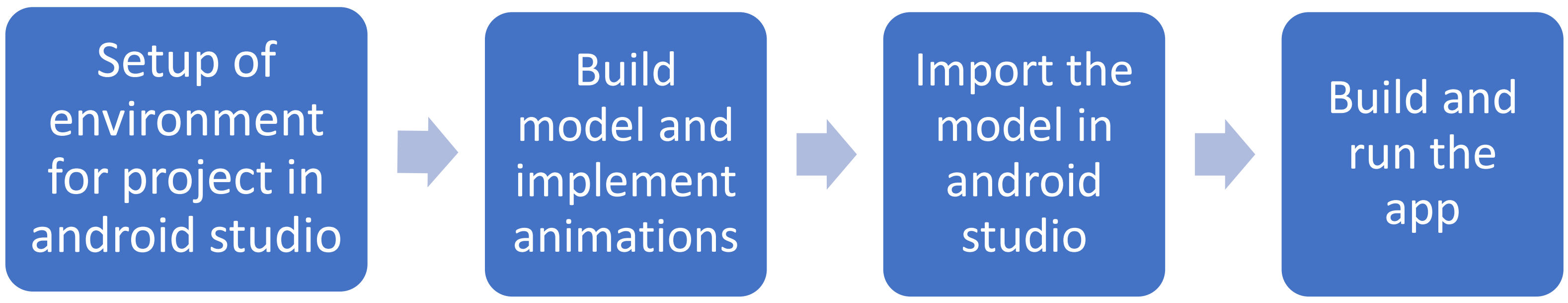
Workflow for development of the project:


Setup of environment for project in android studio
Build model and implement animations
Import the model in android studio
Build and run the app

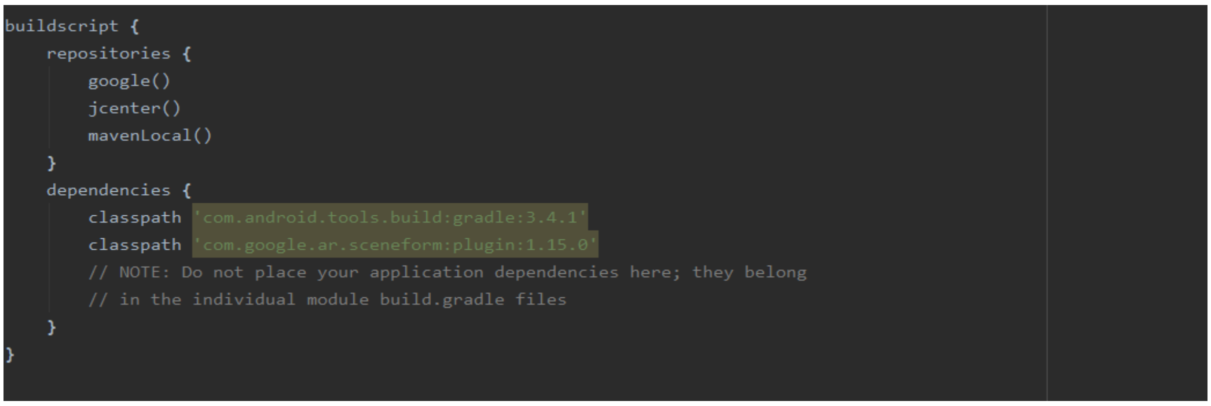
For the project, we have android studio as the software which manipulates the data for AR app to be executed. ARCore is software development kit by Google that assists in the building of AR applications in studio**.** Blender has been used for character modelling and providing required animation to the skeleton model.

Steps used for the development of the project are:

1. First, we have to set the environment for the development of the project. Sceneform is the plugin used for viewing, importing and building 3D assets. To install the plugin:

