TD3 Dice Roller - Kotlin

## 

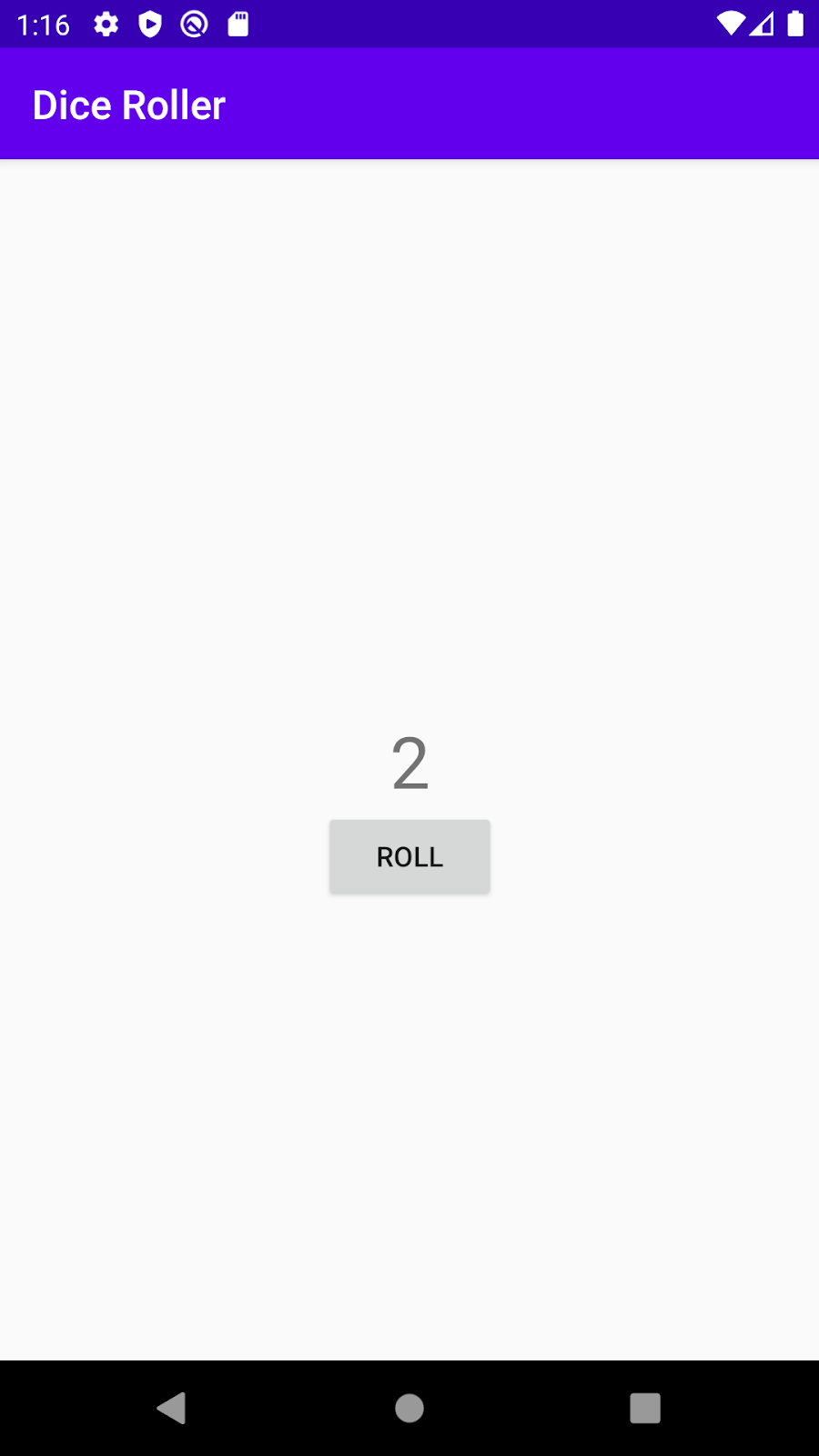
1. Prérequis

Dans ce td, vous allez créer une application Android Dice Roller où les utilisateurs peuvent cliquer sur un bouton dans l'application pour lancer un dé. Le résultat du lancé sera affiché dans un TextView à l'écran. Vous allez utiliser l'éditeur de mise en page dans Android Studio pour créer la vue de votre application, puis écrire du code Java pour savoir ce qui se passe lorsque vous cliquez sur le bouton.

