DIVERSITY by EPITECH

ARCOBRA

MON PREMIER FILTRE EN AR

ARCOBRA

Pour cette activité vous allez utiliser l'interface **Android Studio** avec le langage **Java**.

Si vous voulez installer l'application que vous allez faire sur votre propre téléphone **Android**, demandez à un Cobra de vous aider à le faire !

N'hésitez pas à soliciter les *Cobras* présents pour toute autre aide durant l'activité.

Introduction

Vous êtes un développeur indépendant et vous souhaitez sortir votre propre filtre *Snapchat/Instagram* grâce au nouveau kit de développement de Google : **ARCore** !
Vous allez donc être guidé à travers cette activité pour utiliser des *objets 3D* tout en trackant un visage à travers la caméra.

Installation (à faire par un Cobra)

Android Studio

Allez sur `https://developer.android.com/studio` et téléchargez Android Studio puis désarchivez le téléchargement.
Déplacez le dossier `android-studio` à la racine (`/home/[user_name]`)
Ouvrez un terminal au dossier, lancez : `./bin/studio.sh` et installez Android Studio

Google ARCore pour Émulateur

Allez sur `https://github.com/google-ar/arcore-android-sdk/releases` et téléchargez `Google_Play_Services_for_AR_X.XX.X_x86_for_emulator.apk`.

Projet ARCobra

Allez sur `https://github.com/nicklamyeeman/ArCore-Cobra`, désarchivez le téléchargement et ouvrez le projet dans Android Studio.

Modifiez le fichier `Graddle Scripts > local.properties` si Android Studio ne vous le propose pas pour relocaliser le SDK d'Android (`/home/[user_name]/Android/Adk`)

Installer l'émulateur d'Android Studio

Dans ***Android Studio > Tools > Device Manager*** créez un nouveau device.

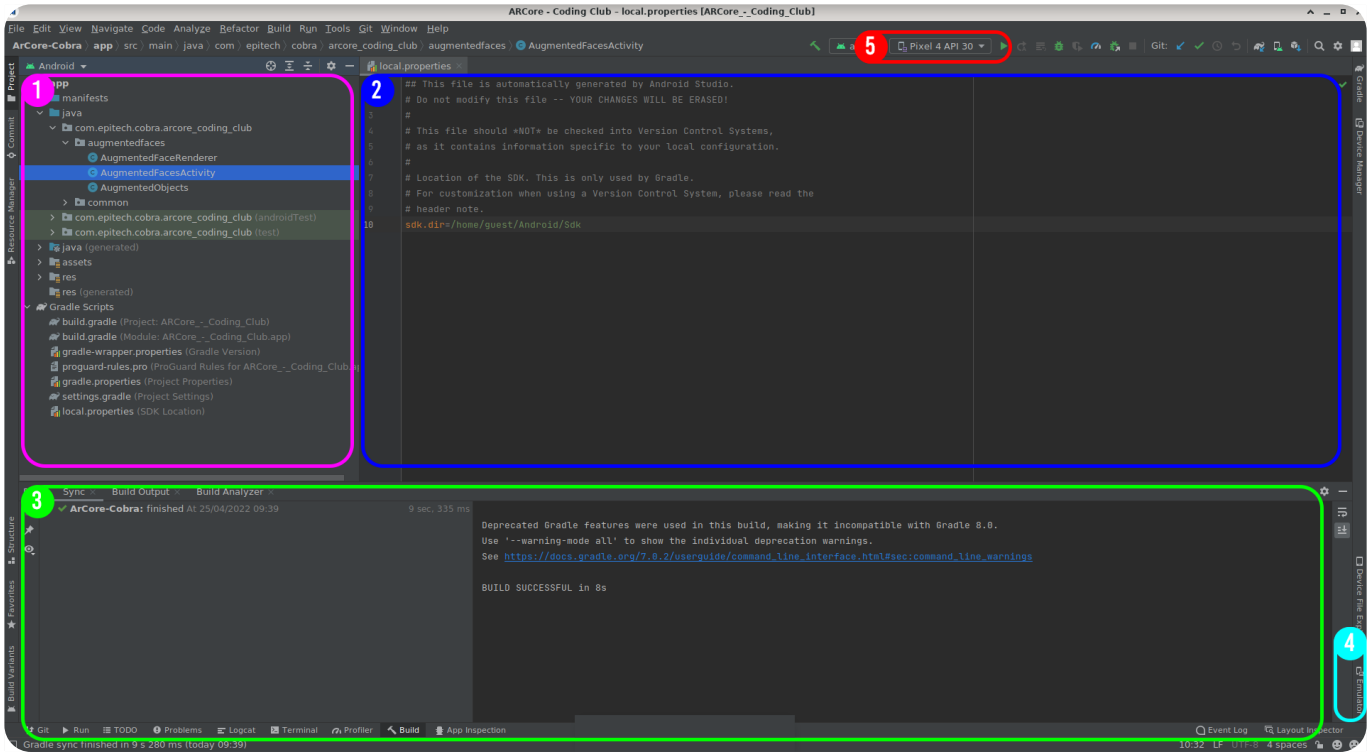
API Level 24 minimum. Par défaut : Pixel 4 API 30
Attention : Lorsque vous sélectionnez une image système, allez bien dans l'onglet x86
Affichez les paramètres avancés et sélectionnez `webcam0` pour la caméra frontale

