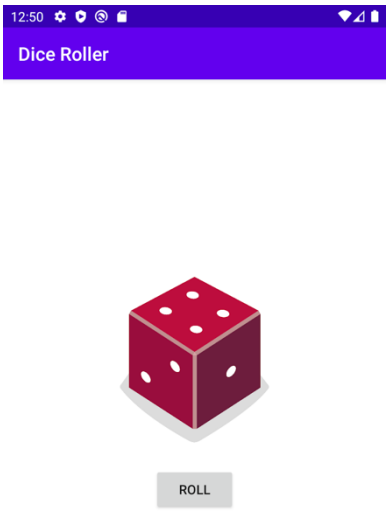
CP3406_CP5307 Codelab 2.4: Add Images to Dice Roller App

Contents

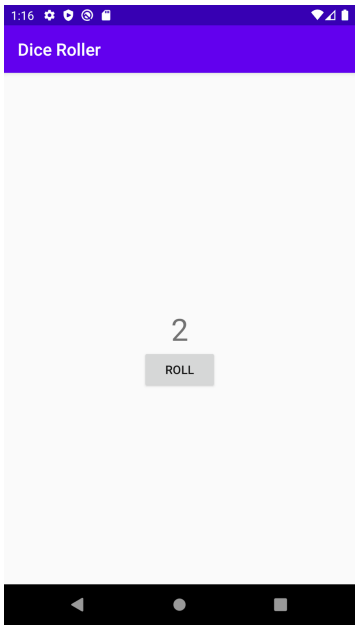


1. Overview

In this codelab, you'll add dice images to your existing Dice Roller app for Android. First, make sure tocomplete the previous codelab on the base compilation of the Dice Roller app.

Instead of showing the value of the dice launch in a `TextView`, your app will display an image of the appropriate dice corresponding to the value rolled. This will give your app more visual appeal and improved user experience.

A zip file containing the dice images is available alongside the prac worksheet in LearnJCU.



2. Update the design of the app

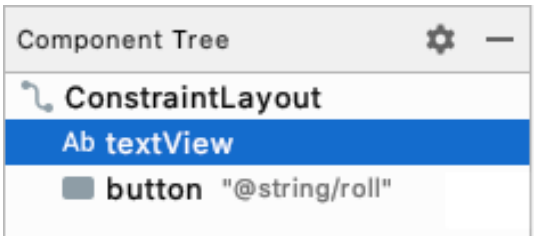
In this task, you will replace the `TextView` with an `ImageView` that shows a graphical representation of the dice roll result.

Open the Dice Roller app

1. Open and run the Dice Roller app from the previous code lab. It should look like the image displayed here.
2. Open `activity_main.xml` in the design view.

Deleting the TextView element

1. In the **design editor**, select the item `TextView` in **Component Tree**.



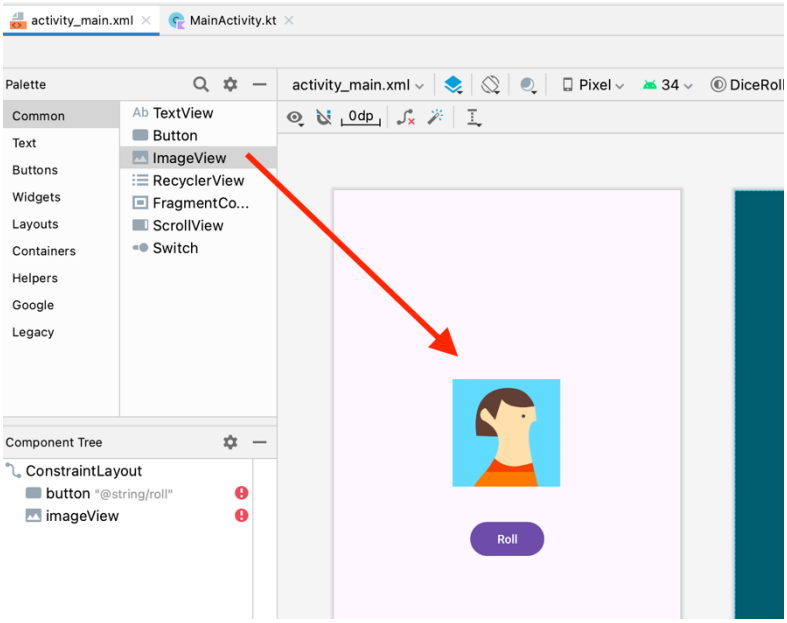
Tip: As you include more UI components and add and remove restrictions, you may find an item `View` at times overlaps with another, which will make it difficult to select the one that is behind. In that case, you

can use **Component Tree** to select the `View`.