Les acquis du Td :

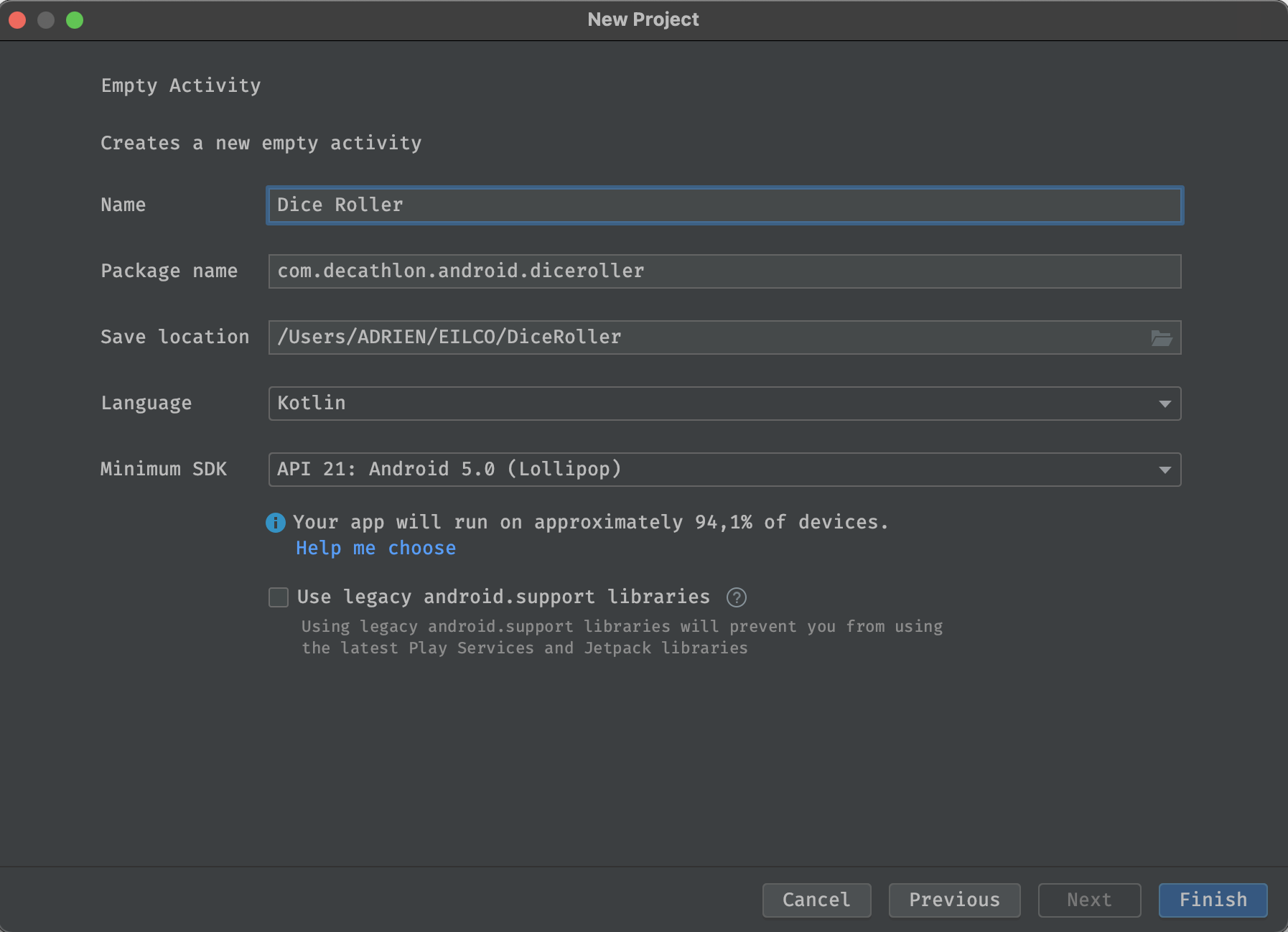
* Comment ajouter un bouton à une application Android.
* Comment ajouter un comportement lorsqu'un bouton est appuyé dans l'application.
* Comment ouvrir et modifier le code d'activité d'une application.
* Comment afficher un message Toast.
* Comment mettre à jour le contenu d'un TextView pendant que l'application est en cours d'exécution.