Lancez l'émulation et glissez (*Drag & drop*) l'apk sur l'émulateur pour installer Google Services for AR

Android Studio

L'interface

Bienvenue sur **Android Studio**, un environnement de travail pour tous les développeurs Android.
Voici une petite description de l'interface :



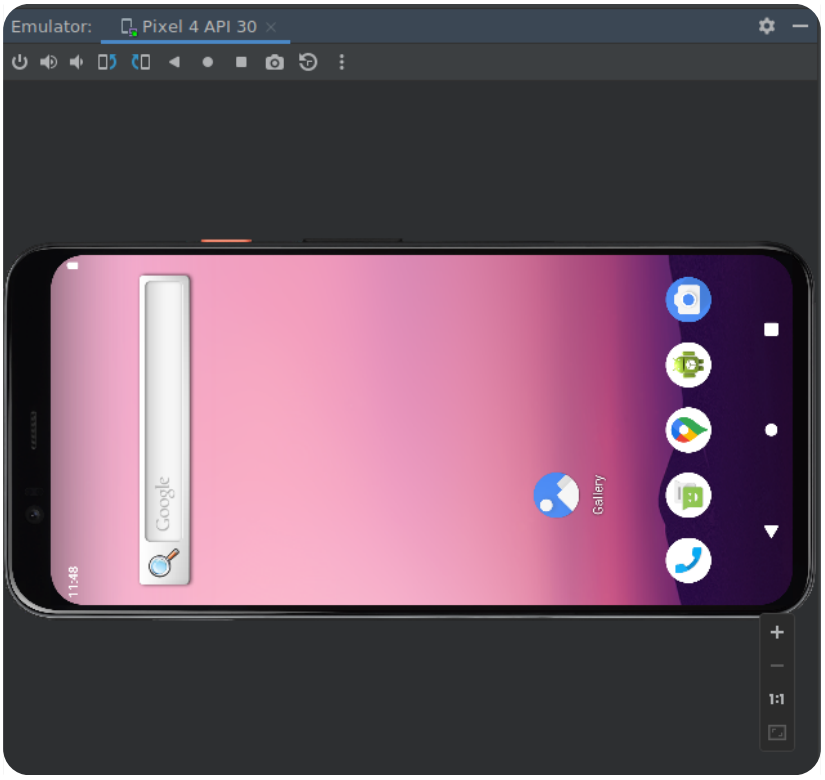
1. Ceci est ce qu'on appelle ***l'arborescence*** du projet, c'est là où se trouvent tous les fichiers utiles à la création de l'application Android
2. Là vous trouverez ***l'éditeur de code***, c'est ici que vous allez écrire du code
3. Ici se trouvent différentes fenêtres utiles, comme la fenêtre ***Build*** qui permet de suivre la compilation ou la fenêtre ***Logcat*** qui permet de voir les messages des développeurs pendant que l'application est active

La compilation c'est la transformation d'un code lisible par un humain en une application lisible par un ordinateur

4. Ce petit bouton permet d'ouvrir la fenêtre de ***l'émulateur*** Android
5. Et celui-ci permet de ***lancer l'application*** sur l'émulateur (Pixel 4 API 30) ou sur votre smartphone si vous l'aviez demandé

L'émulateur

Si vous lancez l'émulateur sur Android Studio, faites en sorte de l'orienter en paysage comme ci-dessous.