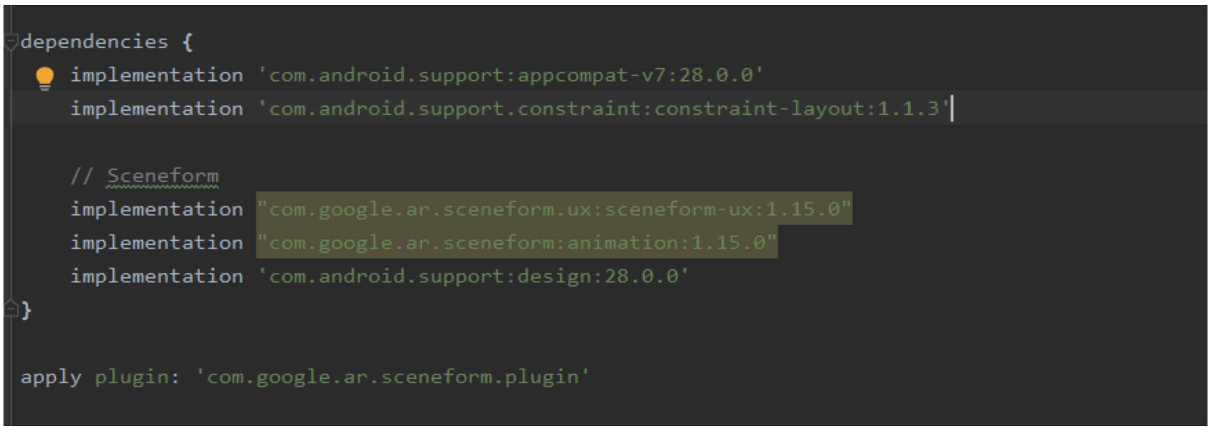
File > Settings > Plugins > Browse Repositories

After that, we have to make changes in the gradle(build) by adding classpath of Sceneform and gradle(app) by adding implementations of Sceneform and animation and Sceneform plugin too.



```
buildscript {
    repositories {
        google()
        jcenter()
        mavenLocal()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'com.google.ar.sceneform:plugin:1.15.0'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

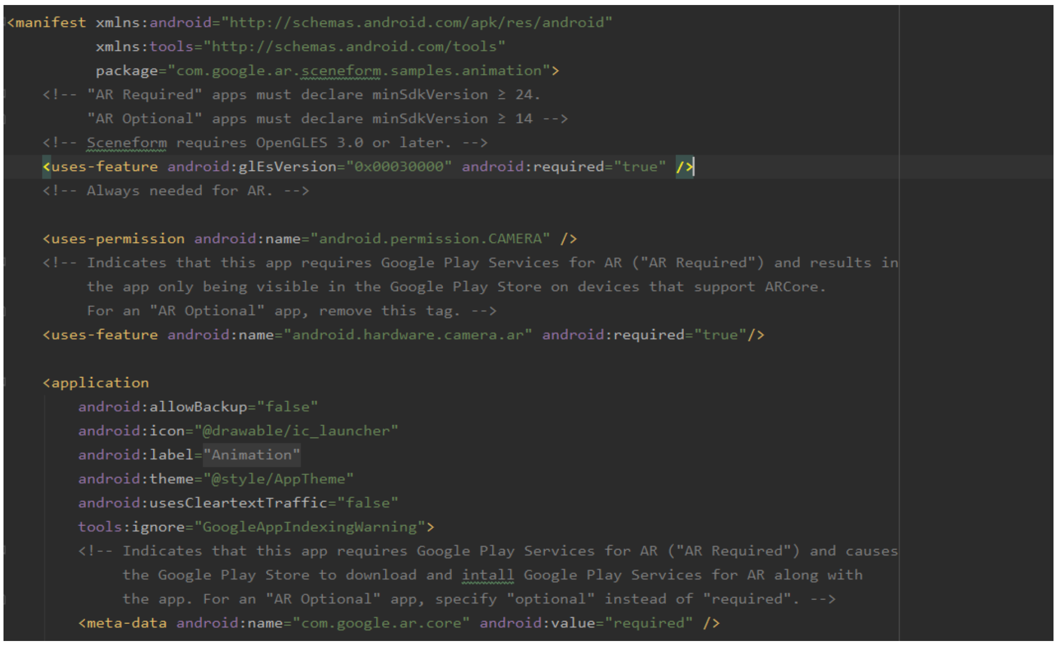
Build Gradle



```
dependencies {
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'

    // Sceneform
    implementation "com.google.ar.sceneform.ux:sceneform-ux:1.15.0"
    implementation "com.google.ar.sceneform:animation:1.15.0"
    implementation 'com.android.support:design:28.0.0'
}

apply plugin: 'com.google.ar.sceneform.plugin'
```

Build Gradle(app)

Also, we have to make changes in **AndroidManifest.xml** as to give App access to the camera and to indicate whether your app is AR Optional or AR Required.



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
          package="com.google.ar.sceneform.samples.animation">
    <!-- "AR Required" apps must declare minSdkVersion ≥ 24.
         "AR Optional" apps must declare minSdkVersion ≥ 14 -->
    <!-- Sceneform requires OpenGLES 3.0 or later. -->
    <uses-feature android:glEsVersion="0x00030000" android:required="true" />
    <!-- Always needed for AR. -->

    <uses-permission android:name="android.permission.CAMERA" />
    <!-- Indicates that this app requires Google Play Services for AR ("AR Required") and results in
         the app only being visible in the Google Play Store on devices that support ARCore.
         For an "AR Optional" app, remove this tag. -->
    <uses-feature android:name="android.hardware.camera.ar" android:required="true"/>

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="Animation"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="false"
        tools:ignore="GoogleAppIndexingWarning">
        <!-- Indicates that this app requires Google Play Services for AR ("AR Required") and causes
             the Google Play Store to download and intall Google Play Services for AR along with
             the app. For an "AR Optional" app, specify "optional" instead of "required". -->
        <meta-data android:name="com.google.ar.core" android:value="required" />
```