Voici à quoi ressemblera l'application lorsque vous aurez terminé ce td



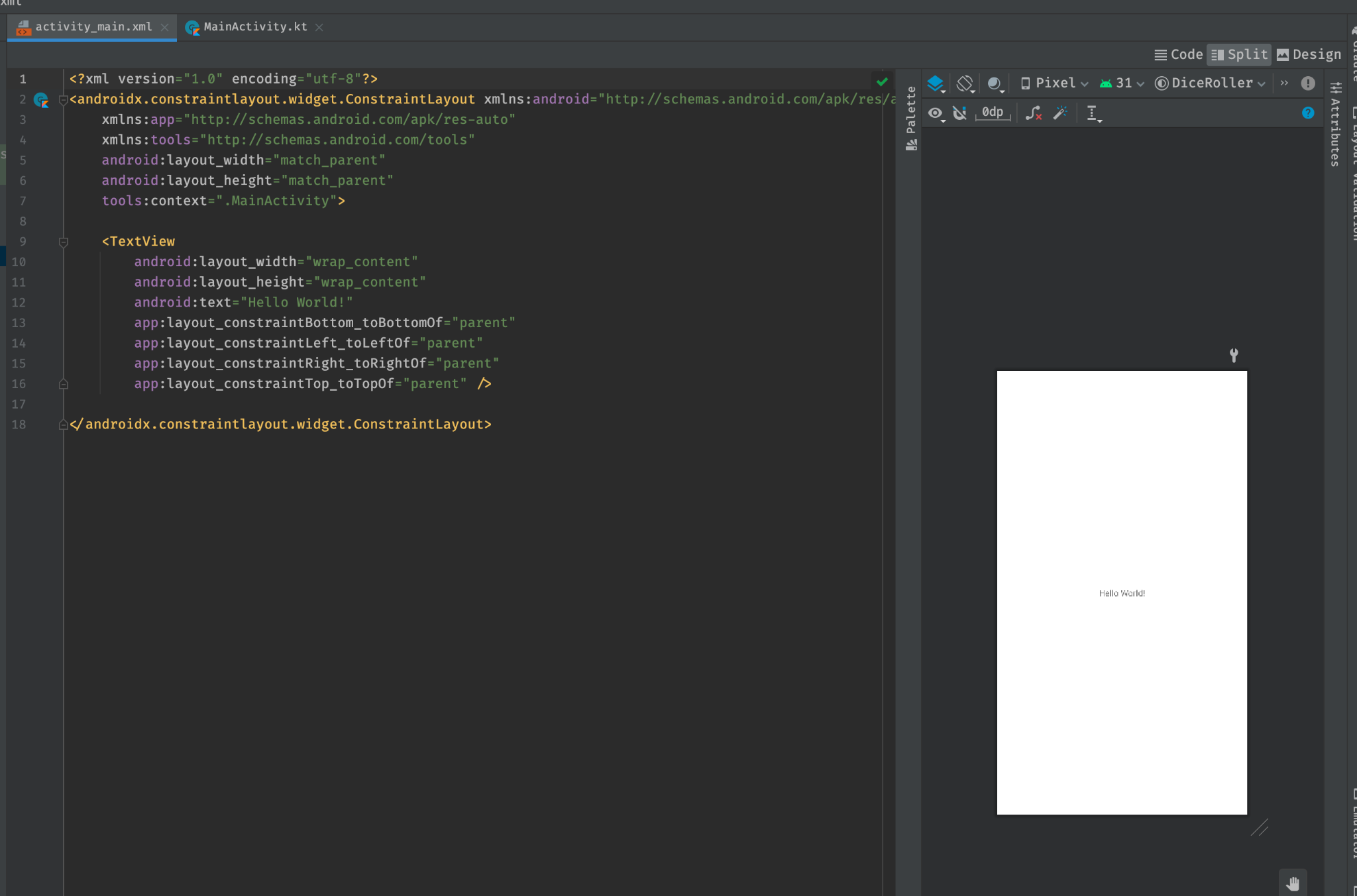
1. Configurez votre application

Créer un nouveau projet en utilisant le template Empty Activity. Nommer votre application Dice Roller.



