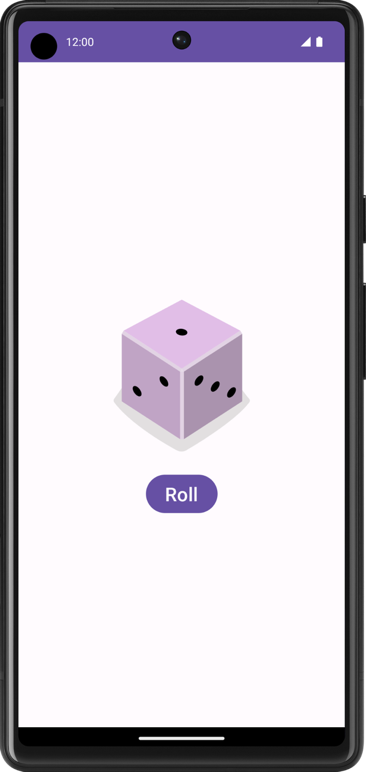
**Лабораторная работа: приложение Dice Roller**

# [Создайте интерактивное приложение Dice Roller](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app)

В этой лабораторной работе вы создадите интерактивное приложение **Dice Roller** , которое позволяет пользователям нажимать на Buttonсоставной элемент, чтобы бросать игральные кости. Результат броска отображается в виде Imageкомпоновки на экране.

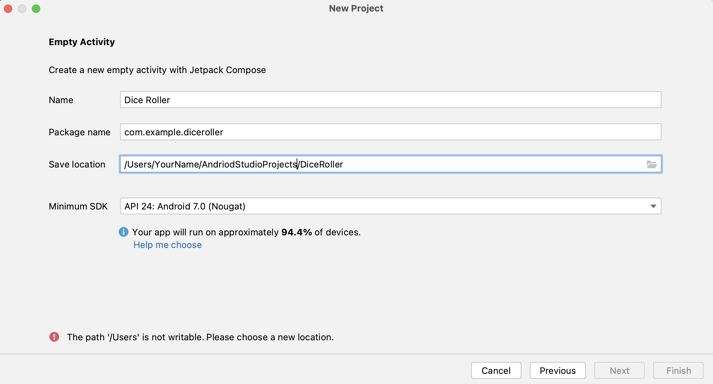
Вот как будет выглядеть приложение:



## [**2. Установите базовый уровень**](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app#1)

## **Создать проект**

1. Создайте проект EmptyActivity с именем Dice Roller.



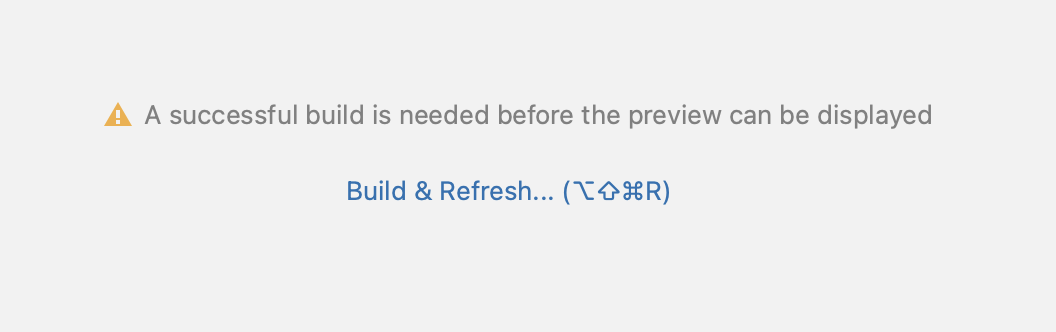
**Примечание.** Путь, показанный на изображении, недействителен и используется только в качестве примера. Недопустимый путь генерирует предупреждение: «Путь «/Users» недоступен для записи. Выберите новое местоположение». Это предупреждение не появится, если путь действителен, и его можно игнорировать в изображении.