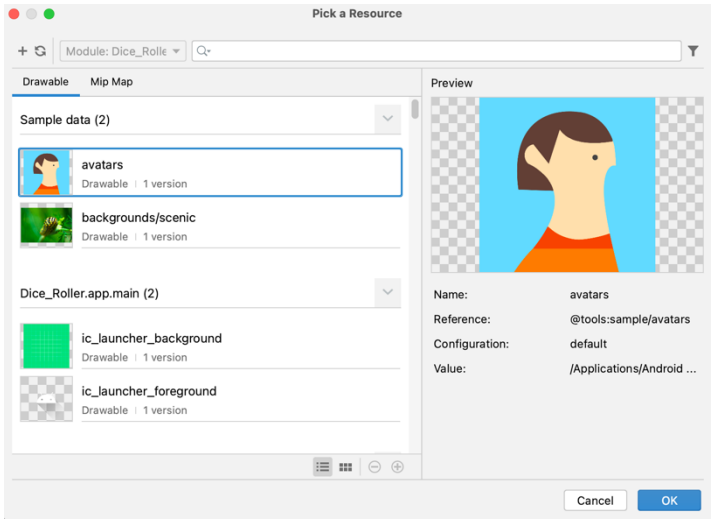
2. Right-click and select **Delete** or press the key `Delete`.
3. For the moment, ignore the warning in the `Button`. You will correct that in the next step.

Add an ImageView element to the design

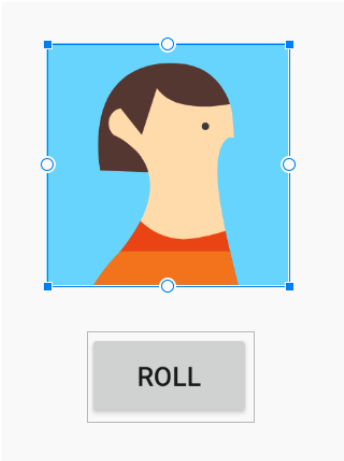
1. Drag an element `ImageView` from **Palette** to the View **Design** and place it on the `Button`.



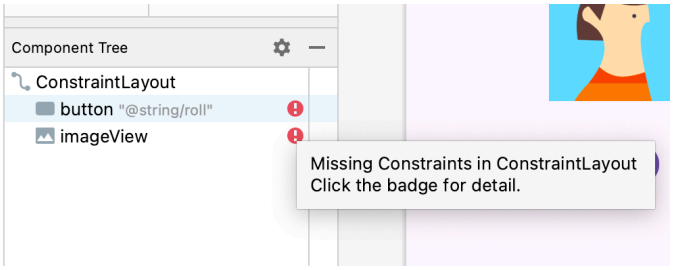
2. In the **Pick a Resource** dialog, select **avatars** in **Sample data**. This is the temporary image you will use until you add the dice images in the next task.



3. Press **OK**. The **Design** view of your app should look similar to this:

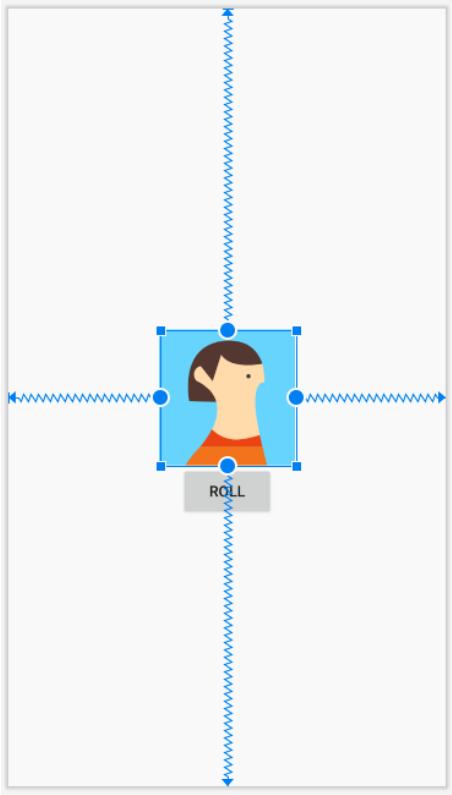


4. In **Component Tree**, you'll see two errors. The element `Button` has no vertical constraints, and the element `ImageView` has no vertical or horizontal constraints.