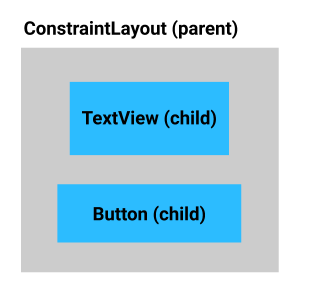
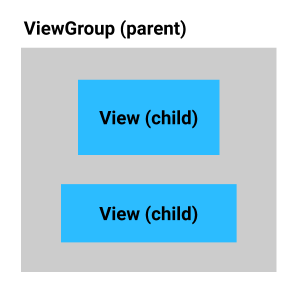
1. Création de la vue principale

Dans la fenêtre Projet, double-cliquez sur activity\_main.xml (app> res> layout> activity\_main.xml) pour l'ouvrir. Vous devriez voir l'éditeur de vue, avec uniquement le TextView "Hello World" au centre de l'application.



Ajoutez maintenant un bouton sous le TextView "Hello World". Le TextView et le Button seront situés dans un ConstraintLayout, qui est un type de ViewGroup.

Lorsqu'il y a des vues dans un ViewGroup, les Views sont considérées comme des enfants du ViewGroup parent. Dans le cas de votre application, TextView et Button seraient considérés comme des enfants du ConstraintLayout parent.

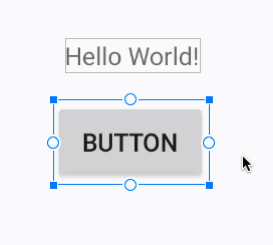


Remarque: comme dans un arbre généalogique, dans une hiérarchie de vues, les vues parentes peuvent elles-mêmes être des vues enfants et les vues enfants peuvent être les parents d'autres enfants.

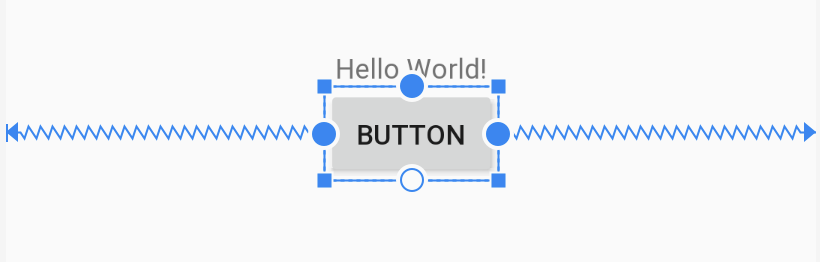
1. Ajouter un bouton au layout

| <Button  android:id="@+id/button"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Button" /> |
| --- |

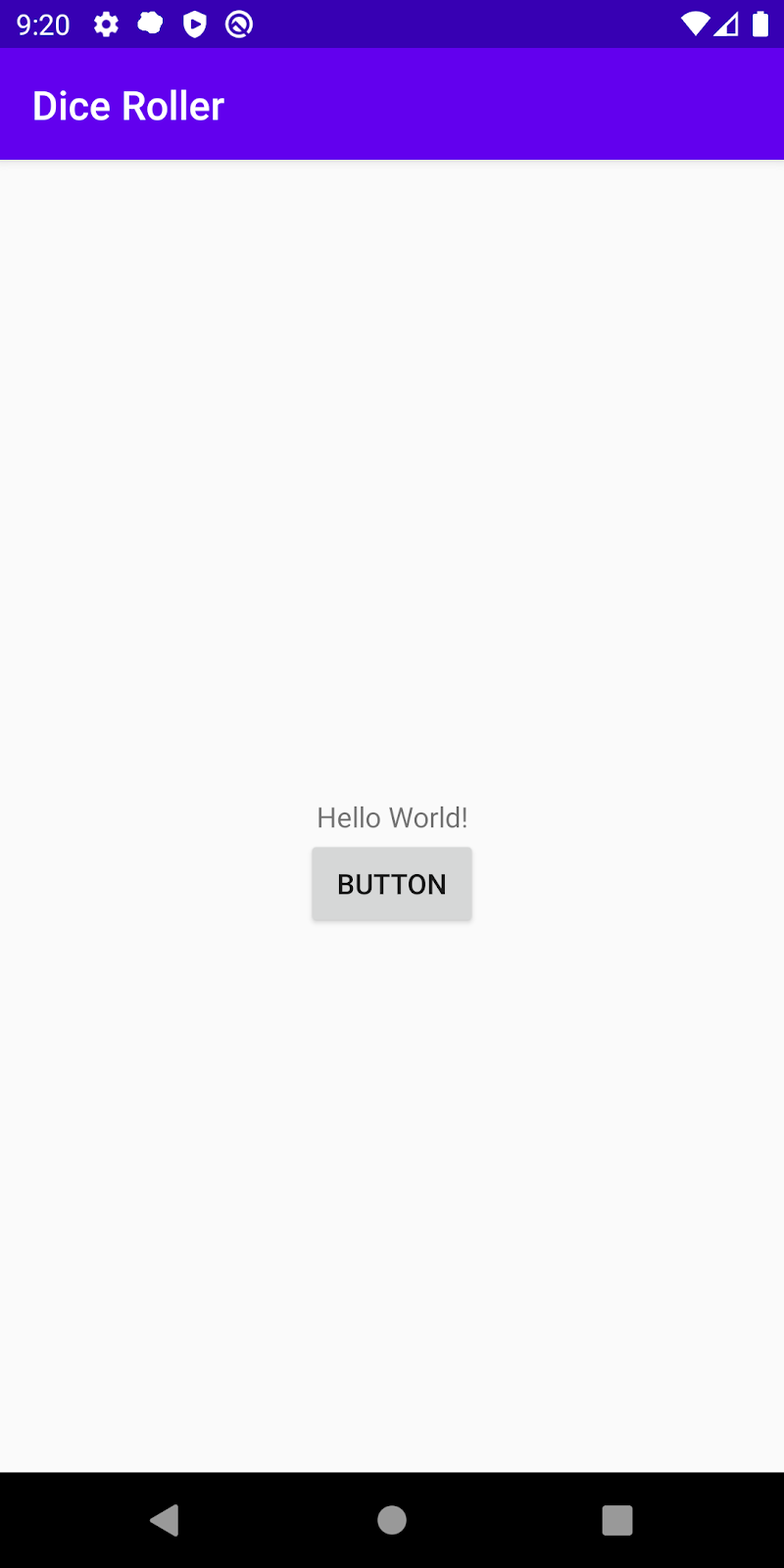
* Ajouter une contrainte verticale du haut du Button au bas de TextView. Cela positionnera le bouton sous le TextView.