## [**3. Создайте инфраструктуру макета**](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app#2)

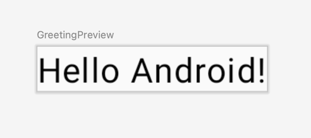
## **Предварительный просмотр проекта**

Чтобы просмотреть проект:

* Нажмите **«Build & Refresh»** на панели **«Split»** или **«Design»** .



Теперь вы должны увидеть предварительный просмотр на панели **«Design»** . Если он выглядит маленьким, не волнуйтесь, поскольку он меняется при изменении макета.



## **Реструктуризация примера кода**

Вам необходимо изменить часть сгенерированного кода, чтобы он больше напоминал тему приложения для игры в кости.

Как вы видели на скриншоте финальной версии приложения, здесь есть изображение игральной кости и кнопка для ее броска. Вы структурируете компонуемые функции, чтобы отразить эту архитектуру.

Чтобы реструктурировать пример кода:

1. Удалить функцию GreetingPreview().
2. Создайте функцию DiceWithButtonAndImage() с аннотацией @Composable.

Эта составная функция представляет компоненты пользовательского интерфейса макета, а также содержит логику нажатия кнопок и отображения изображений.

1. Удалить функцию Greeting(name: String).
2. Создайте функцию DiceRollerApp()с аннотациями @Preview и @Composable.

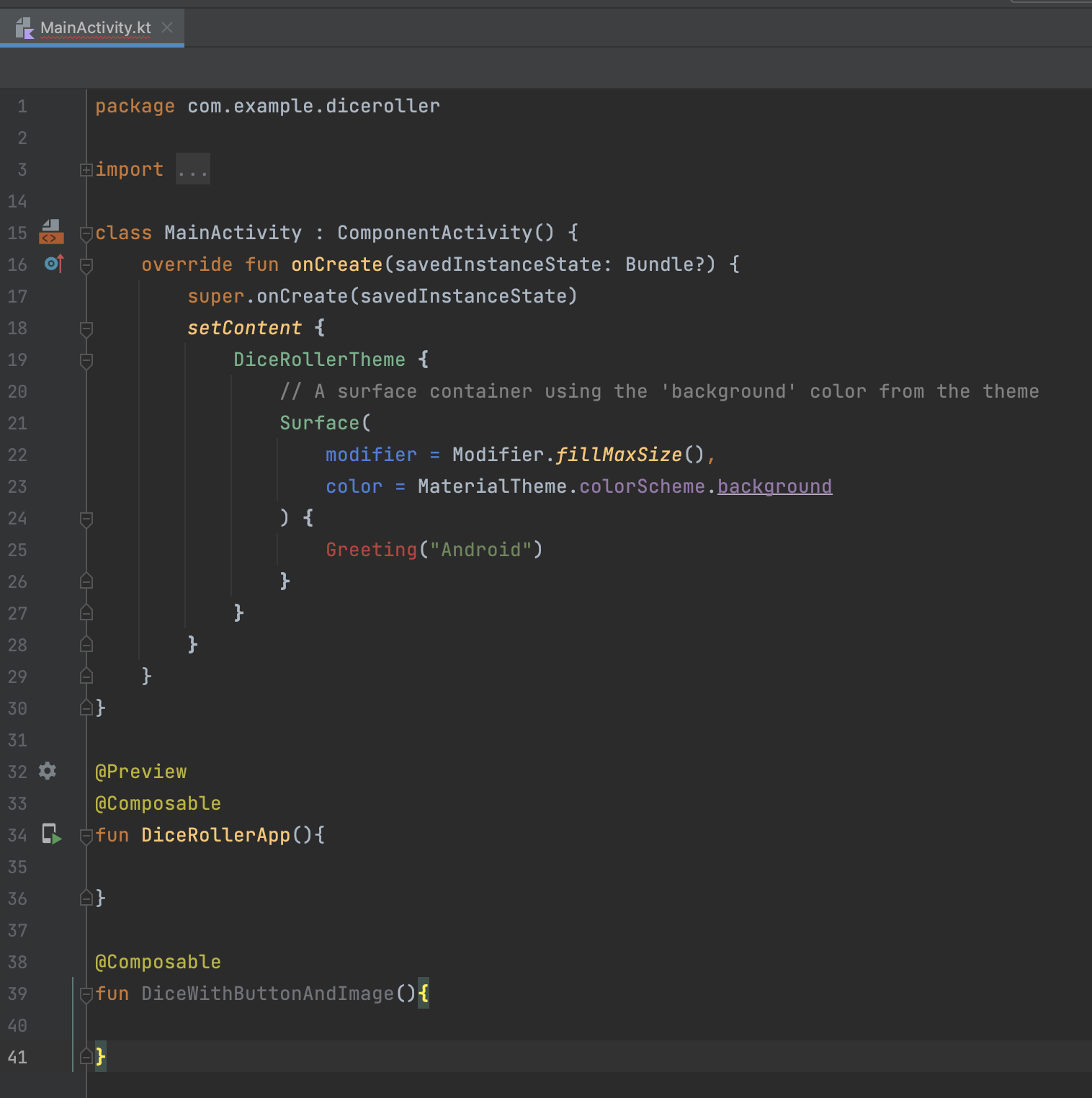
Поскольку это приложение состоит только из кнопки и изображения, воспринимайте эту составную функцию как само приложение. Вот почему это называется функцией DiceRollerApp().

