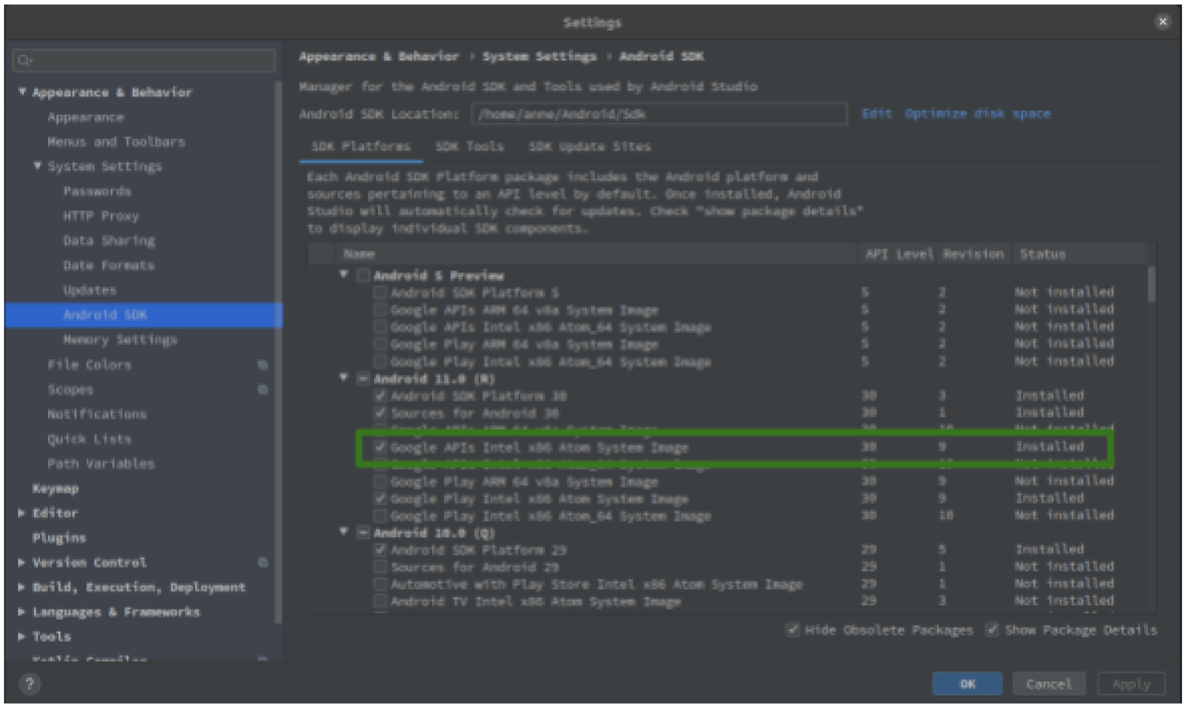
# Creating a Basic AR App

Don't forget to run your app and check your progress after each part! We don't have to finish this tutorial in our time; as long as we get through Step 2, you'll have a taste of Android AR. After that, each step will just add on to create a more complex app.

## Part 1 - Setup, Dependencies, and Permissions

[https://developers.google.com/ar/develop/java/enable-arcore](https://developers.google.com/ar/develop/java/enable-arcore)

1. Start with an Empty Activity.
2. Check settings and AVD setup



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

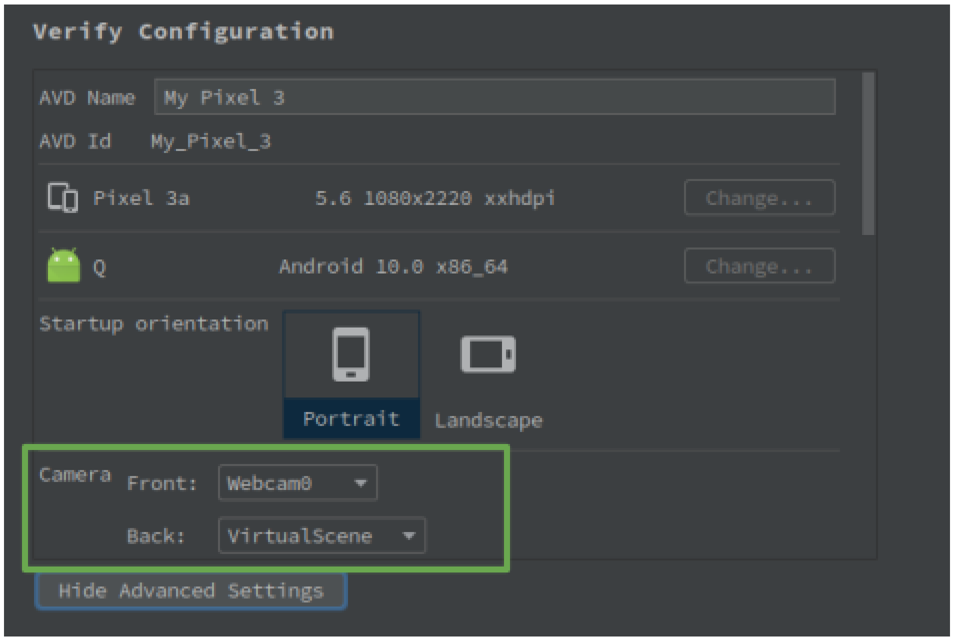
Android SDK Location: /home/anne/Android/Sdk   Edit   Optimize disk space