* Ajouter une contrainte horizontale du côté gauche du Button au côté gauche du ConstraintLayout parent



Exécutez l'application. Cela devrait ressembler à la capture d'écran ci-dessous. Vous pouvez cliquer sur le bouton, mais il ne fait encore rien. Continuons!



* Changer le texte du bouton par Lancer le dé. N’oubliez pas d’extraire la chaîne de caractère de le fichier resource strings.xml
* Changer le texte du TextView pour qu’il n’affiche rien pour le moment, il affichera la valeur du lancé de dé par la suite.
* Nommez de manière explicite l’id du bouton ainsi que celui du TextView, cela nous servira pour retrouver cet id dans le code Java

Exécutez votre application. Voici à quoi ressemble l'application. Le texte du TextView ne s'affiche pas pour le moment. C'est le bon comportement.



Vous avez donc une application avec un bouton, mais si vous appuyez sur le bouton, rien ne se passe. Pour changer cela, vous devrez écrire du code Kotlin qui lance le dé et met à jour l'écran lorsque vous appuyez sur le bouton.

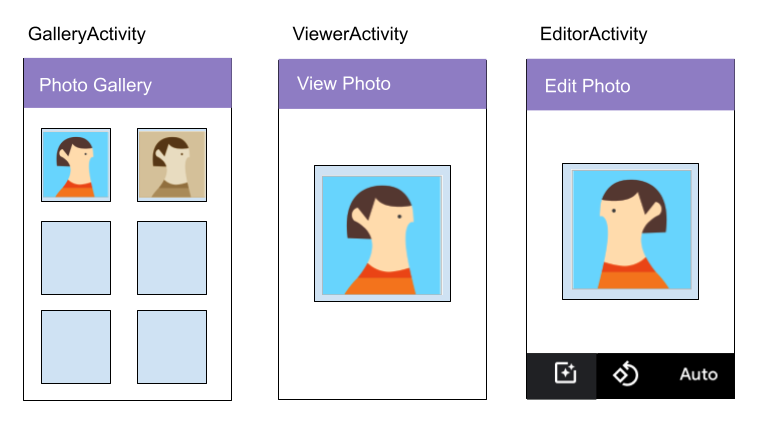
Pour effectuer ce changement, vous devez comprendre un peu plus comment une application Android est structurée.

1. Introduction aux activités

Une activité fournit la fenêtre dans laquelle votre application dessine son interface utilisateur. En règle générale, une activité occupe tout l'écran de votre application en cours d'exécution. Chaque application a une ou plusieurs activités. L'activité de niveau supérieur ou première est souvent appelée MainActivity et est fournie par le modèle de projet. Par exemple, lorsque l'utilisateur fait défiler la liste des applications sur son appareil et appuie sur l'icône de l'application «Dice Roller», le système Android démarre l'activité principale de l'application.

