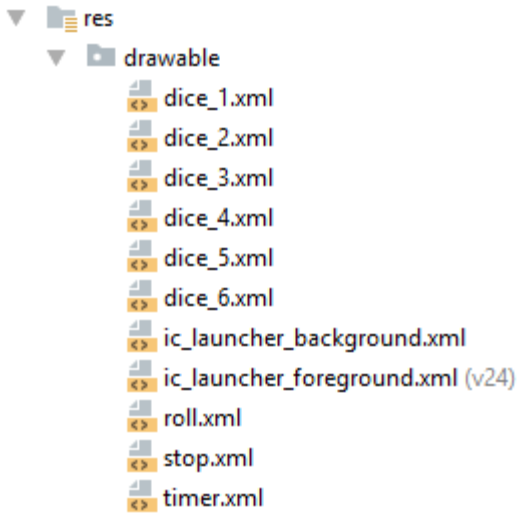
Dice Roller Application Documentation

This material shows the reader how to create a *Dice Roller* app that rolls one to three dice when the user taps the Roll button in the app bar.

Download the drawables.zip file, which contains all the drawables used by the Dice Roller app. Unzip the file contents, and place the `XML` files in the Dice Roller project's `res/drawable` directory



Heres How your MVC Architecture should look like:

Model	View	Controller
Dice.java	activity_main.xml	MainActivity.java

The Model

The Model is a single Dice class, which can create an individual Dice object. The Dice constructor calls a `setNumber()` method, which allows the Dice object to be set to a number between 1 and 6 and associates the related vector drawable with the dice. You will need to code this yourself.

The View

The figures below define the app's colors and strings.

Replace the existing `XML` in the appropriate files with the `XML` in the figures below.

res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="colorPrimary">#3F51B5</color>
   <color name="colorPrimaryDark">#303F9F</color>
   <color name="colorAccent">#FF4081</color>
   <color name="colorDice">#D50000</color>
</resources>
```

res/values/strings.xml.

```
<resources>
   <string name="app_name">Dice Roller</string>
   <string name="action_roll">Roll</string>
   <string name="action_one">One</string>
   <string name="action_two">Two</string>
   <string name="action_three">Three</string>
   <string name="action_stop">Stop</string>
   <string name="dice_one">1</string>
</resources>
```

The main activity's layout file uses `ImageViews` to display three dice. The height of each `ImageView` is 0dp, so the weight is used to calculate the height. All three `ImageViews` have a weight of 1, so all three `ImageViews` are assigned equal heights that fill the `LinearLayout` parent.

Replace the existing `XML` in `activity_main`.xml with the `XML` in the figure below.

res/layout/activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   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:orientation="vertical"
   android:gravity="center"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="com.zybooks.diceroller.MainActivity">

   <ImageView
       android:id="@+id/dice1"
       android:layout_width="match_parent"
       android:layout_height="0dp"
       android:layout_weight="1"
       android:contentDescription="@string/dice_one"
       android:tint="@color/colorDice"
       android:src="@drawable/dice_1" />

   <ImageView
       android:id="@+id/dice2"
       android:layout_width="match_parent"
       android:layout_height="0dp"
       android:layout_weight="1"
       android:contentDescription="@string/dice_one"
       android:tint="@color/colorDice"
       android:src="@drawable/dice_1" />

   <ImageView
       android:id="@+id/dice3"
       android:layout_width="match_parent"
       android:layout_height="0dp"
       android:layout_weight="1"
       android:contentDescription="@string/dice_one"
       android:tint="@color/colorDice"
       android:src="@drawable/dice_1" />

</LinearLayout>
```

The `appbar_menu.xml` file defines the app bar action items and overflow menu. Create the menu resource:

1. In the Android project view, right-click the `res` directory and choose `New → Directory` from the context menu.
2. Name the directory "`menu`" and press OK.
3. Right-click on the `res/menu` directory and choose `New → File` from the context menu.
4. Name the file **`appbar_menu.xml`** and press OK.

Replace the existing `XML` in `appbar_menu.xml` with the `XML` in the figure below.

res/menu/appbar_menu.xml.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">

   <item android:id="@+id/action_stop"
       android:title="@string/action_stop"
       android:icon="@drawable/stop"
       android:visible="false"
       app:showAsAction="ifRoom"/>

   <item android:id="@+id/action_roll"
       android:title="@string/action_roll"
       android:icon="@drawable/roll"
       app:showAsAction="ifRoom"/>

   <item android:id="@+id/action_one"
       android:title="@string/action_one"
       app:showAsAction="never"/>

   <item android:id="@+id/action_two"
       android:title="@string/action_two"
       app:showAsAction="never"/>

   <item android:id="@+id/action_three"
       android:title="@string/action_three"
       app:showAsAction="never"/>

</menu>
```

The Controller:

You will need to code the Controller yourself. Here are some tips to get you started.

Controller: Changing visible dice

The overflow menu allows the user to choose the number of visible dice: One, Two, or Three. You need to handle the selection of the menu items by calling a method in your **Controller** that sets the visibility property on each ImageView to `View.VISIBLE` to make the ImageView visible or `View.GONE` to make the ImageView invisible and not take any space.

Example (Pseudo Code):

```
private void MakeDiceVisible(int numVisible) {

   // Make dice visible
   for (…) {
      DiceImageView[i]. setVisibility(View.VISIBLE);
   }
}
```

Rolling the dice

To implement rolling the dice, the action button `action_roll` needs to start the dice rolling. The `action_stop` button should be made visible while the dice are rolling so pressing the button stops the rolling prematurely if desired. The `action_stop` button should no longer be visible when the roll is complete.

In order for the `action_stop` button's visibility property to be altered, a reference to the app bar menu needs to be saved when the menu is inflated in `onCreateOptionsMenu()`.

Updating Dice ImageView while its rolling:

Use CountDownTimer class:

https://developer.android.com/reference/android/os/CountDownTimer.html

Schedule a countdown until a time in the future, with regular notifications on intervals along the way. Example of showing a 30 second countdown in a text field:

```
 new CountDownTimer(30000, 1000) {

     public void onTick(long millisUntilFinished) {
         mTextField.setText("seconds remaining: " + millisUntilFinished /
1000);
     }

     public void onFinish() {
         mTextField.setText("done!");
     }
  }.start();
```

The calls to `onTick(long)` are synchronized to this object so that one call to `onTick(long)` won't ever occur before the previous callback is complete. This is only relevant when the implementation of `onTick(long)` takes an amount of time to execute that is significant compared to the countdown interval.