MainActivity.kt

@Preview  
@Composable  
fun DiceRollerApp() {  
}

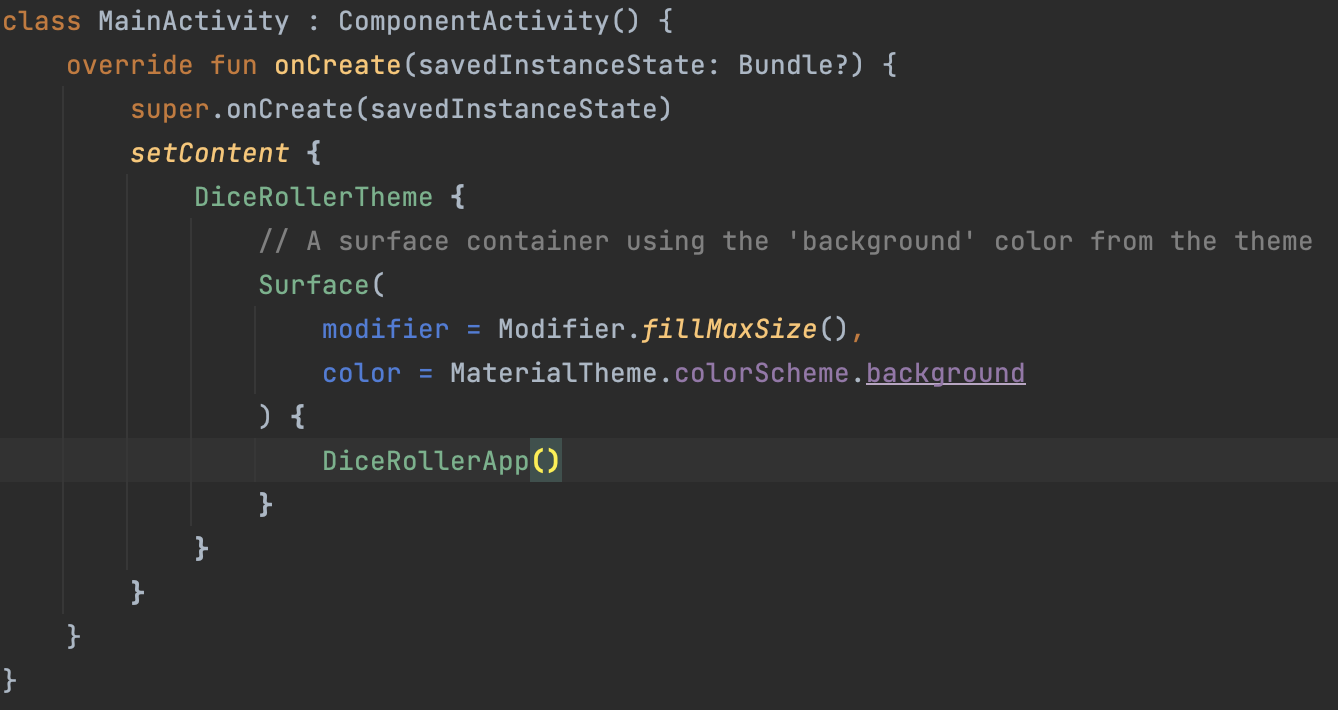
@Composable  
fun DiceWithButtonAndImage() {  
  