Pour les applications plus compliquées, il peut y avoir plusieurs écrans et plus d'une activité. Chaque activité a un but spécifique.

Par exemple, dans une application de galerie de photos, vous pouvez avoir une activité pour afficher une grille de photos, une deuxième activité pour afficher une photo individuelle et une troisième activité pour éditer une photo individuelle.



1. Rendre le bouton interactif

Vous allez ajouter un code pour répondre à une pression sur un bouton dans MainActivity.

Accédez au fichier MainActivity.kt et ouvrez-le (app> kotlin> com.example.diceroller> MainActivity.kt). Vous obtenez alors le code suivant.

| package com.decathlon.android.diceroller  import androidx.appcompat.app.AppCompatActivity  import android.os.Bundle  class MainActivity : AppCompatActivity() {  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContentView(R.layout.*activity\_main*)  }  } |
| --- |

Vous n'avez pas besoin de comprendre chaque mot du code ci-dessus, mais vous devez avoir une idée générale de ce qu'il fait. Plus vous travaillez avec le code Android, plus il deviendra familier et plus vous le comprendrez.

Nous allons maintenant afficher un message lorsque l'utilisateur clique sur le bouton. Dans cette étape, vous spécifierez que lorsque vous cliquez sur le bouton, un bref message s'affiche en bas de l'écran.

Ajoutez le code suivant à la méthode onCreate () après l'appel setContentView (). La méthode findViewById () recherche le Button dans la disposition. R.id.button est l'ID de ressource du Button, qui est un identifiant unique pour celui-ci. Le code enregistre une référence à l'objet Button dans une variable appelée rollButton, pas l'objet Button lui-même.

| val rollButton = findViewById<Button>(R.id.*button*) |
| --- |

Remarque: Android attribue automatiquement des numéros d'identification aux ressources de votre application. Par exemple, le bouton Roll a un ID de ressource et la chaîne du texte du bouton a également un ID de ressource. Les ID de ressources sont de la forme R. <type>. <nom>; par exemple, R.string.roll. Pour les ID de vue, le <type> est id, par exemple, R.id.button.

La méthode onCreate () devrait maintenant ressembler à ceci.

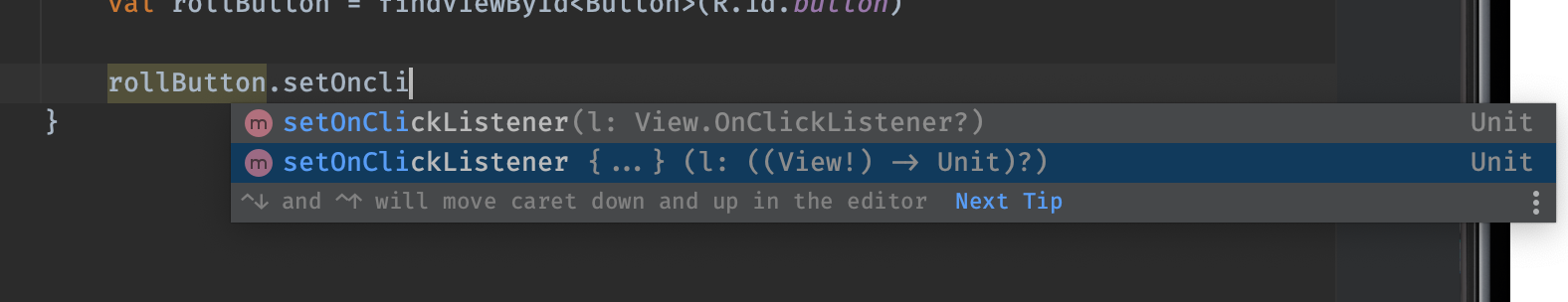
| class MainActivity : AppCompatActivity() {  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContentView(R.layout.*activity\_main*)  val rollButton = findViewById<Button>(R.id.*button*)    }  } |
| --- |

Ensuite, vous devez associer du code au Button, afin que le code puisse être exécuté lorsque le Button est touché. Un écouteur de clic (listener) est un code indiquant ce qu'il faut faire lorsqu'un tap ou un clic se produit. Vous pouvez le considérer comme un code qui attend, "écoutant" pour que l'utilisateur clique, dans ce cas, sur le bouton.