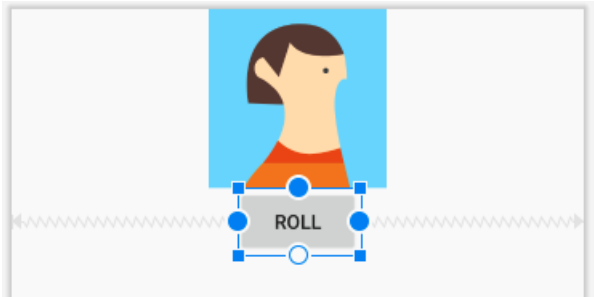
The `Button` has no vertical constraints because you removed the element `TextView` which it was constrained to. Let's add some new constraints to the item `ImageView` and the `Button`.

How to position the ImageView and Button elements



We want to position the `ImageView` vertically on the screen, regardless of the `Button`.

1. Adds constraints to the ImageView to connect it to all four sides of the ConstraintLayout, as shown in the image to the right.
2. Add a vertical restriction to `Button` by connecting the top of `Button` to the lower edge of `ImageView`. The `Button` will slide up until it's underneath the `ImageView`.



All warnings regarding the restrictions should be removed.

You may see a warning on the `ImageView` in **Component Tree** indicating that a content description should be added to your `ImageView`. For the moment, don't worry about this warning, as later in the codelab you will set the content description of `ImageView` according to which dice image is displayed. This change will be made in the Kotlin code.

3. Add the dice images

In this task, you will download some dice images and add them to your app.

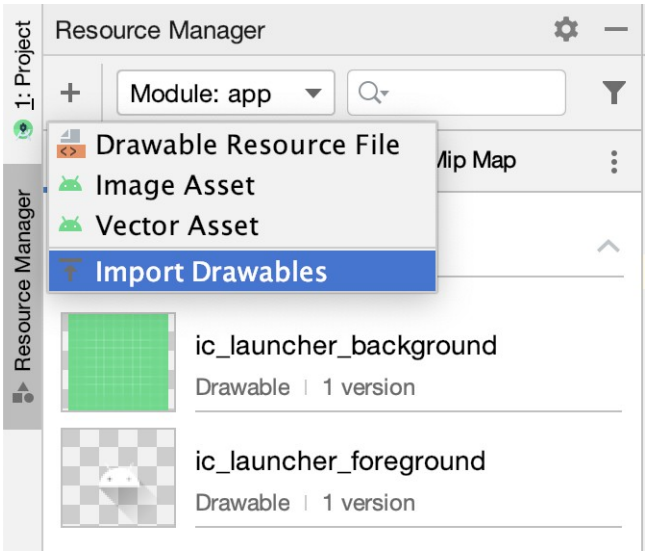
Download the dice images

1. Download the zip file with the dice images from LearnJCU.
2. Locate the file on your computer (probably in the **Downloads** folder).

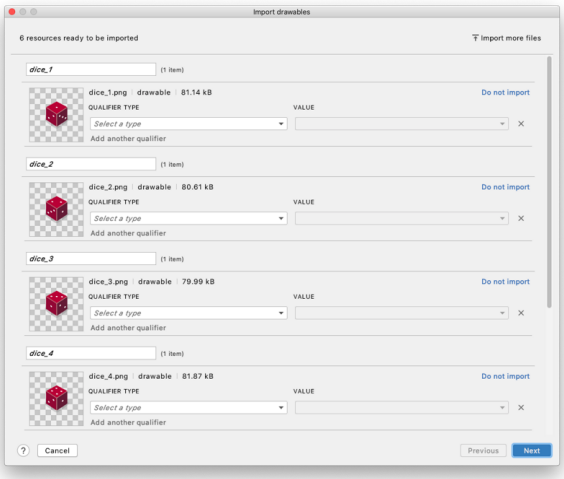
3. Double-click the ZIP file to decompress it. A folder will be created `dice_images` with the 6 dice image files, which display the dice values from 1 to 6.

