création des ImageTarget

On peut maintenant créer des images target dans notre scène en faisant clic-droit dans notre hiérarchie, puis Vuforia Engine, puis “image”. Pour associer l’image souhaitée à ce nouvel objet, il faut cliquer sur l’objet, aller dans l’inspecteur, partie “image target behavior”, et sélectionner la base de données que vous souhaitez, puis l’image de cette base de données que vous voulez utiliser.



Il est également nécessaire de modifier le comportement des ImageTarget en remplaçant le *DefaultTrackableEventHandler* par un script personnalisé plus adapté à nos besoins : *ModifiedTrackableEventHandler*. Celui-ci permet de garder en mémoire les noms des objets actuellement affichés à l’écran, ce qui nous permettra d’afficher la description correspondante lorsque l’utilisateur appuie sur son infobulle.

Pour simplifier l’ajout de nouveaux objets, il existe un préfab “*example AR*” présent dans les assets de notre application qui contient l’ImageTarget et l’objet associé (par défaut un cube) que vous pouvez modifier selon vos souhaits. L’ImageTarget de ce préfab contient déjà le script *ModifiedTrackableEventHandler*.

En revanche, il est nécessaire de lui associer un nom affiché et une description. Pour cela, il faut les ajouter dans la classe ObjectsConstants, dans la fonction Initialize().

La forme de la ligne à rajouter est la suivante :

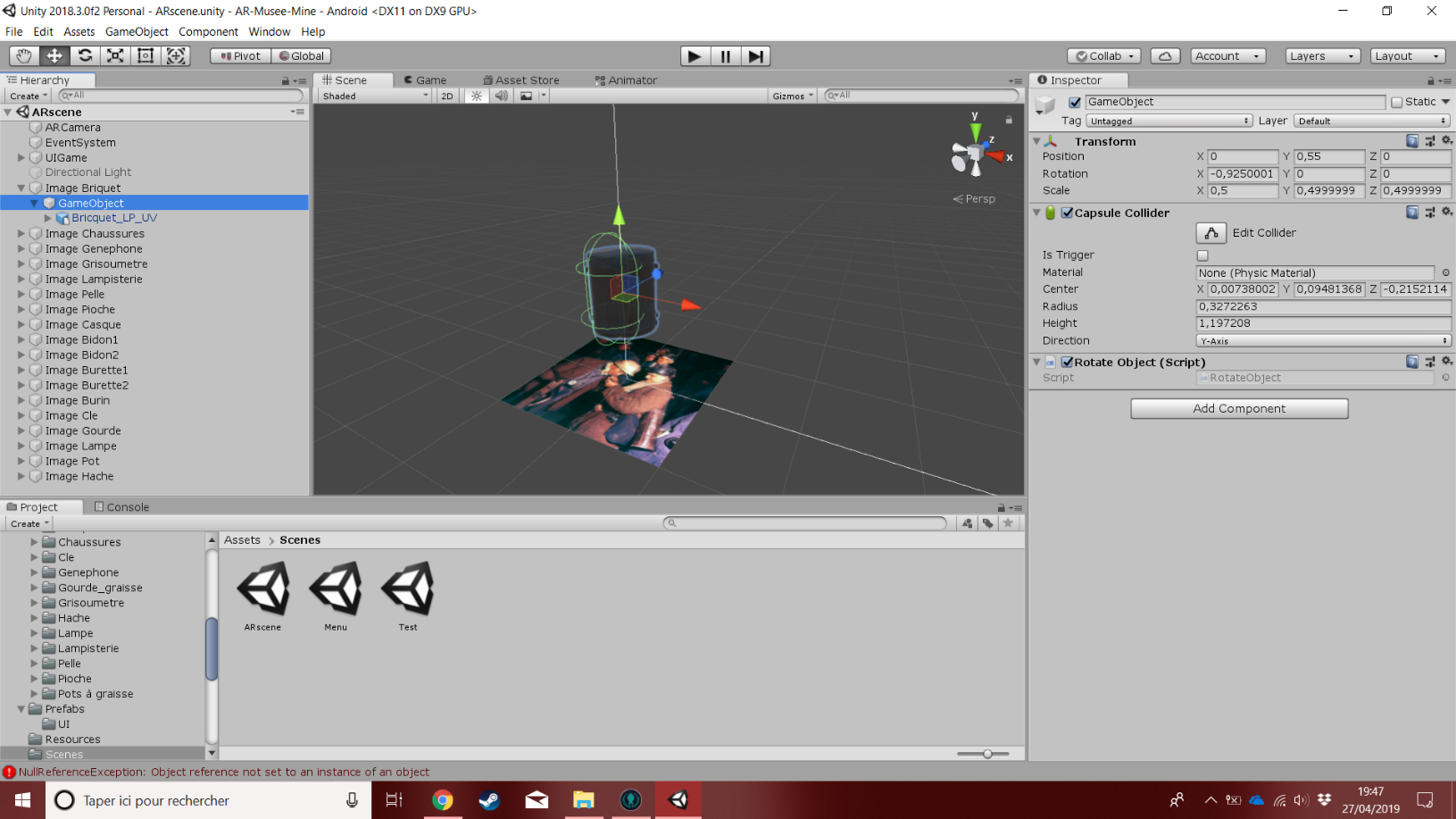
infos.Add(new Info(Nom de l’ImageTarget Unity, Nom affiché, Description));