Utilisez l'objet rollButton et définissez un listener dessus en appelant la méthode setOnClickListener ().

| rollButton.setOnClickListener **{**    **}** |
| --- |

Au fur et à mesure que vous tapez, Android Studio peut afficher plusieurs suggestions. Dans ce cas, choisissez l'option setOnClickListener {...} et AndroidStudio fait le reste pour vous



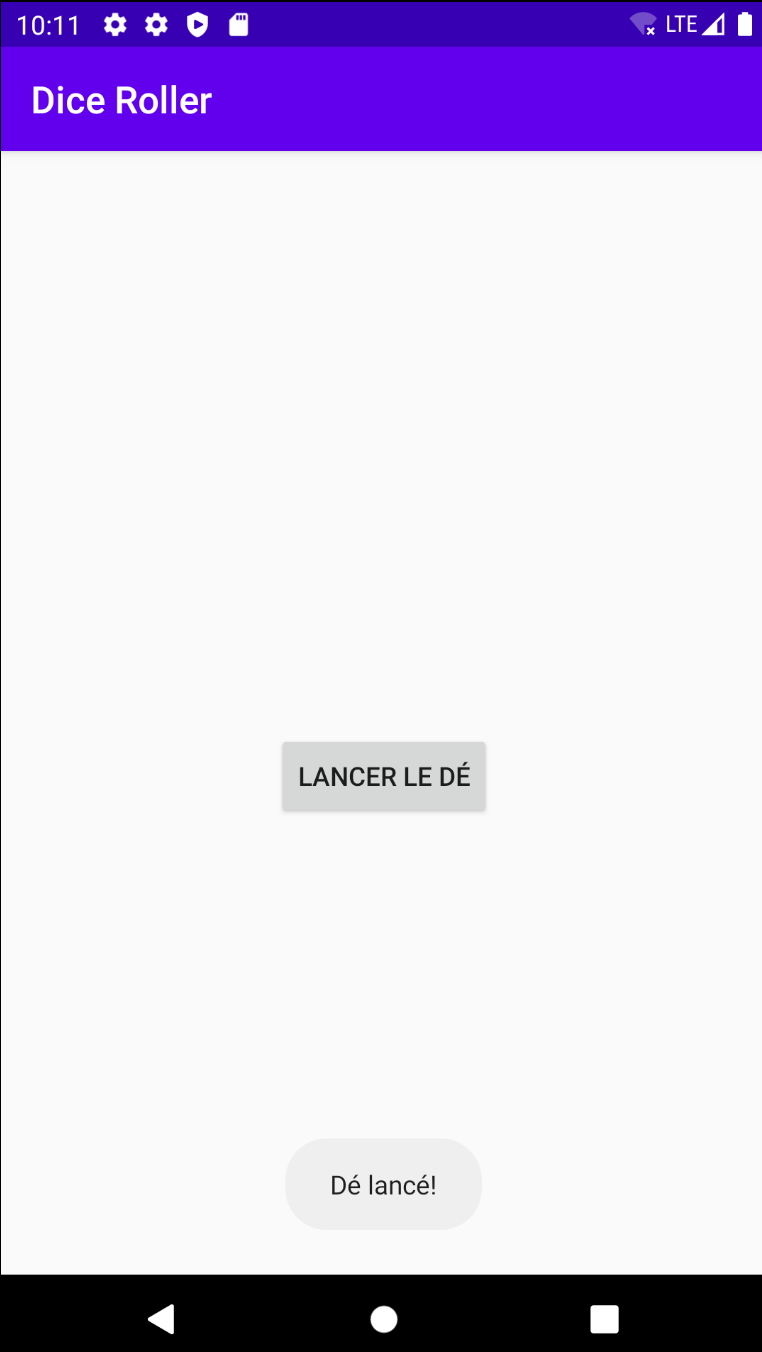
Dans la lambda {}, mettez des instructions sur ce qui doit se passer lorsque vous appuyez sur le bouton. Pour l'instant, votre application affiche un toast, qui est un bref message qui apparaît à l'utilisateur.