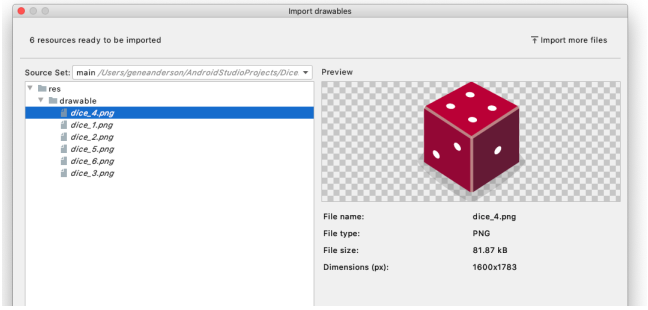
How to add dice images to your app

1. In Android Studio, click **View > Tool Windows > Resource Manager** in the menu or select the **Resource Manager** tab to the left of the **Project/Android** window.
2. Click **+** in **Resource Manager** and select **Import Drawables**. A file browser will open.

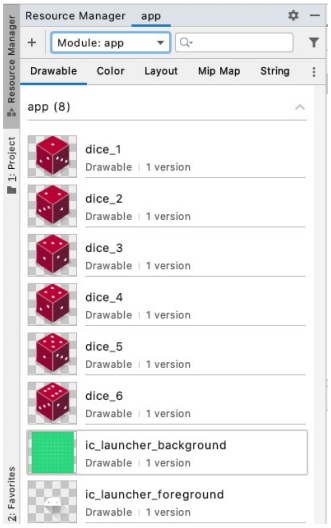


3. Search and select the 6 dice image files.
4. Click **on Open**.
5. Click **Next** and then on **Import** to confirm that you want to import these 6 resources.





6. If the files were successfully imported, the 6 images will appear in Resource Manager(**app.res.drawable)** of your app.



Good job. In the next task, you'll use these images in your app.

Note: You will refer to these images in your Kotlin code using your Resource IDs:

- `R.drawable.dice_1`
- `R.drawable.dice_2`
- `R.drawable.dice_3`
- `R.drawable.dice_4`
- `R.drawable.dice_5`
- `R.drawable.dice_6`

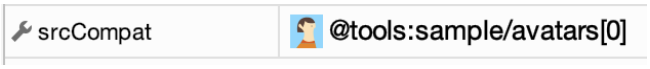
4. Use the dice images

How to replace the example avatar image in ImageView

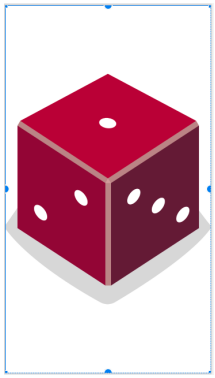
1. In **Design Editor**, select the item `ImageView`.
2. In **Attributes**, in the **Declared Attributes** section, searches for the **srcCompat** attribute of the tool, which is established as the avatar image.

The **tool's srcCompat** attribute shows the image only within the **Design** view of Android Studio. The image is only shown to developers as the app is compiled, but it won't be seen when you run it on the emulator or on a device. It's like a placeholder.

3. Click on the small preview of the avatar. A dialog box will be opened to select a new resource for you to use with this `ImageView`.



4. Select the design element `dice_1`and click **OK**.



Oh, no, the dice image occupies the whole screen!

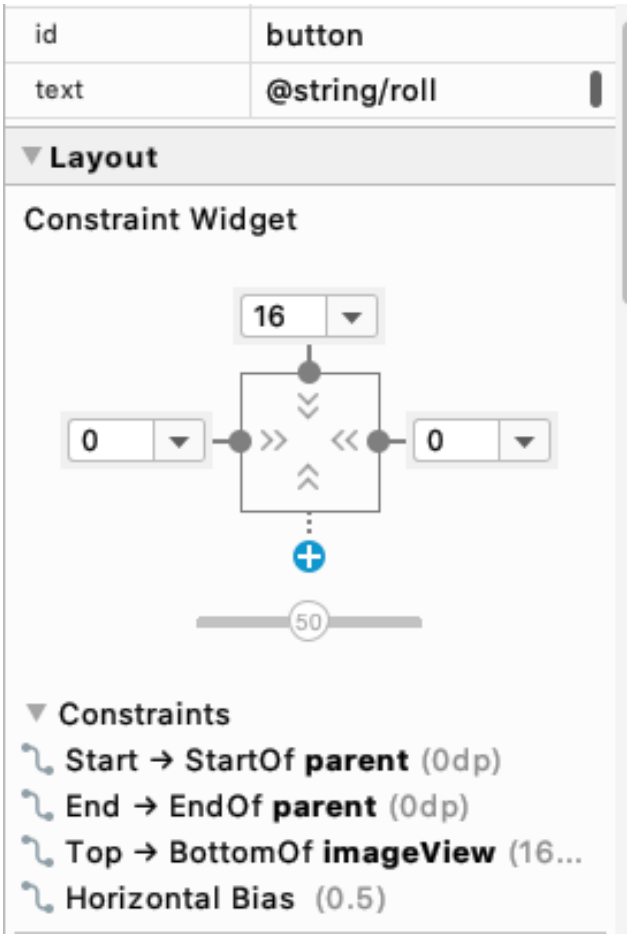
Let's adjust the width and height of the `ImageView` so that it doesn't cover the `Button`.