D’ici, les descriptions des objets déjà présents dans l’application peuvent également être modifiées.

# Interactions avec les objets

## Rotation des objets

Pour les différents objets (à l’exception du généphone), il est possible d’effectuer des rotations selon deux axes pour permettre aux utilisateurs de voir l’objet sous tous ses angles. Pour ce faire, il faut tout d’abord ajouter un collider à l’objet. Ce collider peut être une ”box collider”, une ”sphere collider” ou encore une “capsule collider” selon la forme de l’objet. Il permet d’associer une zone de collision à cet objet. Nous utilisons alors la méthode “OnMouseDrag” qui permet de détecter les glissements de doigts que l’utilisateur fait en touchant un objet. En fonction de cette entrée, une rotation va être effectuée selon les axes de l’ARCamera.



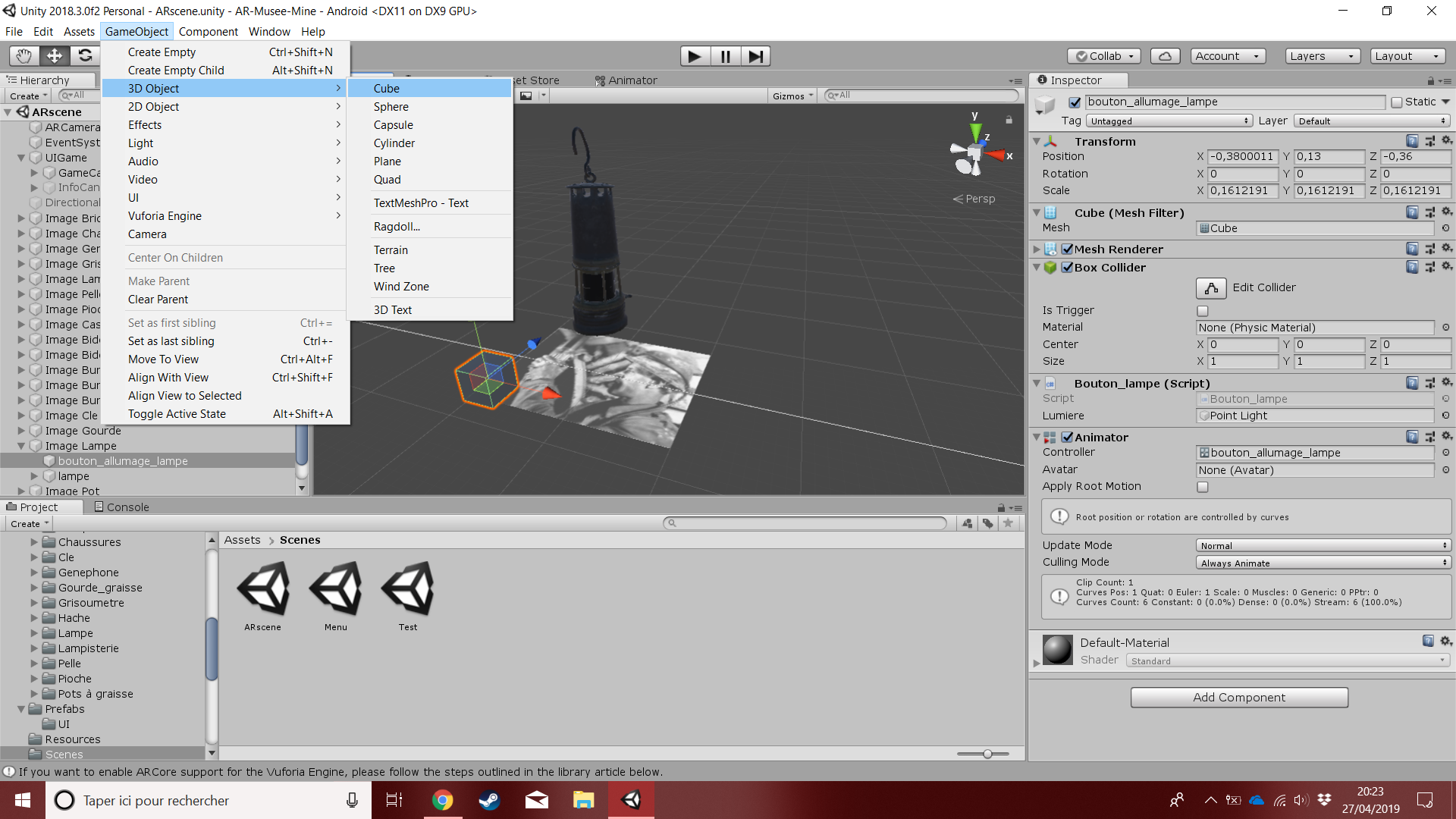
Exemple du briquet entouré d’une “capsule collider”