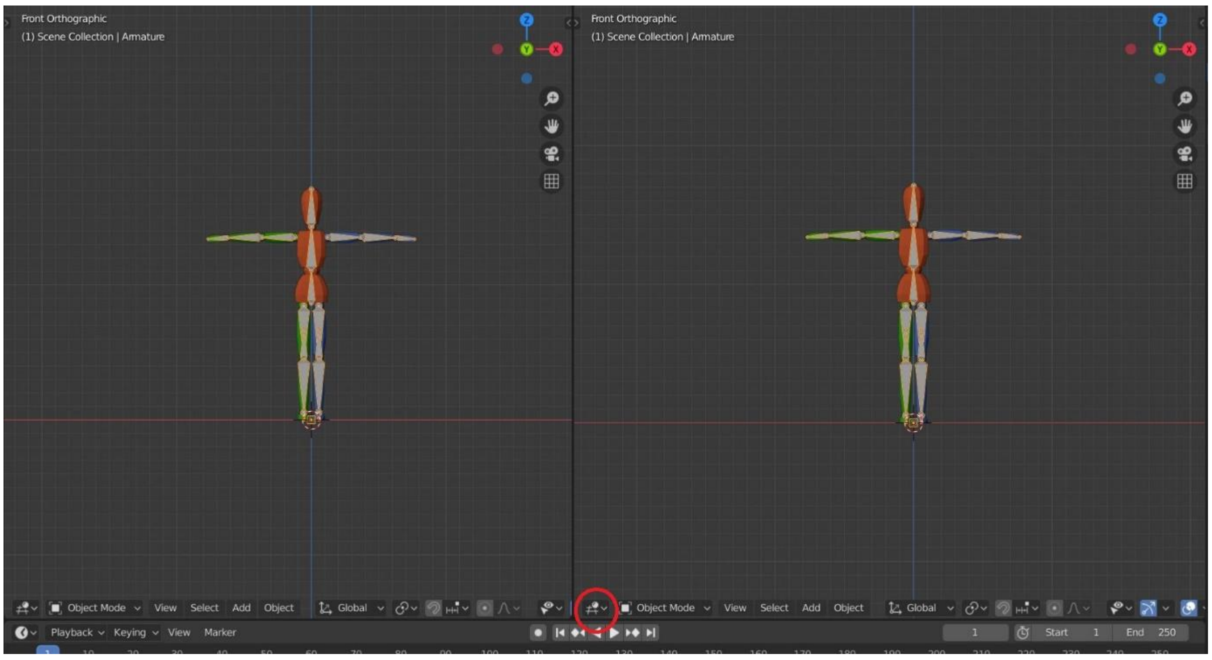
2. Now, we work on to build our skeleton model. For that, we will be using the Blender interface. We can either import the model or build it from scratch. To build the skeleton model, use the concepts of character modelling. The concept description and procedure to export file from blender to the studio are given in this pdf:

   https://documentcloud.adobe.com/link/review?uri=urn:aaid:scds:US:37b459d2-c827-4223-a8b6-de2d3b675332