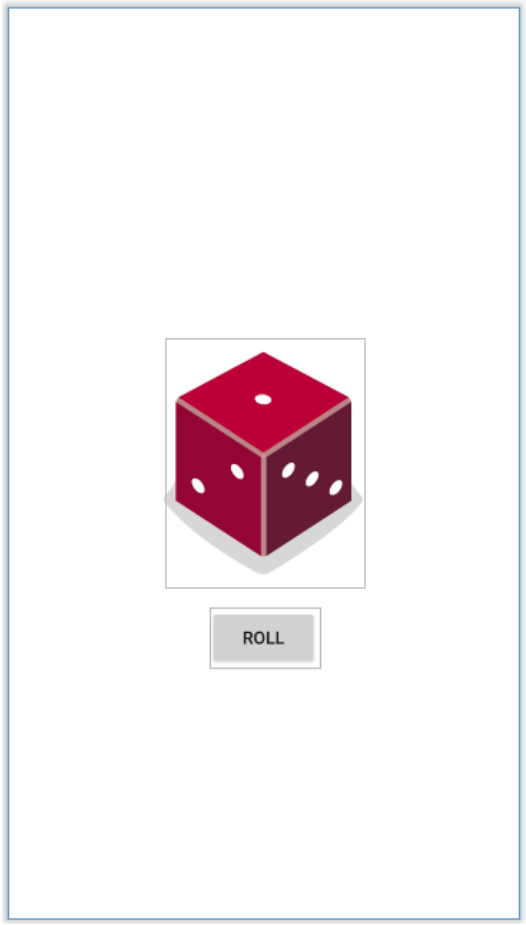
5. In the **Attributes** window under the **Constraints Widget**, look for the attributes **layout_width** and **layout_height**. Its current configuration is **wrap_content**, which means that the element `ImageView` will be as high and as wide as the content (the source image) within it.
6. Instead, lets establish a fixed width of 160dp and a fixed height of 200dp for `ImageView`. Enter those values into **layout_width** and **layout_height** respectively, pressing Enter after each.

   Woohoo, `ImageView` is now a lot smaller.

   Actually, now the `Button` is too close to the picture.

7. Add a top margin of 16dp to the Button. Configure this in the **Constraint Widget** section.





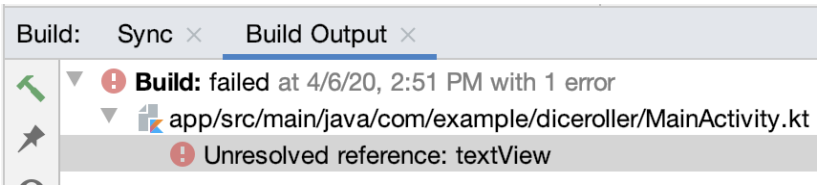
When the **Design** view is updated, the app will look much better.

Note: We use density-dependent pixels (dp) as a unit to define these dimensions, so that the image size displays properly on devices with different pixel resolutions.

How to change the image of the dice when you click on thebutton

Now that the design is updated, we must update the class `MainActivity` to use the dice images.

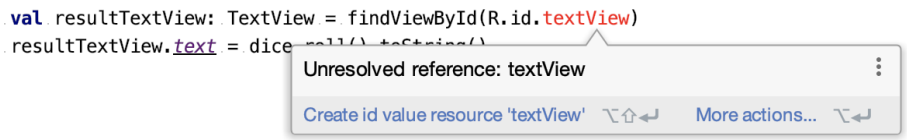
Right now, there's an error in the app, in the file `MainActivity.kt`. If you try to run it, you'll see the following build error:



This is because your code still refers to the element `TextView` you've removed from the design.