Étape 1 : Créer un endroit où stocker les objets en réalité augmentée

Pour toutes les étapes, vous n'aurez à modifier qu'un seul fichier : **AugmentedFacesActivity**.

Toutes les étapes sont marquées dans ce fichier, il suffit de trouver les blocs de commentaires correspondant à l'étape

Les commentaires et blocs de commentaires sont des lignes qui ne sont pas lues par le programme.
Les blocs de commentaires commencent par :

```
/**
 *  STEP 1 : [...]
 */
```

et se terminent par :

```
/**
 *
 */
```

Cette ***première étape*** sert d'introduction très rapide au Java.

Le fichier que vous modifiez s'appelle une ***Classe***.

```
public class AugmentedFacesActivity [...]
```

Cette ***classe*** permet d'afficher le contenu de l'application. Pour cela elle a besoin de ressources qu'on peut stocker dans des *variables* qu'on appelle des ***attributs***.

```
[...]
private GLSurfaceView surfaceView;
private boolean installRequested;
private Session session;
private final SnackbarHelper messageSnackbarHelper = new SnackbarHelper();
[...]
```

Nous allons donc créer notre propre attribut pour y mettre tous les objets en Réalité Augmentée. Basons nous sur l'attribut ***SnackbarHelper*** :

```
/**    options        classe          nom de l'attribut        initialisation    **/
    private final SnackbarHelper messageSnackbarHelper = new SnackbarHelper();
```

En prenant exemple sur l'initialisation de l'attribut juste au dessus nous allons créer notre attribut avec les informations suivantes :

- ✓ options : `private final`
- ✓ classe : `AugmentedObjects`

```
private final AugmentedObjects augmentedObjects = new AugmentedObjects();
```

Étape 2 : Créer les objets en réalité augmentée

Maintenant que nous avons créé un espace où ranger nos objets, nous allons les ***créer***.

À l'étape 1 vous avez donné un **nom** à votre attribut, celui-ci est très important parce que c'est ce nom que nous allons utiliser tout au long du programme.

```
                                    /** nom de **/
private final AugmentedObjects augmentedObjects = new AugmentedObjects();
                                /** l'attribut **/
```

Grâce à ce nom vous pourrez accéder aux ***méthodes*** de la classe `AugmentedObjects`

Une **méthode** est une instruction qui éxecute une action précise.
Par exemple, dans les jeux vidéos, les classes `Hero` ont généralement des méthodes pour se déplacer ou pour se battre.

Ici les ***méthodes*** seront écrites de la manière suivante :

```
nom_attribut.nom_methode(option1, option2, ...);
```

Voici donc les informations nécessaires pour créer vos objets :

- ✓ nom de l'attribut : `augmentedObjects` *(dépendant de comment vous l'aviez appelé au tout début)*
- ✓ nom de la méthode : `addObject()`
- ✓ options :
  - `context` : *le contexte de l'application. Ici ce sera toujours* `this`
  - `name` : *le nom de l'objet que vous voulez. Exemple :* `"nez"`
  - `objName` : *le chemin de l'objet. Exemple :* `"models/objects/nose.obj"`
  - `materialName` : *le chemin de la texture. Exemple :* `"models/materials/nose_fur.png"`
  - `area` : *l'endroit où placer l'objet sur le visage*

Vous manipulerez des objets et des textures déjà conçus pour l'activité.
Dans l'arborescence de l'interface, vous trouverez les objets dans le dossier
- ***assets > models > objects***
et les textures dans le dossier
- ***assets > models > materials***

Il n'existe que 3 endroits possibles sur le visage :

```
/** En haut a gauche **/
AugmentedObjects.FACE_AREA.FOREHEAD_LEFT

/** En haut a droite **/
AugmentedObjects.FACE_AREA.FOREHEAD_RIGHT

/** Au centre **/
AugmentedObjects.FACE_AREA.CENTER
```

```
/** Exemple creation d'un objet sur le nez **/
    augmentedObjects.addObject(
        this,
        "nez",
        "models/objects/nose.obj",
        "models/materials/nose_fur.png",
        AugmentedObjects.FACE_AREA.CENTER
    );
```

BONUS : Modifier le masque du visage

Cette étape est facultative

Vous aurez surement remarqué le bloc de commentaire “BONUS”

```
/**
 *    BONUS : Changement du masque sur le visage
 */
```

La ligne de code juste en dessous permet de mettre un masque sur le visage.
Par défaut, la texture est transparente : `"models/meshes/empty.png"`

À vous d’essayer de modifier cette texture !