SDK Platforms   SDK Tools   SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

| Name | API Level | Revision | Status |
|---|---|---|---|
| **Android S Preview** | | | |
| Android SDK Platform S | S | 2 | Not installed |
| Google APIs ARM 64 v8a System Image | S | 2 | Not installed |
| Google APIs Intel x86 Atom_64 System Image | S | 2 | Not installed |
| Google Play ARM 64 v8a System Image | S | 2 | Not installed |
| Google Play Intel x86 Atom_64 System Image | S | 2 | Not installed |
| **Android 11.0 (R)** | | | |
| ✓ Android SDK Platform 30 | 30 | 3 | Installed |
| ✓ Sources for Android 30 | 30 | 1 | Installed |
| Google APIs ARM 64 v8a System Image | 30 | 10 | Not installed |
| ✓ Google APIs Intel x86 Atom System Image | 30 | 9 | Installed |
| Google APIs Intel x86 Atom_64 System Image | 30 | 10 | Not installed |
| Google Play ARM 64 v8a System Image | 30 | 9 | Not installed |
| ✓ Google Play Intel x86 Atom System Image | 30 | 9 | Installed |
| Google Play Intel x86 Atom_64 System Image | 30 | 10 | Not installed |
| **Android 10.0 (Q)** | | | |
| ✓ Android SDK Platform 29 | 29 | 5 | Installed |
| Sources for Android 29 | 29 | 1 | Not installed |
| Automotive with Play Store Intel x86 Atom System Image | 29 | 1 | Not installed |
| Android TV Intel x86 Atom System Image | 29 | 3 | Not installed |

✓ Hide Obsolete Packages   ✓ Show Package Details



Verify Configuration

AVD Name: My Pixel 3

AVD Id: My_Pixel_3

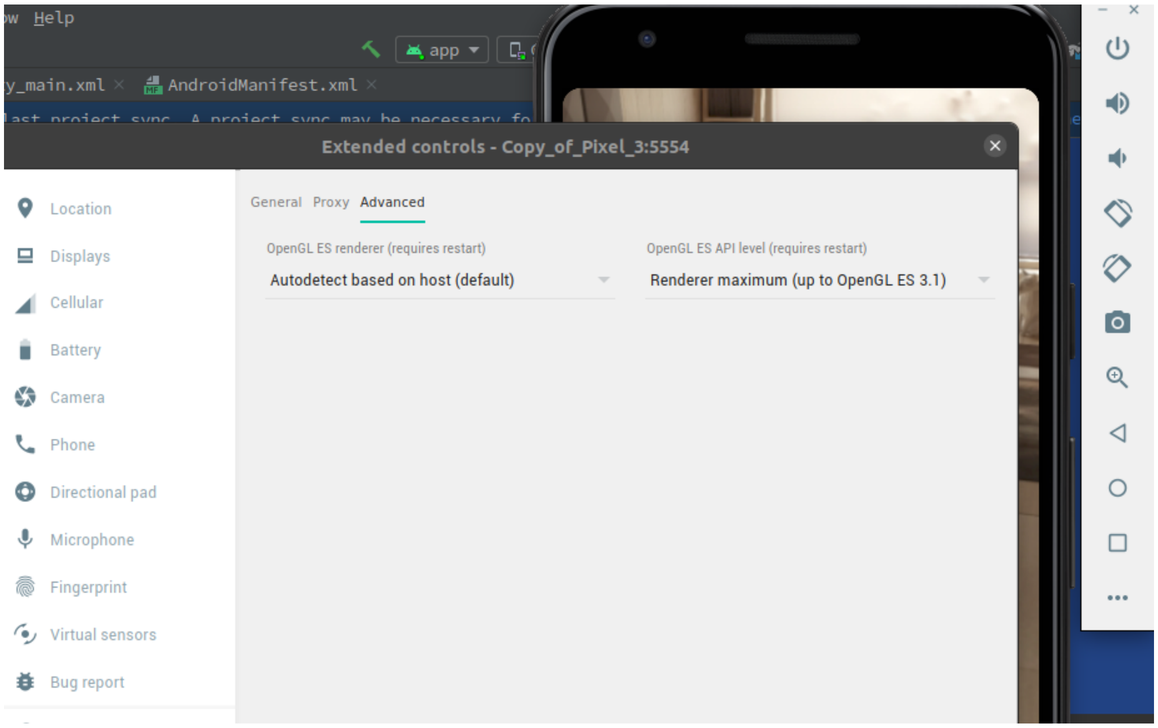
Pixel 3a 5.6 1080x2220 xxhdpi Change...