1. Open up. `MainActivity.kt`

The code refers to the element `R.id.textView` but Android Studio doesn't recognize it.



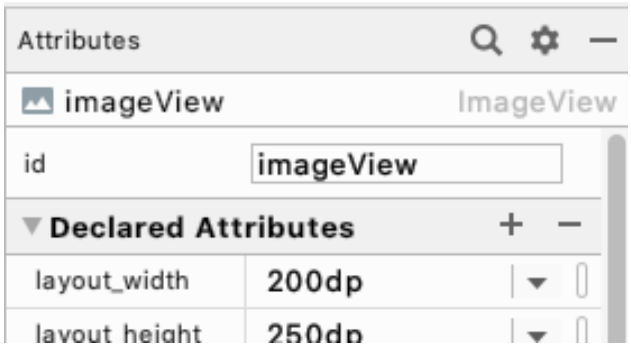
2. Within the method `rollDice()`, select the lines of code that refer to the `TextView` and remove them.

```
// Update the TextView with the dice roll
val resultTextView: TextView = findViewById(R.id.textView)
resultTextView.text = dice.roll().toString()
```

3. Still inside `rollDice()`, create a new variable called `diceImage` of type `ImageView`. Use the method `findViewById()` and pass the resource ID for `ImageView`, `R.id.imageView` as the argument.

```
val diceImage: ImageView = findViewById(R.id.imageView)
```

If you don't know how to determine the exact resource ID of `ImageView`, check the **ID** located at the top of the **Attributes** window.



4. Add this line of code to prove that you can correctly update the item `ImageView` when you click on the button. The release of the dice will not always be "2," but for we will use the image `dice_2` to test.

```
diceImage.setImageResource(R.drawable.dice_2)
```

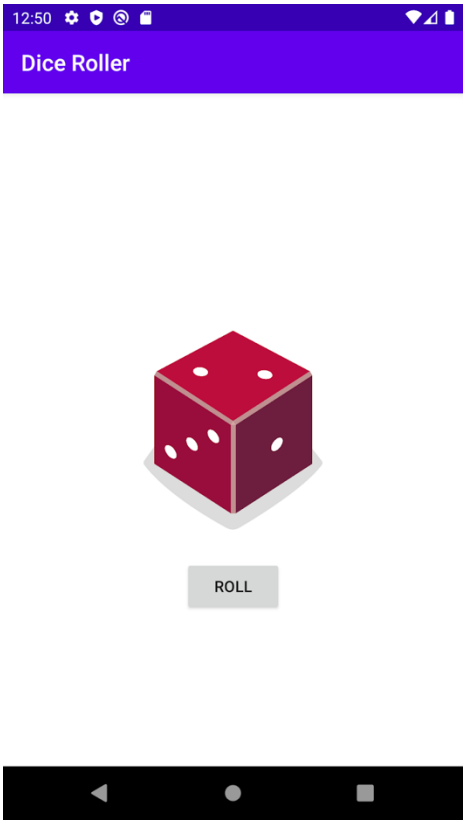
This code calls the method `setImageResource()` of `ImageView` and passes the image resource ID `dice_2`. The `ImageView` on screen will display the image `dice_2`.

The complete rollDice method() should now be:

```
private fun rollDice() {
    val dice = Dice(6)
    val diceRoll = dice.roll()
    val diceImage: ImageView = findViewById(R.id.imageView)
```

```
    diceImage.setImageResource(R.drawable.dice_2)
}
```

5. Run your app to check that no mistakes occur. The app should started with a blank screen that only contains the **Roll** button. When you press the button, the dice_2 image will display. Nice!



You could have made the image change after you press the button. You're getting closer and closer.

5. Display the correct image depending on roll

Clearly, the result of the dice roll won't always be a 2. Let's use control flow logic so that the appropriate dice imag is displayed on the screen depending on the random roll.

Before you start writing code, you should conceptually think about how the app will behave and write *pseudocode* that describes what should happen. For example:

If the user throws a 1, show the image `dice_1`.

If the user throws a 2, show the image `dice_2`.

And so on.