}

Поскольку вы удалили  функцию Greeting(), вызов Greeting("Android") в теле лямбды DiceRollerTheme() выделен красным. Это потому, что компилятор больше не может найти ссылку на эту функцию.



1. Измените код : внутри лямбды DiceRollerTheme{} вызовите функцию DiceRollerApp().

MainActivity.kt



1. В функции DiceRollerApp()вызовите функцию DiceWithButtonAndImage().

MainActivity.kt

## 

## **Добавить модификатор**

Compose использует объект [Modifier](https://developer.android.com/reference/kotlin/androidx/compose/ui/Modifier) - набор элементов, которые украшают или изменяют поведение элементов пользовательского интерфейса Compose.

Чтобы добавить модификатор:

1. Измените функцию DiceWithButtonAndImage(), чтобы она принимала аргумент modifier типа Modifier и присваивала ему значение по умолчанию Modifier.

MainActivity.kt



Разберем код.

Функция позволяет передавать параметр modifier. Значением параметра modifier по умолчанию является объект Modifier, следовательно часть сигнатуры метода. Значение параметра по умолчанию = Modifier позволяет любому, кто вызовет этот метод в будущем, решить, передавать ли значение параметра. Если они передадут свой собственный объект Modifier, они смогут настроить поведение и оформление пользовательского интерфейса. Если они решают не передавать объект Modifier, он принимает значение по умолчанию, то есть простой объект Modifier. Вы можете применить эту практику к любому параметру. Дополнительные сведения об аргументах по умолчанию [Аргументы по умолчанию](https://kotlinlang.org/docs/functions.html#default-arguments) .

***Примечание****. Этот оператор import androidx.compose.ui.Modifier импортирует пакет androidx.compose.ui.Modifier, который позволяет вам ссылаться на объект Modifier.*

1. Теперь, когда у составного объекта DiceWithButtonAndImage() есть параметр-модификатор, передайте модификатор при вызове составного объекта. Поскольку сигнатура метода функции DiceWithButtonAndImage() изменилась, при ее вызове необходимо передать объект Modifier с желаемыми украшениями. Класс Modifier отвечает за украшение или добавление поведения к компонуемой функции DiceRollerApp(). В этом случае к объекту Modifier, передаваемому в функцию, нужно добавить несколько важных украшений DiceWithButtonAndImage().

Вы можете задаться вопросом, зачем вообще передавать аргумент Modifier, если есть значение по умолчанию. Причина в том, что компонуемые объекты могут подвергаться рекомпозиции, что по сути означает, что блок кода в @Composable методе выполняется снова. Если объект Modifier создается в блоке кода, его потенциально можно создать заново, а это неэффективно.

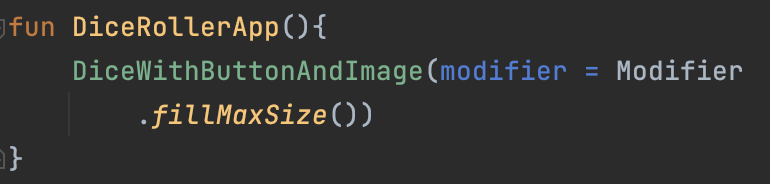
MainActivity.kt



1. Привяжите метод [fillMaxSize()](https://developer.android.com/reference/kotlin/androidx/compose/ui/Modifier#(androidx.compose.ui.Modifier).fillMaxSize(kotlin.Float))к объекту Modifier так, чтобы макет занимал весь экран.