Q Android 10.0 x86_64 Change...

Startup orientation: Portrait, Landscape

Camera Front: Webcam0

Back: VirtualScene



Extended controls - Copy_of_Pixel_3:5554

OpenGL ES renderer (requires restart): Autodetect based on host (default)

OpenGL ES API level (requires restart): Renderer maximum (up to OpenGL ES 3.1)

3. Download Google Play Services for ARCore APK. Drag this into your emulated device.
   https://github.com/google-ar/arcore-android-sdk/releases/download/v1.23.0/Google_Play_Services_for_AR_1.23.0_x86_for_emulator.apk

4. Add dependencies in build.gradle(s) and the app's manifest:
    a. In build.gradle (Module: AR.app):

```
android {
defaultConfig {
      …
      minSdkVersion 24
}
```

```
dependencies {
    …
    implementation 'com.google.ar:core:1.15.0'
    implementation 'com.google.ar.sceneform.ux:sceneform-ux:1.15.0'
    implementation 'com.google.ar.sceneform:core:1.15.0'
    implementation 'com.google.ar.sceneform:assets:1.15.0'
}
```

    b. In AndroidManifest.xml:

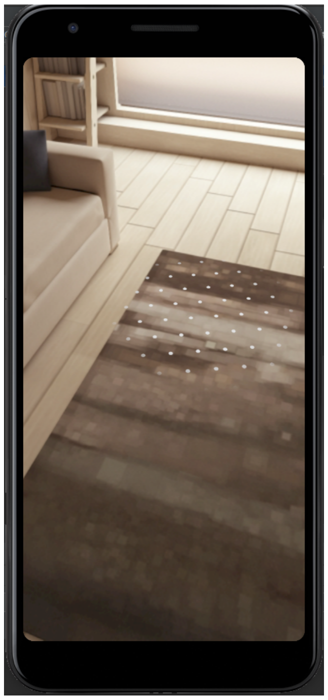
```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera.ar" />
<uses-feature android:glEsVersion="0x00030000" android:required="true" />
<application …>
    …
    <meta-data android:name="com.google.ar.core" android:value="required"/>
</application>
```

5. Build your project so that your dependencies will register. Else the proper imports won't be found in the next steps.
6. Write your activity_main.xml.

   Within a ConstraintLayout, include an AR fragment.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
   <fragment
       android:id="@+id/arView"
       android:name="com.google.ar.sceneform.ux.ArFragment"
       android:layout_width="match_parent"
       android:layout_height="match_parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

7. Check the progress from Part 1. Your app should open and request camera permissions. Since we set the back camera to a VirtualScene, you should be able to view and move around in the virtual world. (Hold Alt and move your mouse; you can also use WASDQE.) You'll notice that SceneForm is already able to detect some surfaces. It will illustrate surfaces with white dots.



## Part 2 - Getting a basic AR Render

1. Create a SceneView and a Renderable Object.
   a. Add this SceneView to activity_main.xml after the arFragment:

```
<com.google.ar.sceneform.SceneView
   android:id="@+id/scene_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

   b. Add the following class fields to MainActivity to keep track of some important things:

```
private static final String TAG = "DEBUG_LOG";
private ModelRenderable modelRenderable;
private ArFragment arFragment;
private boolean modelPlaced = false;
```

   c. Populate the arFragment field.