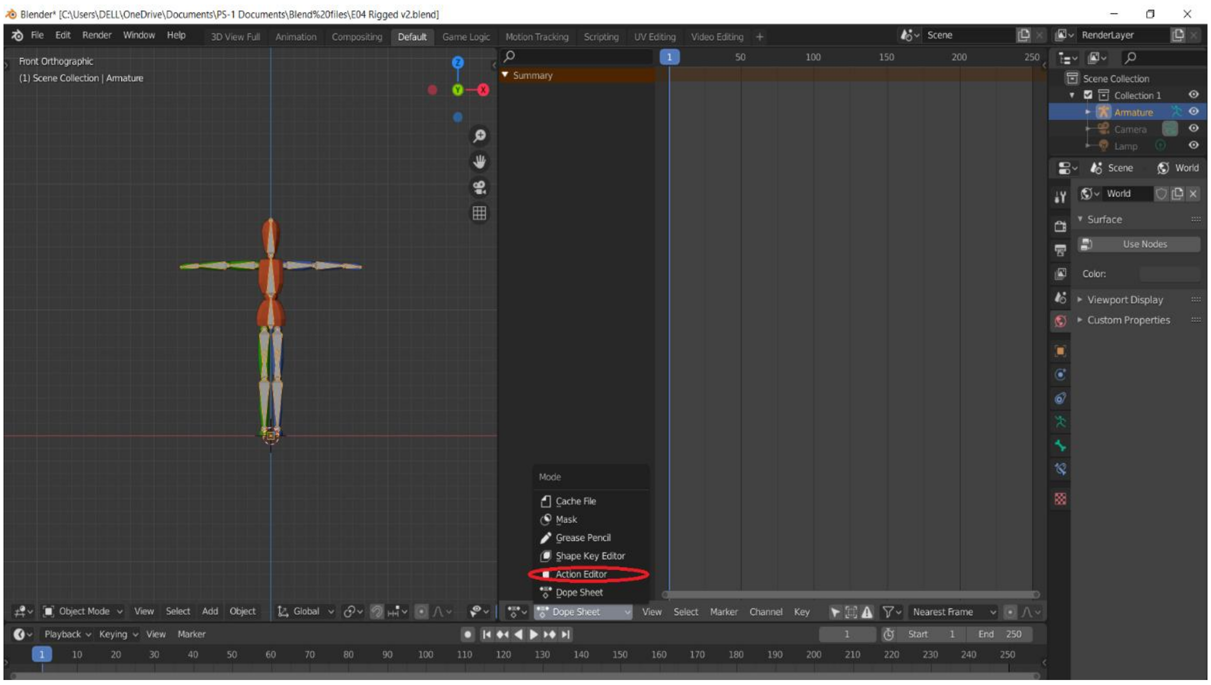
We have used ideas presented in the following videos to model our skeleton:

- https://youtu.be/ocLZNRasU64
- https://youtu.be/WnPkrxz4AQQ
- https://youtu.be/8mZtc33rQ3c
- https://youtu.be/9iaB2GpzIxQ

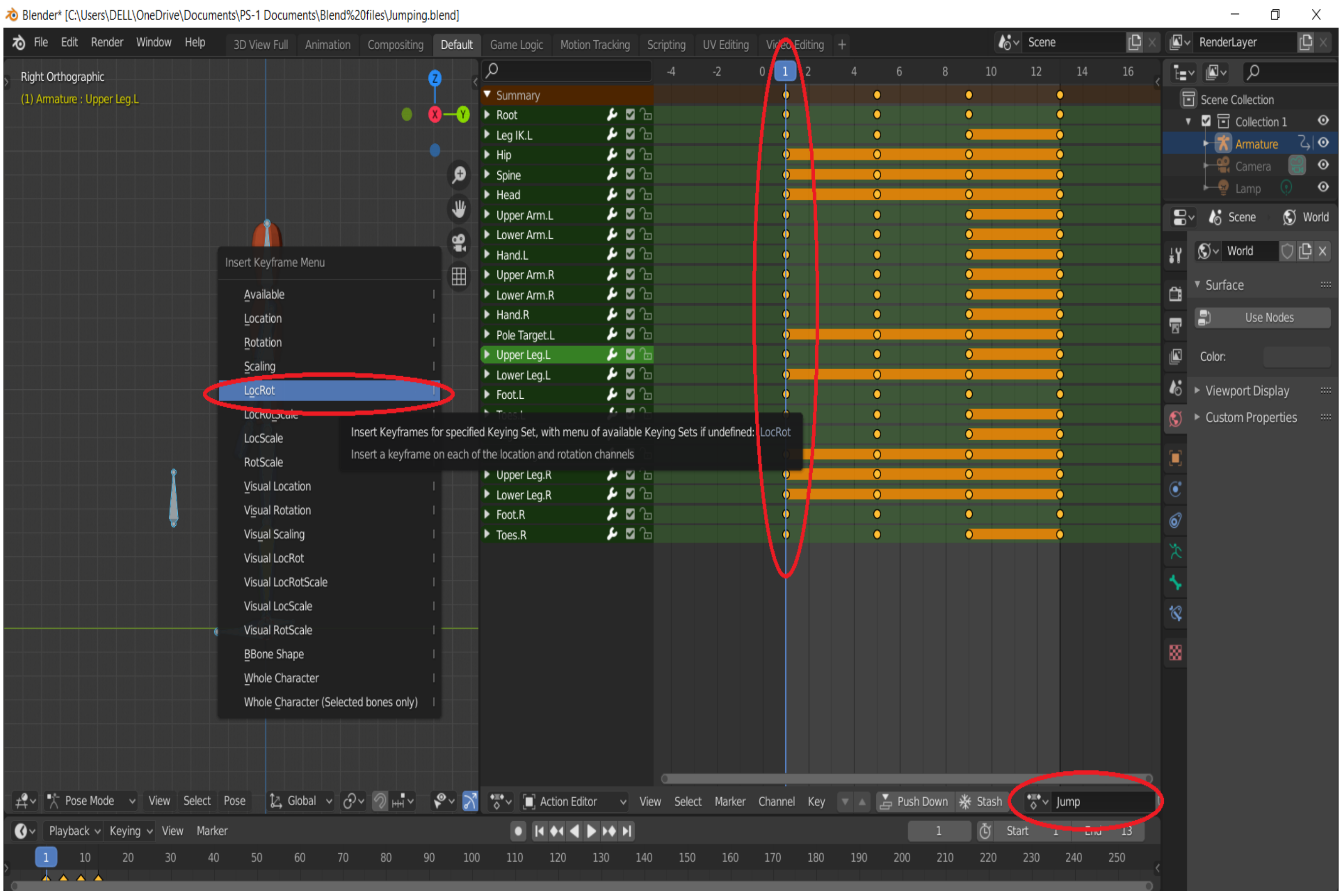
3. Next, we will move on to give the necessary animations to our character model. To get started with animation, we will first split the screen into two. To do that, move the mouse to top left corner of the scene and drag it. The screen will get distributed and shown as:



Now, go to the menu (circled as red) and choose <u>dope sheet</u> given under animation. After that, select action editor as shown in the figure:



This will bring us to the interface where we build actions in the model. To add an action, click on **+New** button present alongside which we used to create a dope sheet. We can rename the action as convenience, and the action editor displays frames where we can add keyframes of our skeleton. Now we will start editing the frames by moving/rotating bones. First, we have to change mode of the skeleton to <u>Pose Mode</u>. To move a bone, press 'G' and press 'R' to rotate the bone after selecting the desired bone. After making required changes to the model, we can add keyframes to the frame by first pressing 'A'(it will select all the bones) and then press 'I' which will show a drop-down menu. Press **LocRot** to set location and rotation of bones to that frame.

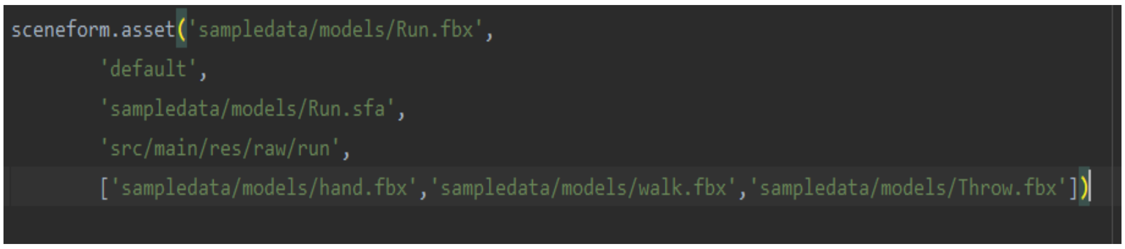


Insert Keyframes for specified Keying Set, with menu of available Keying Sets if undefined: LocRot
Insert a keyframe on each of the location and rotation channels

Refer to the following video for the detailed procedure: https://youtu.be/sTo4adwvulE

After adding keyframes to the frame window of action editor, we can play the animation by using **Play** button present in bottom middle and also can edit start and end frames from the box present below bottom red circle in the previous figure. Procedure to export animation is given in pdf (refer to Step 2). Android studio supports .fbx file for animation, so we have to make sure to export the desired file correctly.

4. We will now import the model (with animations) in the studio. For that, first, we will paste the file in sampledata/models folder present in the app folder. After that, we will edit the build Gradle(app) to import the assets.