## Emission de lumière

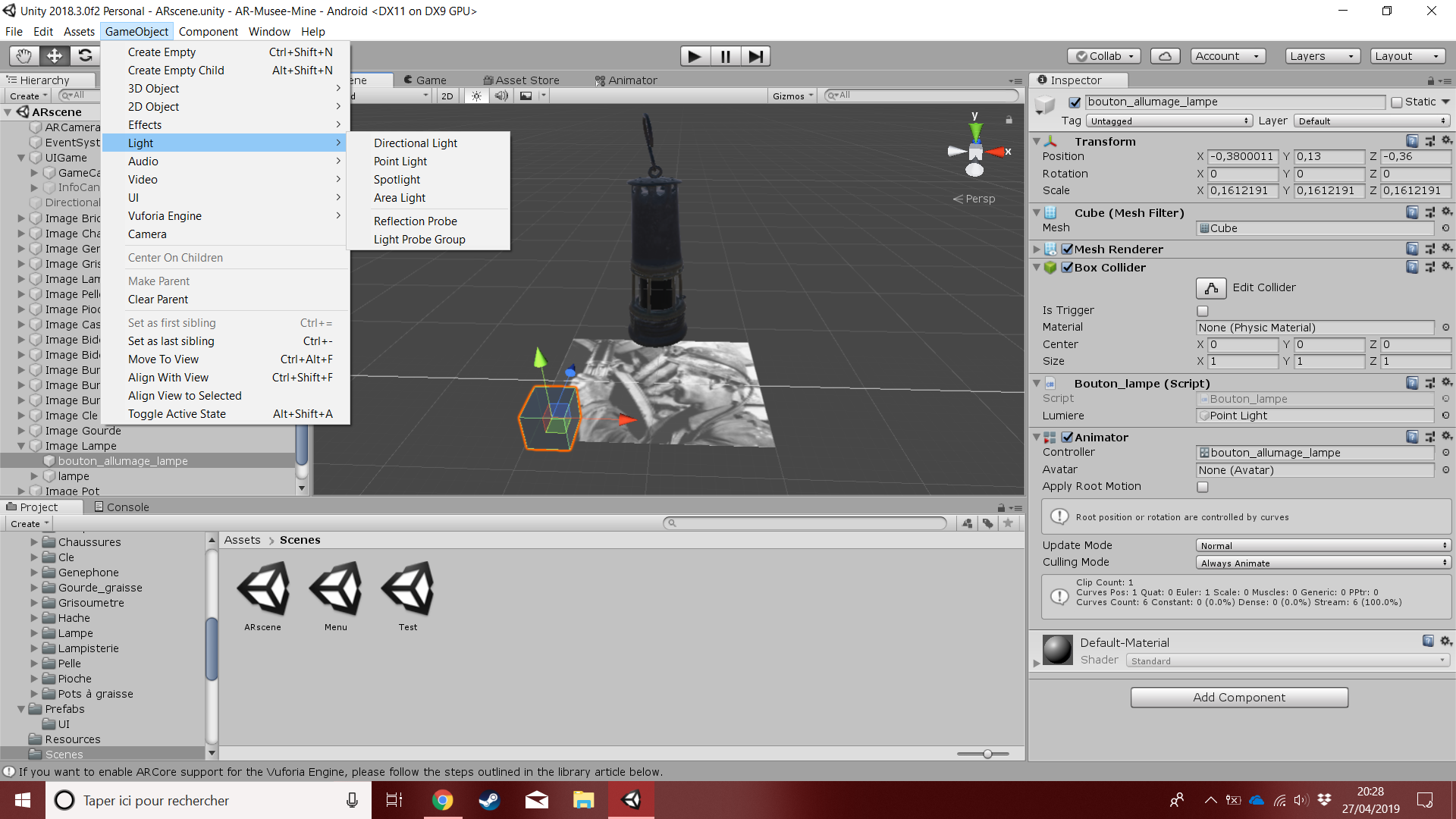
Certains objets tels que le casque ou la lampe sont capables d’émettre une lumière sur action de l’utilisateur.