The previous pseudo code can be written with sentences `if/else` in Kotlin according to the value of the dice.

```
if (diceRoll == 1) {
   diceImage.setImageResource(R.drawable.dice_1)
} else if (diceRoll == 2) {
   diceImage.setImageResource(R.drawable.dice_2)
}
 ...
```

However, writing `if / else` statements for each outcome is quite repetitive. You can express the same logic in a simpler way, with `when`. This is more concise (requires less code). Use this approach in your application.

```
when (diceRoll) {
   1 ->  diceImage.setImageResource(R.drawable.dice_1)
   2 ->  diceImage.setImageResource(R.drawable.dice_2)
   ...
```

Update the rollDice() method

1. In the method `rollDice()`, erase the line of code that sets the image resource ID to always display `dice_2`.

```
diceImage.setImageResource(R.drawable.dice_2)
```

2. Replace it with a `when` statement updating the `ImageView` depending on the value of `diceRoll`.

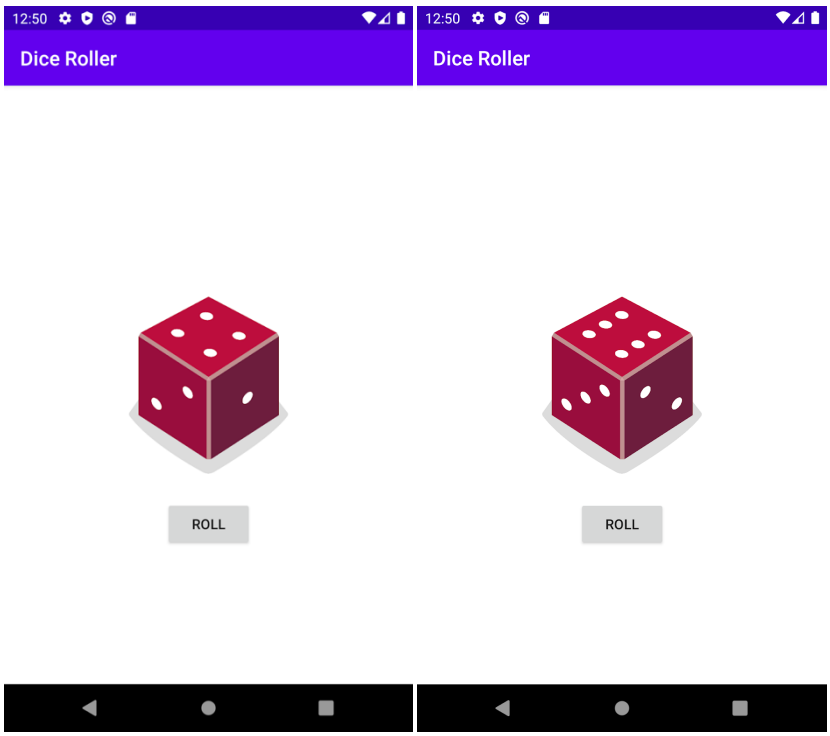
```
   when (diceRoll) {
       1 ->  diceImage.setImageResource(R.drawable.dice_1)
       2 ->  diceImage.setImageResource(R.drawable.dice_2)
       3 ->  diceImage.setImageResource(R.drawable.dice_3)
       4 ->  diceImage.setImageResource(R.drawable.dice_4)
       5 ->  diceImage.setImageResource(R.drawable.dice_5)
       6 ->  diceImage.setImageResource(R.drawable.dice_6)
   }
```

When you finish applying the changes, the method `rollDice()` should be seen as follows:

```
private fun rollDice() {
   val dice = Dice(6)
   val diceRoll = dice.roll()
   val diceImage = findViewById(R.id.imageView)
   when (diceRoll) {
```

```
            1 ->  diceImage.setImageResource(R.drawable.dice_1)
            2 ->  diceImage.setImageResource(R.drawable.dice_2)
            3 ->  diceImage.setImageResource(R.drawable.dice_3)
            4 ->  diceImage.setImageResource(R.drawable.dice_4)
            5 ->  diceImage.setImageResource(R.drawable.dice_5)
            6 ->  diceImage.setImageResource(R.drawable.dice_6)
    }
}
```

3. Run the app. If you click on the **Roll** button, the picture of the dice will change and show a different value each time. It works!



How to optimize your code

If you want to write an even more concise code, you can make the change shown below. It has no visible effect to the user of your app, but the code will be shorter and less repetitive.