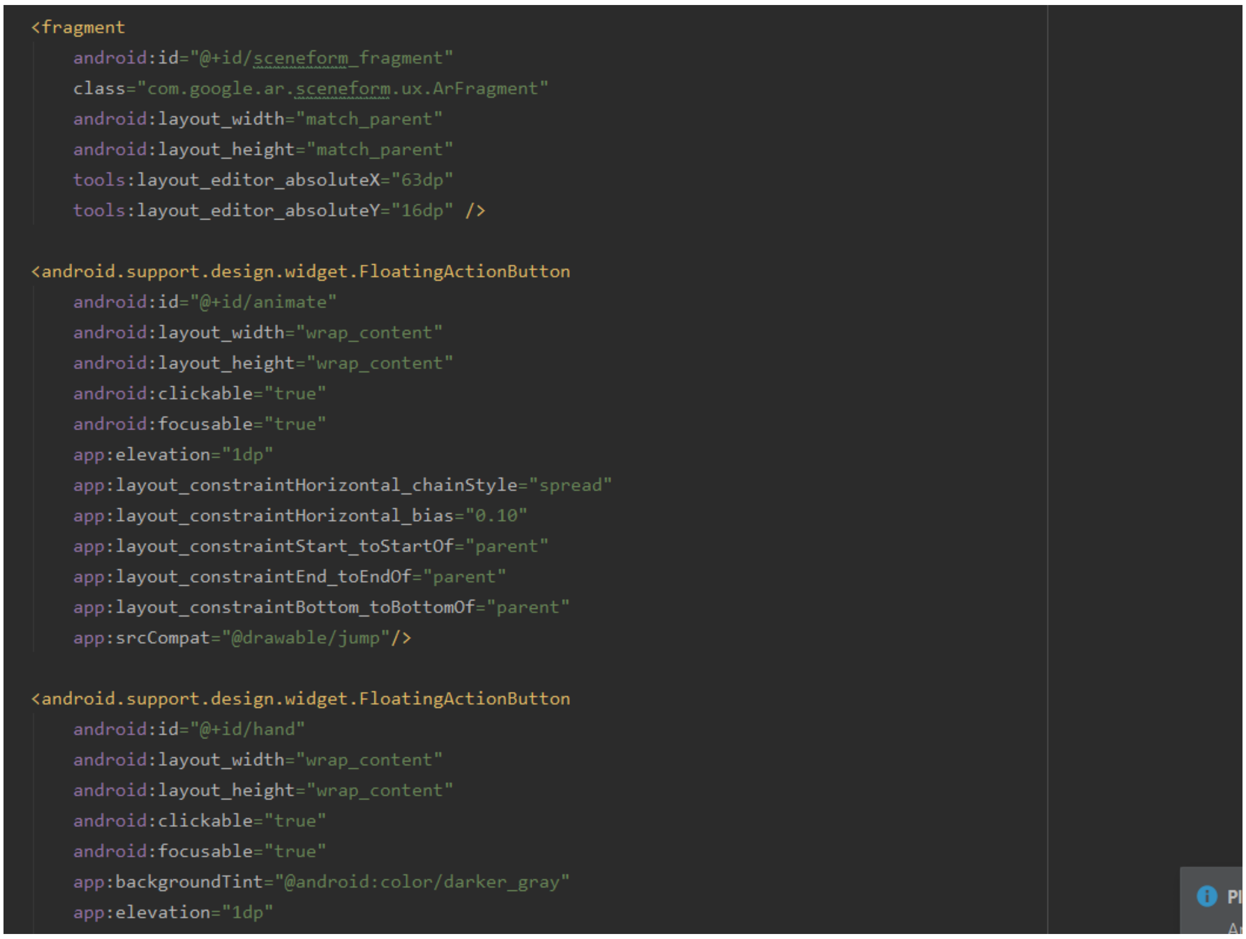


```
sceneform.asset('sampledata/models/Run.fbx',
        'default',
        'sampledata/models/Run.sfa',
        'src/main/res/raw/run',
        ['sampledata/models/hand.fbx','sampledata/models/walk.fbx','sampledata/models/Throw.fbx'])
```

After pressing build, two files (.sfa and .sfb) will be generated. We will now work on the layout by adding a fragment. ArFragment creates:

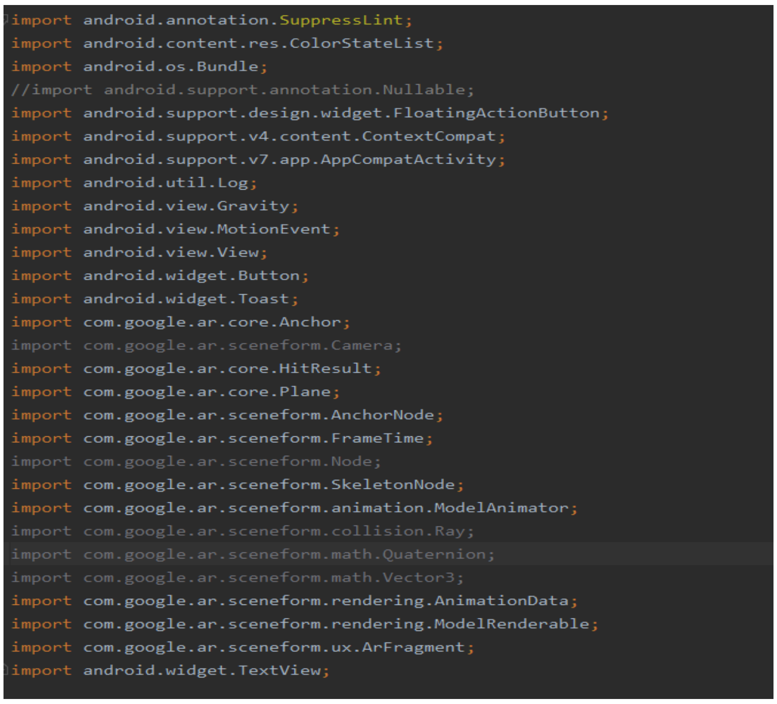
- An ArSceneView, accessible via getArSceneView(), that:
  - Renders the camera images from the session on to its surface
  - Renders a built-in Sceneform UX animation that shows users how they should move their phone to activate the AR experience.
  - Highlights detected Planes using the default PlaneRenderer
- An ARCore Session, accessible via getSession()

We have also added Floating Buttons to ignite animations in the project.



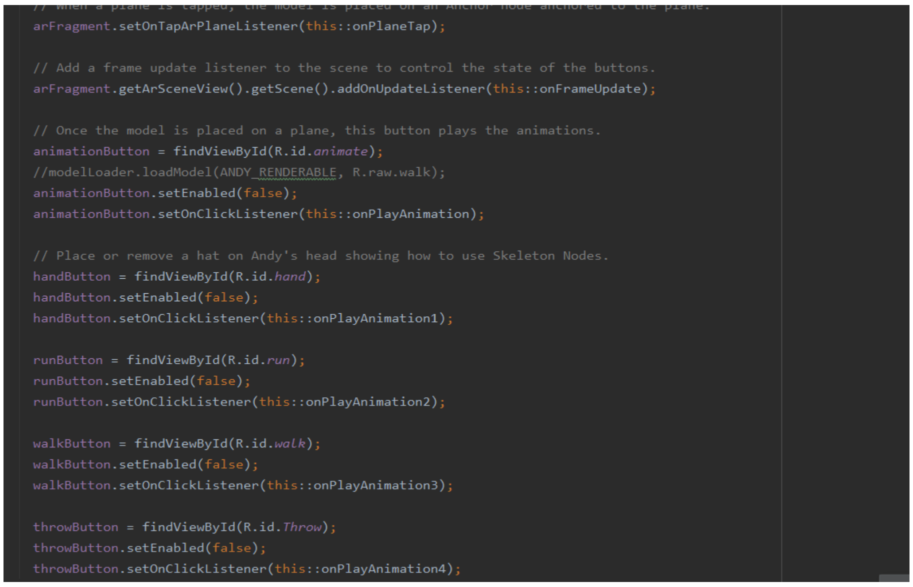
```
<fragment
    android:id="@+id/sceneform_fragment"
    class="com.google.ar.sceneform.ux.ArFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout_editor_absoluteX="63dp"
    tools:layout_editor_absoluteY="16dp" />

<android.support.design.widget.FloatingActionButton
    android:id="@+id/animate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:clickable="true"
    android:focusable="true"
    app:elevation="1dp"
    app:layout_constraintHorizontal_chainStyle="spread"
    app:layout_constraintHorizontal_bias="0.10"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:srcCompat="@drawable/jump"/>

<android.support.design.widget.FloatingActionButton
    android:id="@+id/hand"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:clickable="true"
    android:focusable="true"
    app:backgroundTint="@android:color/darker_gray"
    app:elevation="1dp"
```

5. We have used Java to create and edit classes in the project. For the project, we have used two classes, namely MainActivity.java and ModelLoader.java. ModelLoader is used to load the model from the file and set renderable to it. A Renderable is a 3D model and consists of vertices, materials, textures, and more. We have to first import required packages in MainActivity which are given as:



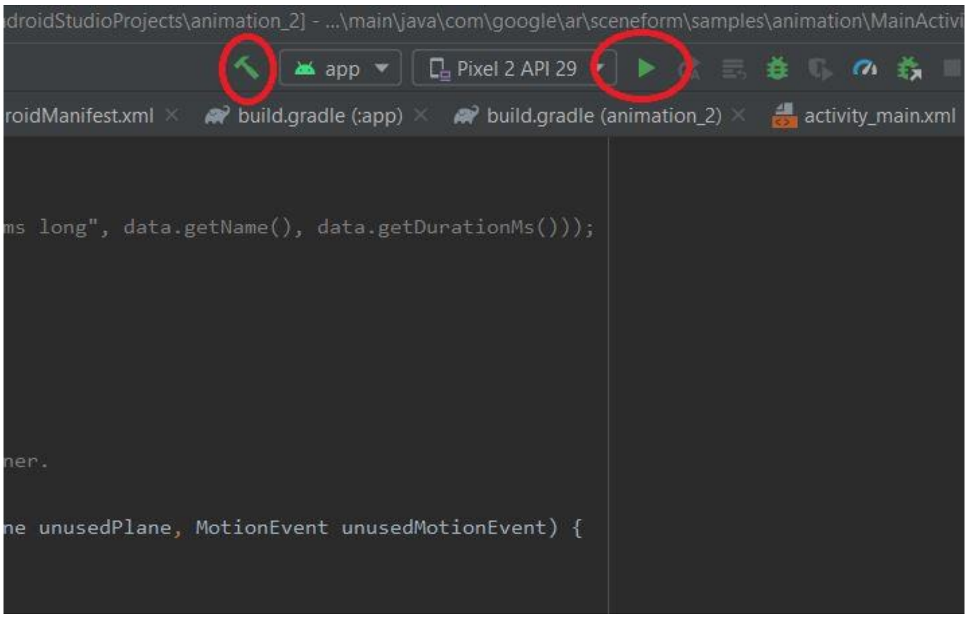
```java
import android.annotation.SuppressLint;
import android.content.res.ColorStateList;
import android.os.Bundle;
//import android.support.annotation.Nullable;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Gravity;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import com.google.ar.core.Anchor;
import com.google.ar.sceneform.Camera;
import com.google.ar.core.HitResult;
import com.google.ar.core.Plane;
import com.google.ar.sceneform.AnchorNode;
import com.google.ar.sceneform.FrameTime;
import com.google.ar.sceneform.Node;
import com.google.ar.sceneform.SkeletonNode;
import com.google.ar.sceneform.animation.ModelAnimator;
import com.google.ar.sceneform.collision.Ray;
import com.google.ar.sceneform.math.Quaternion;
import com.google.ar.sceneform.math.Vector3;
import com.google.ar.sceneform.rendering.AnimationData;
import com.google.ar.sceneform.rendering.ModelRenderable;
import com.google.ar.sceneform.ux.ArFragment;
import android.widget.TextView;
```

We will then give the required references and use loadModel function in ModelLoader.java to load the skeleton.



```java
// When a plane is tapped, the model is placed on an Anchor node anchored to the plane.
arFragment.setOnTapArPlaneListener(this::onPlaneTap);

// Add a frame update listener to the scene to control the state of the buttons.
arFragment.getArSceneView().getScene().addOnUpdateListener(this::onFrameUpdate);

// Once the model is placed on a plane, this button plays the animations.
animationButton = findViewById(R.id.animate);
//modelLoader.loadModel(ANDY_RENDERABLE, R.raw.walk);
animationButton.setEnabled(false);
animationButton.setOnClickListener(this::onPlayAnimation);

// Place or remove a hat on Andy's head showing how to use Skeleton Nodes.
handButton = findViewById(R.id.hand);
handButton.setEnabled(false);
handButton.setOnClickListener(this::onPlayAnimation1);

runButton = findViewById(R.id.run);
runButton.setEnabled(false);
runButton.setOnClickListener(this::onPlayAnimation2);

walkButton = findViewById(R.id.walk);
walkButton.setEnabled(false);
walkButton.setOnClickListener(this::onPlayAnimation3);

throwButton = findViewById(R.id.Throw);
throwButton.setEnabled(false);
throwButton.setOnClickListener(this::onPlayAnimation4);
```

To get animation data and run animation, we have built onPlayAnimation function. It gets access to the animation data from the file and then projects it. When we tap the plane from the phone screen, onPlanTap function will be started which built anchorNode(They ensure that objects appear to stay at the same position and orientation in space, helping you maintain the illusion of virtual objects placed in the real world.) and skeletonNode whose reference is to the model renderable. onFrameUpdate function is used to activate/deactivate buttons required occasions. This all sums up a brief summary of how classes are going to work when we run the app. To run the app, first click on 'build' and then 'run' icon present on the top menu as shown:


droidStudioProjects\animation_2] - ...\main\java\com\google\ar\sceneform\samples\animation\MainActivi
app
Pixel 2 API 29
roidManifest.xml
build.gradle (:app)
build.gradle (animation_2)
activity_main.xml

```
ms long", data.getName(), data.getDurationMs()));

ner.

ne unusedPlane, MotionEvent unusedMotionEvent) {
```