Pour cela, la première étape consiste à positionner la source de lumière à l’endroit souhaité (par exemple à l’intérieur de la lampe) et choisir le type de source adéquat. Pour la lampe, il s’agit d’une lumière éclairant dans toutes les directions, c’est pourquoi il faut choisir une “Point light”.

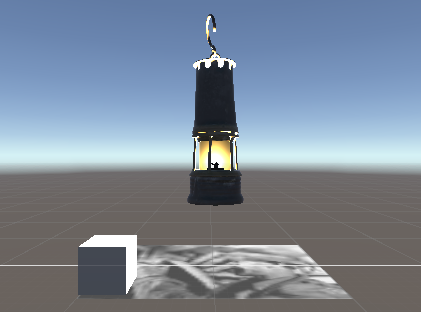
Pour allumer ou éteindre la lampe, nous nous servons d’un bouton. Ce bouton est un objet 3D (un cube dans notre cas) que l’on place dans la scène. Il est lié à la mire et apparaît et disparaît en même temps que l’objet. Le script *Bouton\_lampe* attaché à ce cube permet de modifier l’intensité de la lampe lorsqu’un clic est détecté, ce qui a pour effet de l’allumer ou de l’éteindre.



Création du cube (Il est évidemment possible de prendre un autre objet)

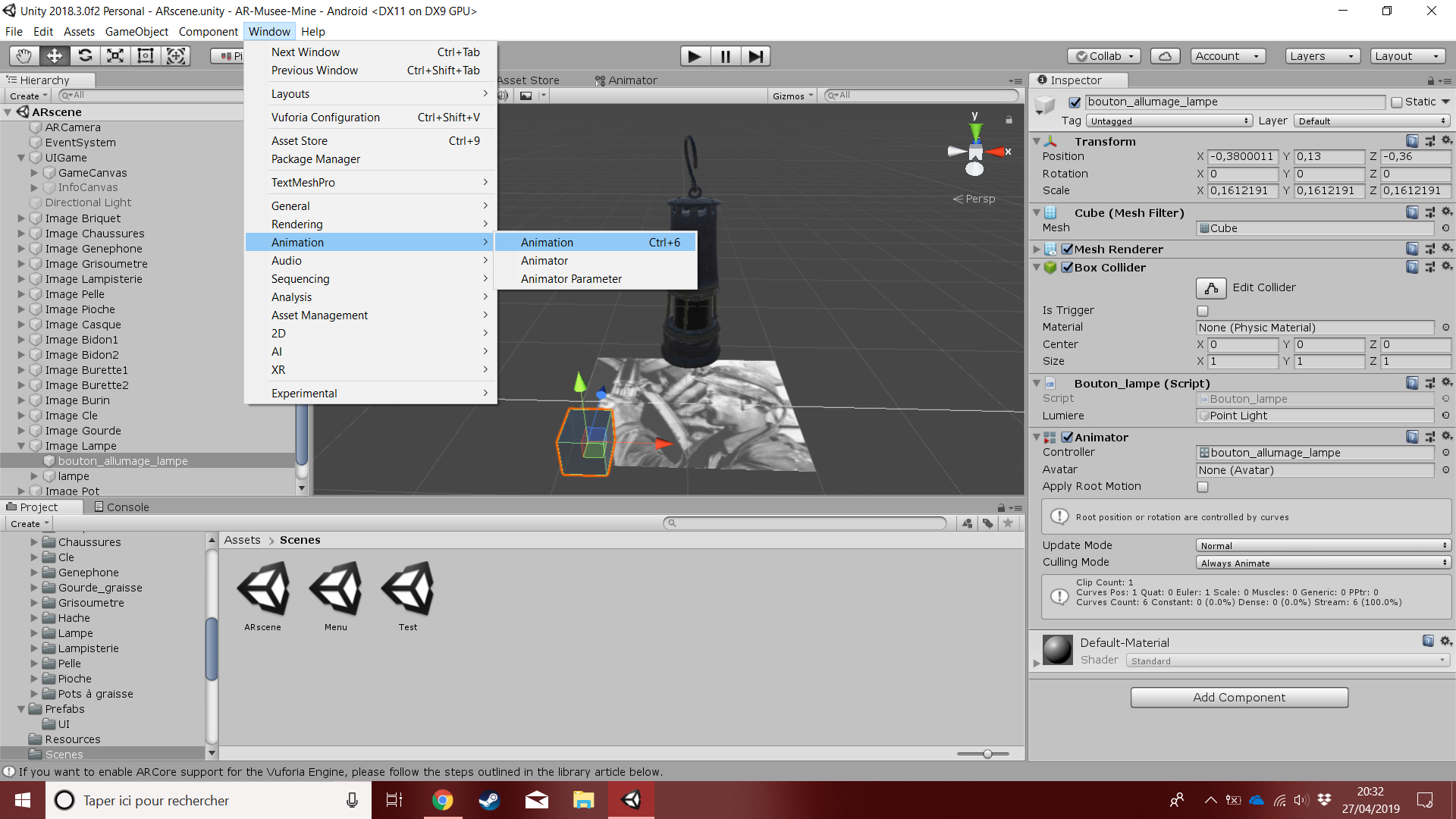


Création de la source de lumière (Ici aussi, en fonction du type d’éclairage souhaité, on a le choix)