Créez un toast avec le texte "Dé lancé" en appelant Toast.makeText ().

| Toast.makeText(*applicationContext*, "Dé lancé", Toast.*LENGTH\_SHORT*).show() |
| --- |

Voici à quoi ressemble votre mise à jour de la classe MainActivity

| class MainActivity : AppCompatActivity() {  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContentView(R.layout.*activity\_main*)  val rollButton = findViewById<Button>(R.id.*button*)  rollButton.setOnClickListener **{**  Toast.makeText(*applicationContext*, "Dé lancé", Toast.*LENGTH\_SHORT*).show()  **}**  }  } |
| --- |

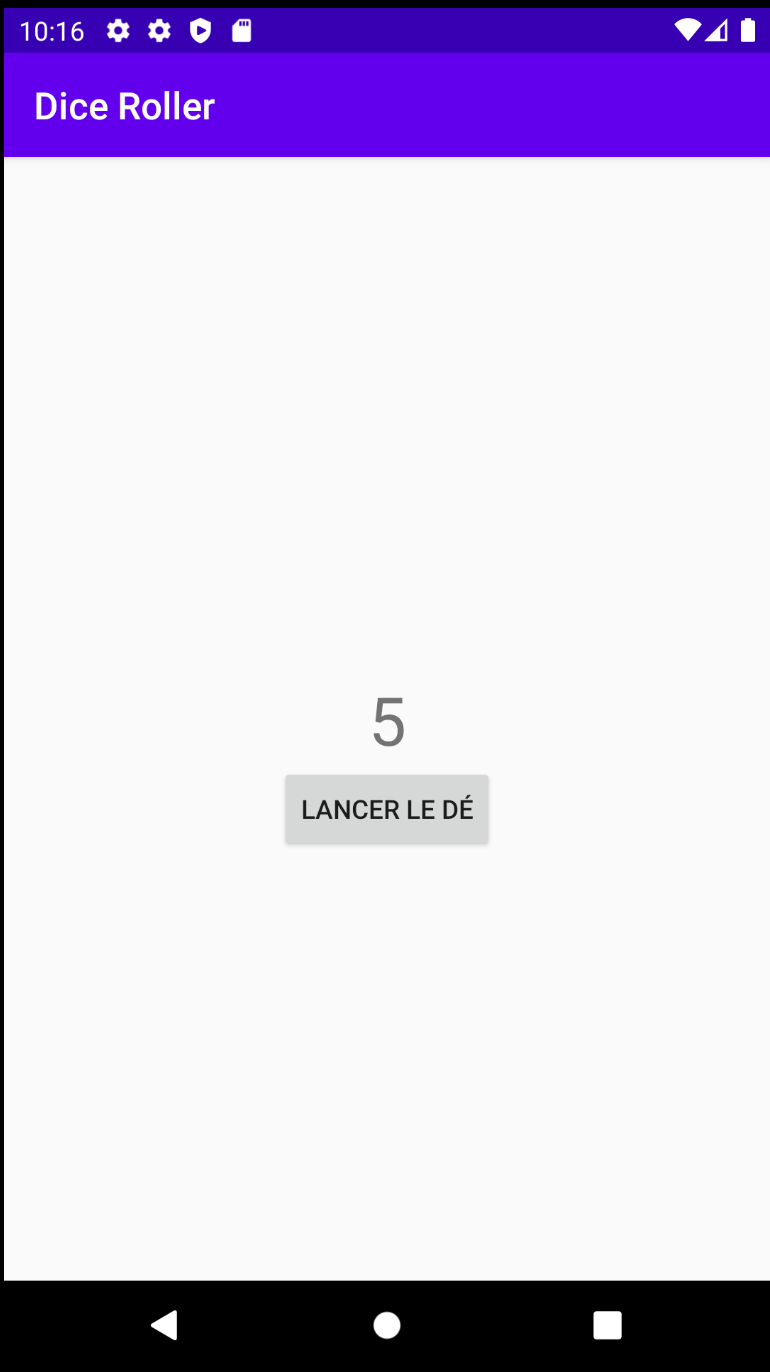


Hourra! Le clic du bouton a fait apparaître le message !

Exercice 1:

Faites en sorte que maintenant au lieu d’afficher un Toast, vous mettrez à jour le texte du TextView avec un valeur prise au hasard comprise entre 1 et 6.

Attention, veuillez commenter votre code pour expliquer la logique de vos fonctions.



Exercice 2:

Faites de même avec deux dés distincts pour obtenir deux résultats.

Vous obtiendrez donc 2 TextViews et un bouton lancer les dés.

Exercices 3: Ajoutez maintenant un EditText pour demander à l’utilisateur de choisir le nombre de face des dés. Puis utiliser ce résultat pour le lancer de dés.