Étape 3 : Afficher les objets en temps réel

Si vous lancez l'application maintenant dans votre émulateur ou sur votre appareil Android, vous n'aurez **aucun** objet d'affiché.

Si vous avez fait l'étape bonus, vous devriez en revanche voir le changement

Avant d'afficher chacun des objets *1 par 1* il y a **deux** lignes à ajouter au début et à la fin :

```
augmentedObjects.updateObjectsMatrix(face);

// [...] L'affichage de vos objets

augmentedObjects.drawObjects(viewMatrix, projectionMatrix, colorCorrectionRgba,
    DEFAULT_COLOR);
```

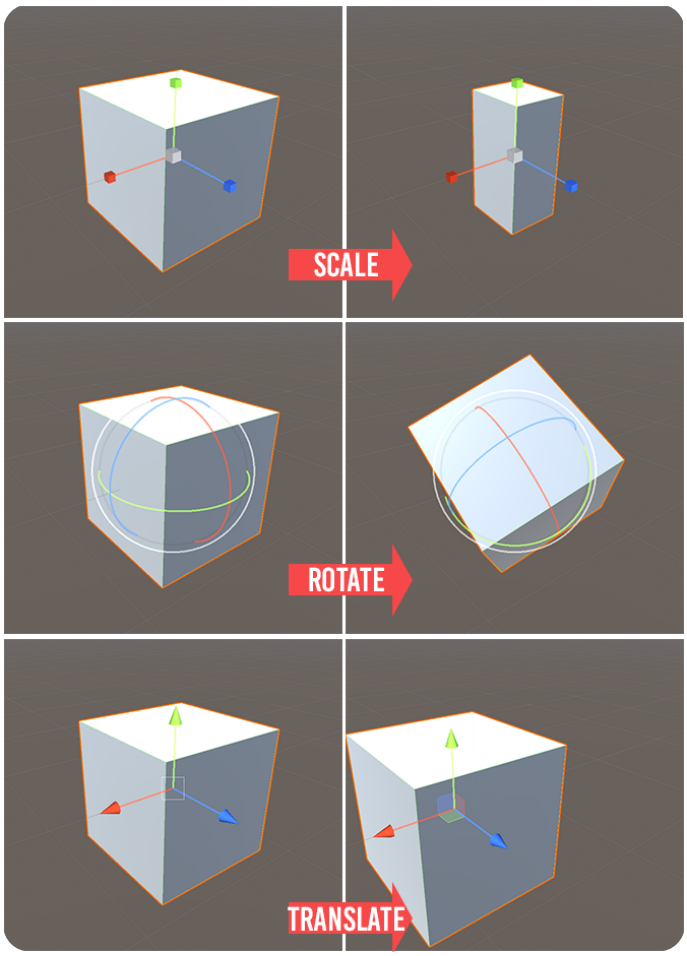
Ces deux lignes sont déjà écrites mais sont en commentaire. Il suffit d'enlever les «**//**» au début de la ligne pour que le programme puisse lire la ligne de code.

Les transformations d'objets 3D

Avant d'entrer dans le vif du sujet, nous allons voir quelques notions.

Chaque objet 3D est défini dans l'espace par ce qu'on appelle une **matrice**.
Cette **matrice** contient toutes les informations nécessaires à un objet (taille, position, angle, etc…)

Les **matrices** peuvent donc être **transformée** en suivant ***3 grandes catégories*** :
- **Scale** *(ou échelle)* : transformation de la taille de l'objet
- **Rotation** *(ou pivot)* : transformation de l'orientation de l'objet dans l'espace
- **Translation** : transformation de la position de l'objet dans l'espace



Ces transformations sont définies dans un espace 3D, donc avec des valeurs sur les 3 axes de l'espace : `x`, `y`, `z`

Pour afficher nos objets il faut **obligatoirement** les transformer :

- ✓ nom de l'attribut : `augmentedObjects` *(dépendant de comment vous l'aviez appelé au tout début)*
- ✓ nom de la méthode : `updateObject()`
- ✓ options :
  - `name` : *le nom de l'objet que vous avez donné précédemment. Exemple :* `"nez"`
  - `scaleFactor` : *les valeurs de scale de l'objet. Exemple :* `new float[]{ 0.5f, 2.0f, -2.0f }`
  - `rotateFactor` : *les valeurs de rotation de l'objet. Exemple :* `new float[]{ 45.0f, 1.0f, 0.0f, 0.0f }`
  - `translateFactor` : *les valeurs de translation de l'ojet. Exemple :* `new float[]{ 0.5f, 0.0f, 0.8f }`