Этот метод указывает, что компоненты должны заполнять доступное пространство. Примечательной особенностью является то, что игральные кости и кнопка расположены по центру экрана. Метод [wrapContentSize()](https://developer.android.com/reference/kotlin/androidx/compose/ui/Modifier#(androidx.compose.ui.Modifier).wrapContentSize(androidx.compose.ui.Alignment,kotlin.Boolean))указывает, что доступное пространство должно быть по крайней мере таким же большим, как и компоненты внутри него. Однако из-за использования метода fillMaxSize(), если компоненты внутри макета меньше доступного пространства, объект [Alignment](https://developer.android.com/reference/kotlin/androidx/compose/ui/Alignment) можно передать методу wrapContentSize(), который определяет, как компоненты должны выравниваться в доступном пространстве.

MainActivity.kt



**Примечание** . Операторы импорта для fillMaxSize() и wrapContentSize() - import androidx.compose.foundation.layout.fillMaxSize и

import androidx.compose.foundation.layout.wrapContentSize соответственно.

1. Привяжите метод wrapContentSize() к объекту Modifier, а затем передайте его Alignment.Center в качестве аргумента, чтобы центрировать компоненты. Alignment.Center указывает, что компонент центрируется как по вертикали, так и по горизонтали.

MainActivity.kt

****

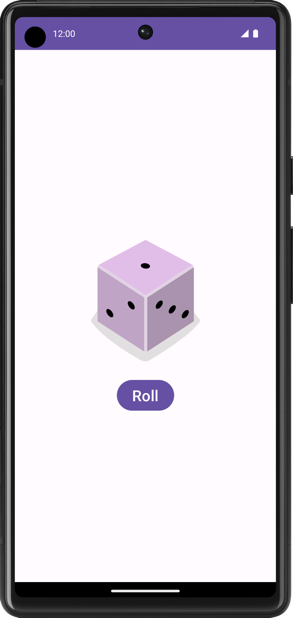
**Примечание** . Оператор импорта объекта Alignment:

import androidx.compose.ui.Alignment.

## [**4. Создайте вертикальный макет**](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app#3)

В Compose с помощью функции Column() создаются вертикальные макеты.

Функция [Column()](https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary#Column(androidx.compose.ui.Modifier,androidx.compose.foundation.layout.Arrangement.Vertical,androidx.compose.ui.Alignment.Horizontal,kotlin.Function1))представляет собой составной макет, в котором дочерние элементы располагаются в вертикальной последовательности. В ожидаемом дизайне приложения вы можете видеть, что изображение игральной кости отображается вертикально над кнопкой броска:



Чтобы создать вертикальный макет:

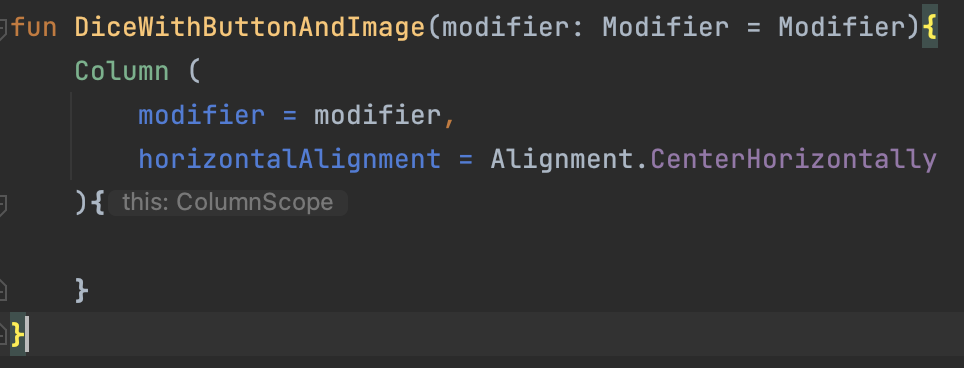
1. В функции DiceWithButtonAndImage() добавьте функцию Column().
2. Передайте аргумент modifier из сигнатуры метода DiceWithImageAndButton() в аргумент Column() модификатора .