```
arFragment = (ArFragment) getSupportFragmentManager()
                         .findFragmentById(R.id.arView);
```

   d. Before adding fun models, we'll start with a generated object (a red sphere). Add this to the onCreate method in MainActivity:

```
MaterialFactory.makeOpaqueWithColor(this,                                    new
Color(android.graphics.Color.RED))
   .thenAccept(material -> { modelRenderable =
                 ShapeFactory.makeSphere(0.1f, new Vector3(0.0f, 0.15f, 0.0f),
material);
               Log.d(TAG, "onCreate: success rendering sphere!");
    });
```

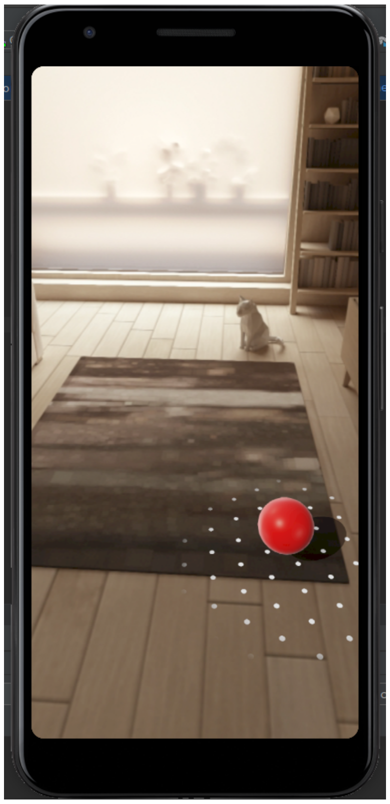
   e. Now for the most important part, where we render our modelRenderable in our arFragment's SceneView. At this point, we'll just render a single sphere wherever the sceneview can first detect a plane (and leave it there). Add the following function to MainActivity and invoke it in onCreate.

```
private void setupSceneView(){
   arFragment.getArSceneView().getScene().addOnUpdateListener(frameTime -> {
       Frame frame = arFragment.getArSceneView().getArFrame();
       if (frame != null && !modelPlaced) {
           for (Plane plane : frame.getUpdatedTrackables(Plane.class)) {
               if (plane.getTrackingState() == TrackingState.TRACKING) {
                   // Log.d(TAG, "onUpdate: got dots!");
                   arFragment.getPlaneDiscoveryController().hide();
                   Iterator<Anchor> iterableAnchor =
frame.getUpdatedAnchors().iterator();
                   if (!iterableAnchor.hasNext()) {
                       List<HitResult> hitTest = frame.hitTest((float)
                   arFragment.getArSceneView().getWidth() / 2, (float)
                   arFragment.getArSceneView().getHeight() / 2);
```

```
                        // iterate through all hits
                        Iterator<HitResult> hitTestIterator =
hitTest.iterator();
                        while (hitTestIterator.hasNext() && !modelPlaced) {
                            HitResult hitResult = hitTestIterator.next();
                            Anchor modelAnchor =
plane.createAnchor(hitResult.getHitPose());
                            AnchorNode anchorNode = new AnchorNode(modelAnchor);

anchorNode.setParent(arFragment.getArSceneView().getScene());
                            TransformableNode transformableNode = new
TransformableNode(arFragment.getTransformationSystem());
                            transformableNode.setParent(anchorNode);
                            transformableNode.setRenderable(modelRenderable);
                            transformableNode.setWorldPosition(new Vector3(
                                    modelAnchor.getPose().tx(),
                                    modelAnchor.getPose().compose(
                                     Pose.makeTranslation(0f, 0.05f, 0f)).ty(),
                                    modelAnchor.getPose().tz()
                            ));
                            transformableNode.setLocalRotation(
                               Quaternion.axisAngle(
                                 new Vector3(0f, 1f, 0f), 180));
                            modelPlaced = true;
                            // Log.d(TAG, "onUpdate: placed!");
                        }
                    }
                }
            }}});
}
```

2. Check the progress from Part 2. Your app should place a small red sphere on a surface.



## Part 3 - Custom Object Models

1. Add the files to the project.

   For the sake of simplicity, I am providing a folder of asset files for you here. These models were made by the 3d artist Tipatat Chennavasin. Add an **assets** folder in app/src/main and put the contents of this zip here.

   In case you're curious on how to obtain your own .sfb files, here's a quick list of steps.

   a. Go to Google Poly or similar 3d model library site (Google Poly will be shut down this June 😞) and find a downloadable OBJ file. These should be zips of two files actually: one .obj and one .mtl. This pair can be converted to the .sfb (sceneform binary) format, which is usable by Android ARCore.

   b. After unzipping that download, you will need the following tool to convert: https://github.com/gdamoreira/google-ar-asset-converter.git

Clone this repo into a convenient place and use this command (be sure the .obj file and the .mtl file are in the same place, since you will only be pointing to one and the location of the other will be assumed):