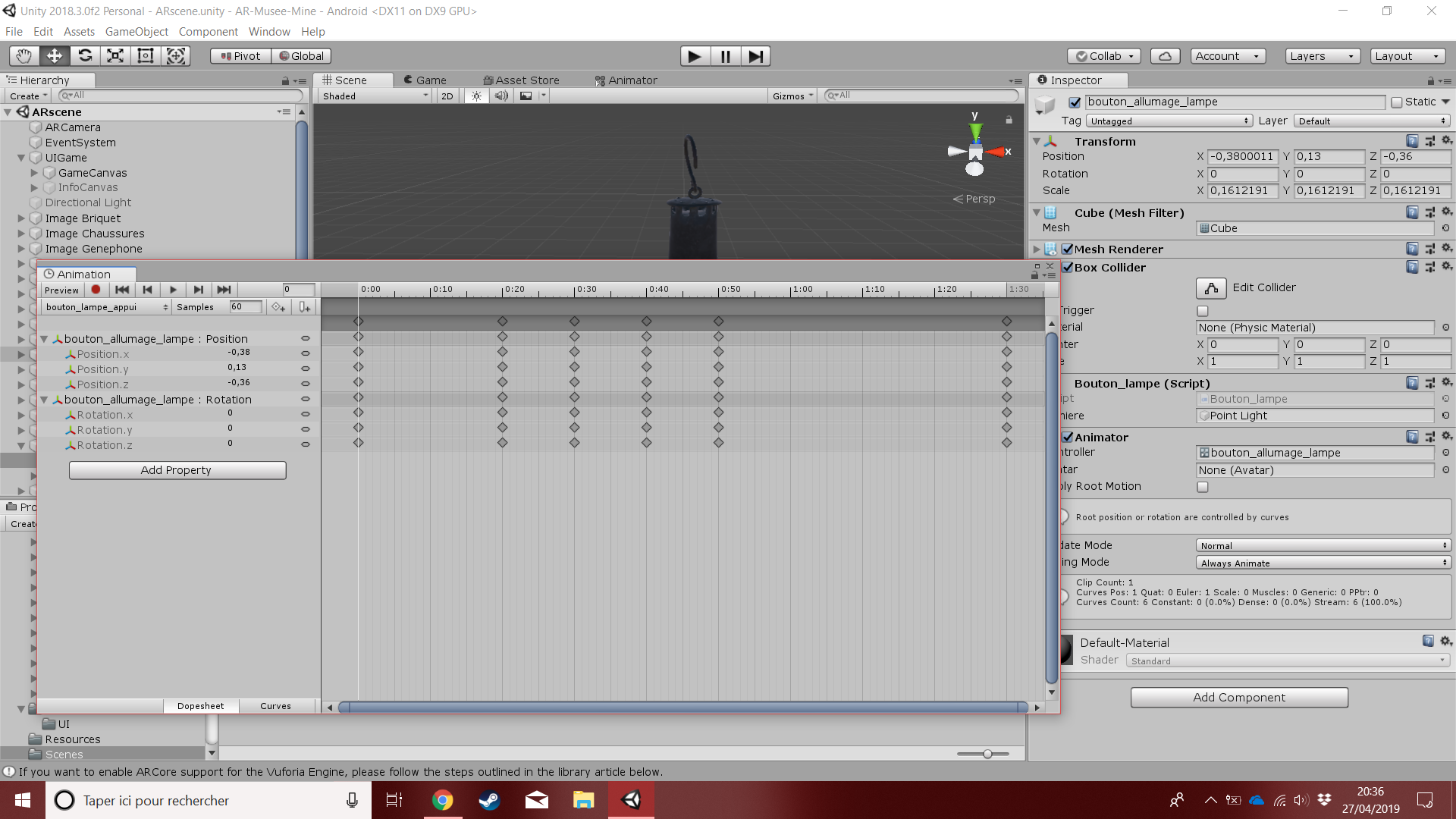
Rendu final de la lampe

Ces mêmes boutons sont par ailleurs animés afin de donner un retour visuel à l’utilisateur. Ils possèdent deux animations : une première qui correspond à leur état normal c’est à dire avant que l’utilisateur ne les presse et une deuxième qui se déclenche quand l’utilisateur appuie dessus. Pour ajouter une animation, cela se fait en sélectionnant l’objet à animer puis en allant dans l’onglet “Window” puis Animation.



Onglet animation

Une fenêtre apparaît alors. Dans celle-ci il suffit de créer une nouvelle animation. Après cela, on peut choisir les paramètres à animer. La plupart du temps il s’agit de paramètres de type “Transform” tel que la position et la rotation.