Maybe you noticed the call to `diceImage.setImageResource()` - it appears 6 times!

```
when (diceRoll) {
    1 ->  diceImage.setImageResource(R.drawable.dice_1)
    2 ->  diceImage.setImageResource(R.drawable.dice_2)
    3 ->  diceImage.setImageResource(R.drawable.dice_3)
    4 ->  diceImage.setImageResource(R.drawable.dice_4)
    5 ->  diceImage.setImageResource(R.drawable.dice_5)
    6 ->  diceImage.setImageResource(R.drawable.dice_6)
}
```

The only thing that changes between each case is the resource ID used. This means you can create a variable to store the resource ID that will be used. Then you can call `diceImage.setImageResource()` only once in your code and pass the correct resource ID.

1. Replace the previous code with the following:

```
val drawableResource = when (diceRoll) {
   1 -> R.drawable.dice_1
   2 -> R.drawable.dice_2
   3 -> R.drawable.dice_3
   4 -> R.drawable.dice_4
   5 -> R.drawable.dice_5
   6 -> R.drawable.dice_6
}
diceImage.setImageResource(drawableResource)
```

A new concept here is that an expression `when` can actually be a variable value. With this new fragment of code, the expression `when` figures out the correct resource ID, which will be stored in the variable `drawableResource`. You can use that variable to update the image resource that will be displayed.

2. Please note that `when` is now underlined in red. If you put the cursor on it, you'll see an error message: **'when' expression must be exhaustive, add the necessary 'else' branch**.

The error is because you must handle all possible cases so that a value is always shown. What if we somehow rolled 8 (we could switch to a 12-sided dice)?. Android Studio suggests adding a branch `else`. To solve this problem, change the case of `6` to `else`. The cases of `1` to `5` remain the same, but `else` handles all other possible values, including `6`.

```
val drawableResource = when (diceRoll) {
   1 -> R.drawable.dice_1
   2 -> R.drawable.dice_2
   3 -> R.drawable.dice_3
   4 -> R.drawable.dice_4
   5 -> R.drawable.dice_5
   else -> R.drawable.dice_6
}diceImage.setImageResource(drawableResource)
```

3. Run the app to check it's still working properly. Make sure you do enough tests to make sure that all the numbers appear with the dice images from 1 to 6.

How to set a proper content description in the ImageViewelement

Now that you replaced the number released with an image, screen readers will no longer be able to determine what number is rolled. To solve this problem, after updating the image resource, update the content description of the `ImageView`. This should be a text description of what is shown in the element `ImageView` so that screen readers can describe it.

```
diceImage.contentDescription = diceRoll.toString()
```

Screen readers can read this content description aloud. Therefore, if the image of the "6" value is

displayed on the screen, the content description will be read aloud as "6."

Note: Normally, a content description should use strings resources that can be translated into other languages, but for now this is fine.

6. Implement best coding practice

Create a better user experience

When the user first opens the app, it is blank (it only contains the **Roll** button), which looks strange. As users may not know what to expect, the UI should show a random dice roll when the app first starts and the element is created in the `Activity`. In this way, users are more likely to understand that, if they press the **Roll** button, a roll of the dice will be generated.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val rollButton: Button = findViewById(R.id.button)
    rollButton.setOnClickListener { rollDice() }
    // Do a dice roll when the app starts
    rollDice()
}
```

Add comments to your code

Add some comments to the code to describe what will happen under the code you wrote.

After all these changes, that's how your method could look like `rollDice()`:

```
/**
* Roll the dice and update the screen with the result.
*/
private fun rollDice() {
    // Create new Dice object with 6 sides and roll the dice
    val dice = Dice(6)
    val diceRoll = dice.roll()
    // Find the ImageView in the layout
    val diceImage: ImageView = findViewById(R.id.imageView)
    // Determine which drawable resource ID to use based on the dice roll
    val drawableResource = when (diceRoll) {
        1 -> R.drawable.dice_1
        2 -> R.drawable.dice_2
```

```
            3 -> R.drawable.dice_3
            4 -> R.drawable.dice_4
            5 -> R.drawable.dice_5
            else -> R.drawable.dice_6
        }
        // Update the ImageView with the correct drawable resource ID
        diceImage.setImageResource(drawableResource)
        // Update the content description
        diceImage.contentDescription = diceRoll.toString()
    }
```

We congratulate you on completing the Dice Roller app. You're now ready for your next board game night!

7. Summary

- Use `setImageResource()` to change the image shown in an `ImageView`.
- Use control flow expressions, `if/else` or `when`, in order to handle different cases in your app, for example, to show different images in different circumstances.