```
./google-ar-asset-converter/sceneform_sdk/linux/converter -d --mat
./google-ar-asset-converter/sceneform_sdk/default_materials/obj_mater
ial.sfm --output [name of output file] [path to input .obj file]
```

   c. Now your .sfb file is ready to put in /app/src/main/assets.

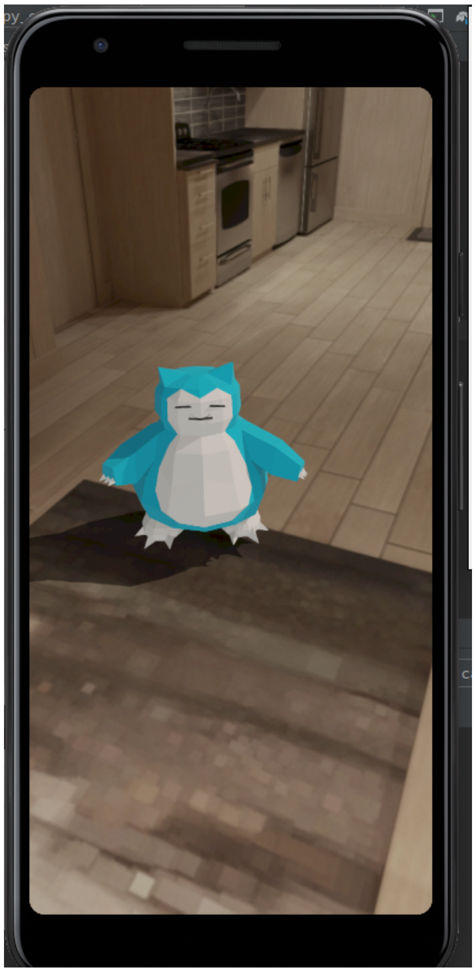
2. Change MainActivity to use this custom model instead of the red sphere.
   a. Add this private field to MainActivity with the file of your choice.

```
private String modelFile = "snorlax.sfb";
```

   b. Comment out the portion that makes the red sphere
   c. In its stead, add this function to MainActivity to pull from the model file. Call this function in onCreate.

```
private void renderObject(){
   ModelRenderable.builder()
           .setSource(this, Uri.parse(modelFile))
           .build()
           .thenAccept(renderable -> {
              renderable.setShadowCaster(false);
               modelRenderable = renderable;
           });
}
```

3. Check the progress from Part 3. Your app should place the pokemon of your choice on a surface.



## Part 4 - Switching Models

1. Let's add something familiar: a toolbar at the top with a dropdown spinner.
   a. In activity_main.xml, wrap the whole thing in a Coordinator Layout and add an AppBarLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <com.google.android.material.appbar.AppBarLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:background="@color/black"
       android:textColor="@color/white"
       android:fitsSystemWindows="true">
       <androidx.appcompat.widget.Toolbar
           android:id="@+id/toolbar"
           android:layout_width="match_parent"
           android:layout_height="wrap_content">
           <Spinner
               android:id="@+id/spinner"
               android:padding="10dp"
               android:layout_height="wrap_content"
               android:layout_width="wrap_content"/>
       </androidx.appcompat.widget.Toolbar>
   </com.google.android.material.appbar.AppBarLayout>
```

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
       xmlns:app="http://schemas.android.com/apk/res-auto"
       xmlns:tools="http://schemas.android.com/tools"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       tools:context=".MainActivity">
       <fragment
           android:id="@+id/arView"
           android:name="com.google.ar.sceneform.ux.ArFragment"
           android:layout_width="match_parent"
           android:layout_height="match_parent"/>
       <com.google.ar.sceneform.ArSceneView
           android:id="@+id/scene_view"
           android:layout_width="match_parent"
           android:layout_height="match_parent"/>
   </androidx.constraintlayout.widget.ConstraintLayout>
```

```
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

b. In MainActivity.java, add the following private fields:

```
private Toolbar toolbar;
private Spinner spinner;
```

c. In MainActivity's onCreate function add the following:

```
toolbar = findViewById(R.id.toolbar);
spinner = findViewById(R.id.spinner);
setSupportActionBar(toolbar);
Objects.requireNonNull(getSupportActionBar()).setDisplayShowTitleEnabled(false);
```

d. In AndroidManifest.xml add the following to the definition of <application/>:

```
<application ...
   android:theme="@style/Theme.Design.NoActionBar">
```