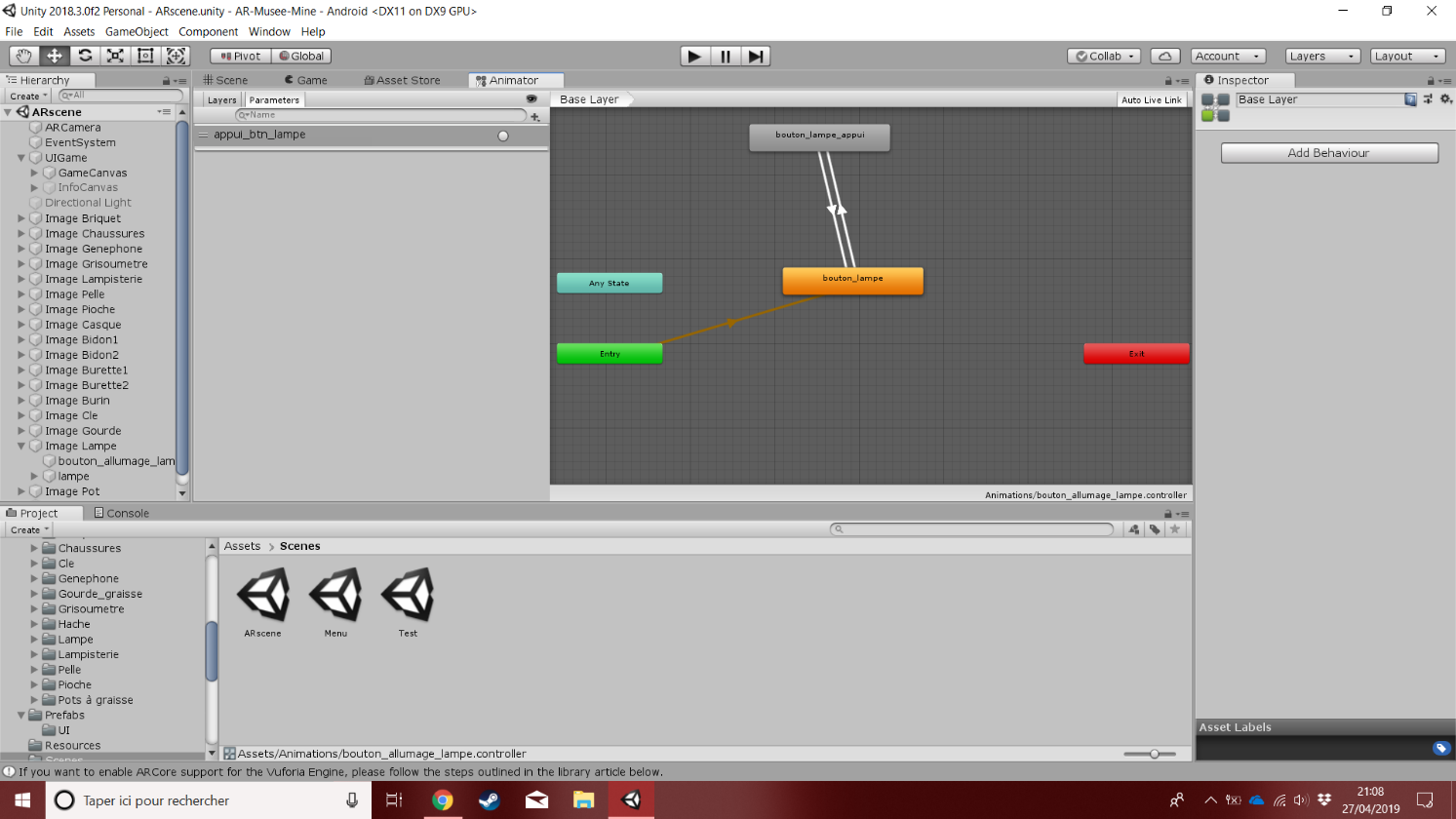


Paramètres pour l’animation de nos boutons.

Pour créer l’animation il suffit alors de cliquer sur le bouton rouge visible sur l’image ci-dessus. En cliquant, on commence l’enregistrement.