Pour ***scale*** un objet il faut donner les valeurs en **x**, **y** et **z**.
Ces 3 valeurs servent de *multiplicateur* sur les axes respectifs.
Par exemple si on souhaite doubler la taille de l'objet uniquement sur l'axe x :

```
new float[]{ 2.0f, 1.0f, 1.0f }
```

Ou si on souhaite diviser par 10 l'objet (parce qu'il est 10 fois trop grand) :

```
new float[]{ 0.1f, 0.1f, 0.1f }
```

Une valeur négative renversera l'objet, ce sera comme le regarder dans un miroir

Pour ***rotate*** un objet, il faut donner **l'angle** de rotation et les valeurs en **x**, **y** et **z**
Par exemple si on souhaite pivoter l'ojet de 45° sur l'axe des Y :

```
new float[]{ 45.0f, 0.0f, 1.0f, 0.0f }
```

Ou si on souhaite retourner l'objet de 90° sur deux axes :

```
new float[]{ 90.0f, 1.0f, 0.0f, 1.0f }
```

Les valeurs en x, y et z sont toujours ***0.0f*** ou ***1.0f***

Pour ***translate*** un objet, il faut donner les valeurs en **x**, **y** et **z**
Par exemple si on souhaite déplacer l'objet vers le haut de 10 unités :
`new float[]{ 0.0f, -10.0f, 0.0f }`
Ou si on souhaite déplacer l'objet en bas à droite tout en le reculant :
`new float[]{ -10.0f, 5.0f, -4.0f }`

Les valeurs sont des déplacement, pas la position dans l'espace. Si on déplace un objet avec `new float[]{ 0.0f, 0.0f, 0.0f }` celui ci ne sera pas au centre de l'écran mais se déplacera de 0 sur l'axe X, 0 sur l'axe Y et 0 sur l'axe Z. En gros *il ne bougera pas*.

Si on ne souhaite pas modifier l'objet, il suffit de mettre les valeurs par défaut.

Les options par défaut sont :
- ***scaleFactor*** : `new float[]{1.0f, 1.0f, 1.0f}`
- ***rotateFactor*** : `new float[]{0.0f, 1.0f, 0.0f, 0.0f}`
- ***translateFactor*** : `new float[]{0.0f, 0.0f, 0.0f}`

```
augmentedObjects.updateObject(
    "nez",
    new float[]{1.0f, 1.0f, 1.0f},
    new float[]{0.0f, 1.0f, 0.0f, 0.0f},
    new float[]{0.0f, 0.0f, 0.0f}
);
```

Si vous voulez utiliser les valeurs par défaut vous pouvez également utiliser `null`

```
augmentedObjects.updateObject(
    "nez",
    null,
    null,
    null
);
```

Étape 4 : À vous de jouer !

Maintenant que vous avez toutes les cartes en main, votre mission est d'afficher :

- ✓ Une oreille gauche de renard (`forehead_left.obj` + `ear_fur.png`)
- ✓ Une oreille droite (`forehead_rightt.obj` + `ear_fur.png`)
- ✓ Un nez (`nose.obj` + `nose_fur.png`)

Avec les valeurs par défaut.

Pour aller plus loin

Si vous souhaitez jouer avec les objets restants, essayer de faire :
- CODING : *en haut à gauche*
- CLUB : *en haut à droite*
- EPITECH : *au centre*

N'hésitez pas à tout essayer et de vous amuser avec les objets et les textures qui existent

Il existe plusieurs ***méthodes*** (en plus de la méthode `updateObject(...)`) que vous pouvez utiliser pour ajouter des transformation sur vos objets.

```
augmentedObjects.scaleObject(name, scaleFactor);
augmentedObjects.rotateObject(name, rotateFactor);
augmentedObjects.translateObject(name, translateFactor);
augmentedObjects.scaleRotateObject(name, scaleFactor, rotateFactor);
augmentedObjects.scaleTranslateObject(name, scaleFactor, translateFactor);
augmentedObjects.rotateTranslateObject(name, rotateFactor, translateFactor);
```

DIVERSITY
by EPITECH