2. Populate the spinner in the toolbar
   a. Add a new file **spinner_item.xml** to **app/src/main/res/layout** and paste this in:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="@color/black"
   android:gravity="start"
   android:padding="5dip"
   android:textColor="@color/white"
   android:textSize="20sp" />
```

   b. Add a new file **Pokemon.java** to app/src/main/java/[package] and paste this in:

```
package com.example.ar1;
import androidx.annotation.NonNull;
public class Pokemon {
   public String name;
   public String file;
   public Pokemon(String n, String f){
       name = n;
       file = f;
   }
   @NonNull
   @Override
   public String toString() {
       return name;
   }
}
```

   c. Add the following private field in MainActivity:

```
private List<Pokemon> pokemon_array = new ArrayList<Pokemon>(List.of(
 new Pokemon("Abra", "abra.sfb"), new Pokemon("Bulbasaur", "bulbasaur.sfb"),
 new Pokemon("Charmander", "charmander.sfb"), new Pokemon("Cubone", "cubone.sfb"),
 new Pokemon("Eevee", "eevee.sfb"), new Pokemon("Haunter", "haunter.sfb"),
 new Pokemon("Jigglypuff", "jigglypuff.sfb"), new Pokemon("Magikarp", "magikarp.sfb"),
 new Pokemon("Mew", "mew.sfb"), new Pokemon("Oddish", "oddish.sfb"),
 new Pokemon("Snorlax", "snorlax.sfb"), new Pokemon("Squirtle", "squirtle.sfb")
));
```

   d. Add the following functions to MainActivity:

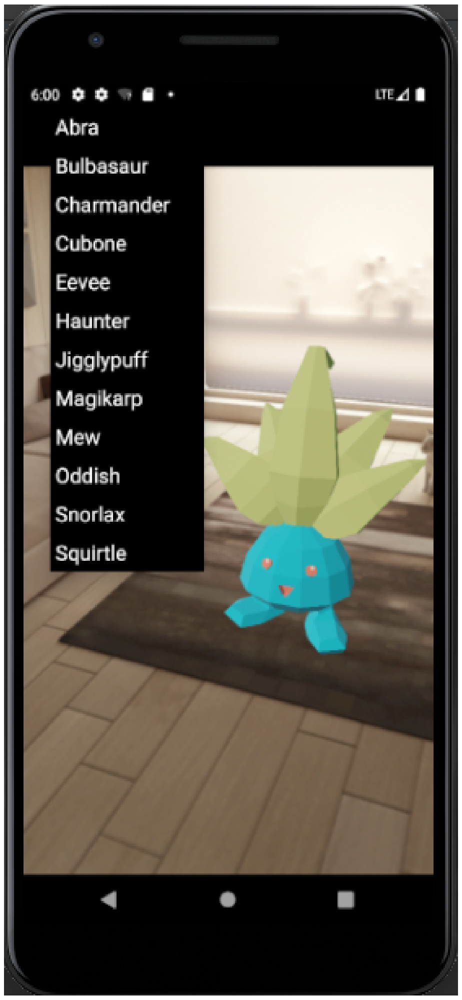
```
private void setupSpinner() {
   spinner.setAdapter(
        new ArrayAdapter<Pokemon>(this, R.layout.spinner_item, pokemon_array));
   spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
     @Override
     public void onItemSelected(AdapterView<?> arg0, View arg1, int pos, long rowid) {
           modelFile = pokemon_array.get((int) rowid).file;
           modelPlaced = false;
           clearScene();
           renderObject();
       }
```

```
        @Override
        public void onNothingSelected(AdapterView<?> arg0) {
        }
    });
}

private void clearScene(){
    List<Node> children = new ArrayList<>(
                                     arFragment.getArSceneView().getScene().getChildren());
    for (Node node : children) {
        if(!node.toString().contains("Sun") && !node.toString().contains("Camera")){
            node.setParent(null);
        }
    }
}
```

e. Invoke the setupSpinner function anywhere in onCreate.

3. Check the progress from Part 4. Your app should allow you to switch between twelve different pokemon models to place in the world. Now you have a basic AR demo app!


6:00
LTE
Abra
Bulbasaur
Charmander
Cubone
Eevee
Haunter
Jigglypuff
Magikarp
Mew
Oddish
Snorlax
Squirtle