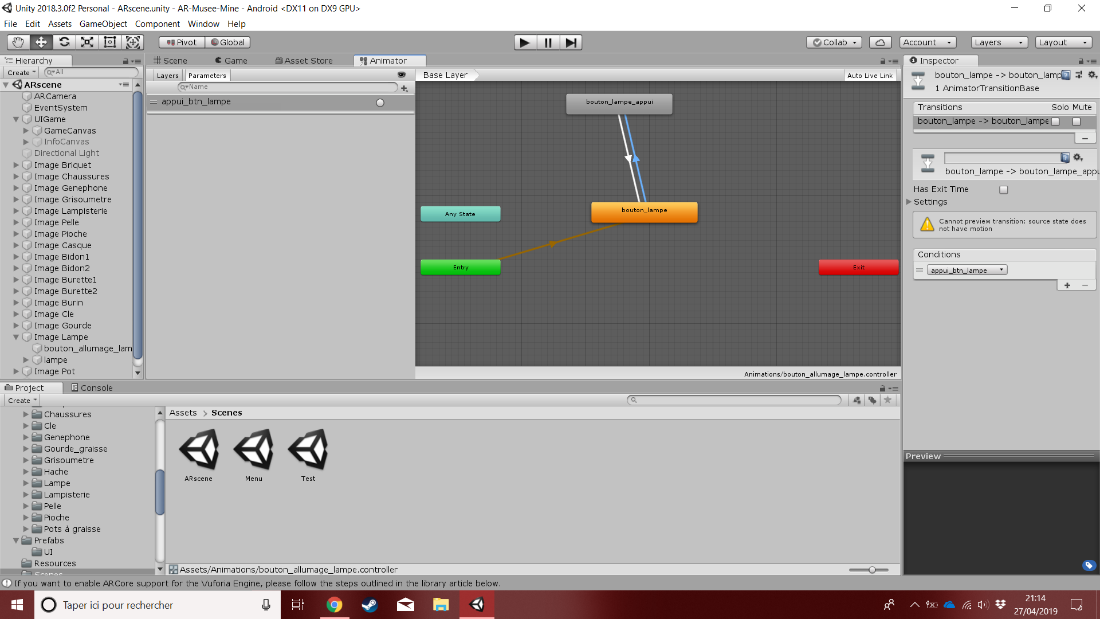


Les losanges gris représentent chacun la valeur d’un paramètre de l’objet à animer. La bande bleue en haut à droite représente le temps. Il suffit alors de se positionner sur le temps voulu puis de bouger l’objet pour le mettre à l’emplacement voulu à cet instant. La transition entre les différentes positions se fait alors toute seule. Il est possible de visualiser l’animation à tout moment en appuyant sur le bouton “play”. Nous pouvons augmenter ou diminuer la durée et également choisir le nombre d’images par seconde de l’animation. Un même objet peut avoir plusieurs animations. C’est d’ailleurs le cas de nos boutons et on doit donc ordonner ces animations. C’est là qu’intervient “l’animator”.



Fenêtre Animator

Les blocs “Entry”, “Any State” et “Exit seront déjà présent. En plus de ces blocs, les animations créées seront également présentes sous forme de blocs. Il suffit alors de faire les liaisons souhaitées de bloc à cloc. On ajoute ensuite une condition de changement d’état. Ici, nous avons ajouté un paramètre “appui\_btn\_lampe” qui est de type trigger. Au lancement, on passe du bloc “Entry” à celui de la première animation puis en cas d’appui sur le bouton, on déclenche la variable “appui\_btn\_lampe” ce qui a pour effet de passer au bloc suivant.



Condition de changement d’animation.

# LightMatching

La couleur ambiante de la scène Unity (et donc celle des objets) est automatiquement adaptée à celle du flux vidéo capturé par la caméra grâce au script *LightMatching*. Ce script est attaché à l’ARCamera. Le taux de rafraichissement de la couleur est fixé à 1 seconde mais peut être modifié depuis l’inspecteur Unity. Ce script est cependant nécessité une quantité conséquente de ressources et pourrait ralentir significativement l’application sur les appareils n’étant pas assez puissants.

Pour diminuer l’impact de cette limite, il serait bien venu de modifier le script pour remplacer le format d’image actuel de la caméra (RGB888) par un format plus compact (RGB565 par exemple). Il serait également possible de rajouter une option pour laisser l’utilisateur choisir d’activer ou non cette fonctionnalité.