Аргумент modifier гарантирует, что компонуемые элементы в функции Column() соответствуют ограничениям, вызванным для экземпляра modifier.

1. Передайте аргумент horizontalAlignment функции Column(), а затем установите для него значение Alignment.CenterHorizontally.

Это гарантирует, что дочерние элементы внутри столбца будут центрированы на экране устройства по ширине.

MainActivity.kt



## [**5. Добавьте кнопку**](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app#4)

1. В файл strings.xml добавьте строку и присвойте ей значение Roll.

res/values/strings.xml

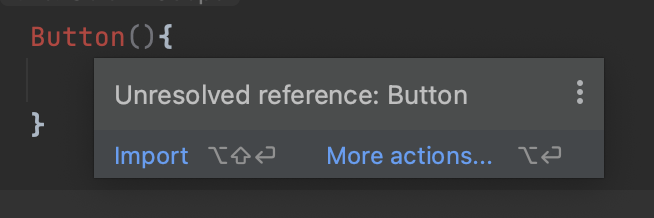
<string name="roll">Roll</string>

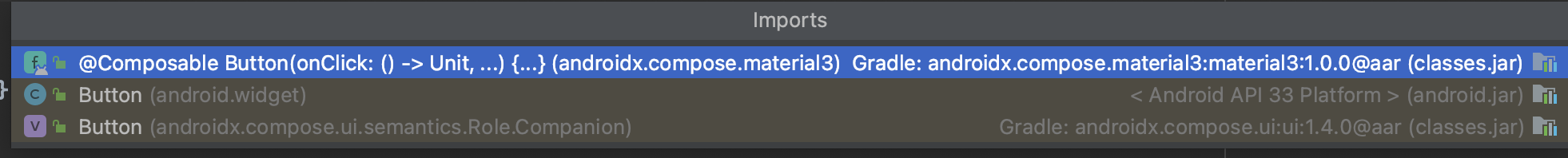
1. В  тело лямбды Column() добавьте функцию Button().

**Примечание.** Оператор импорта для составного объекта Button —

import androidx.compose.material3.Button.

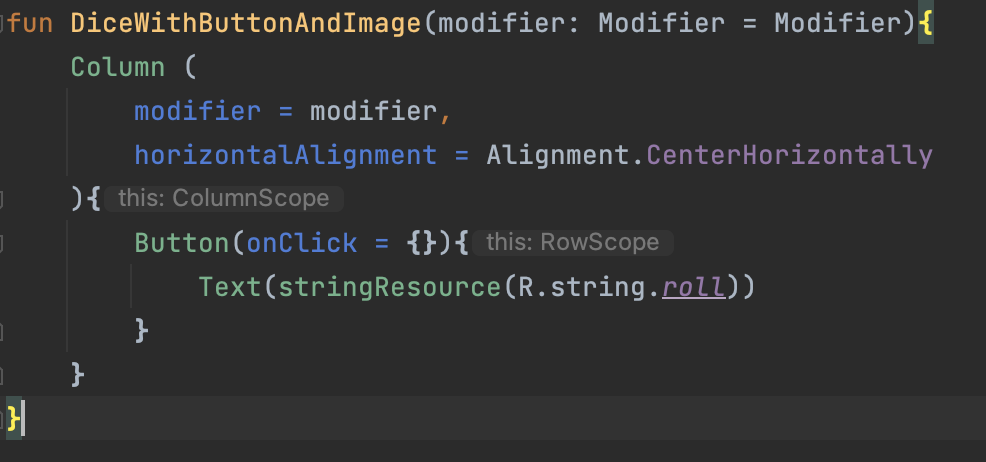
или при наведении на название появляется окно с кнопкой Import. Нажмите ее, выберите первый элемент и пакет добавится в раздел импорт автоматически.





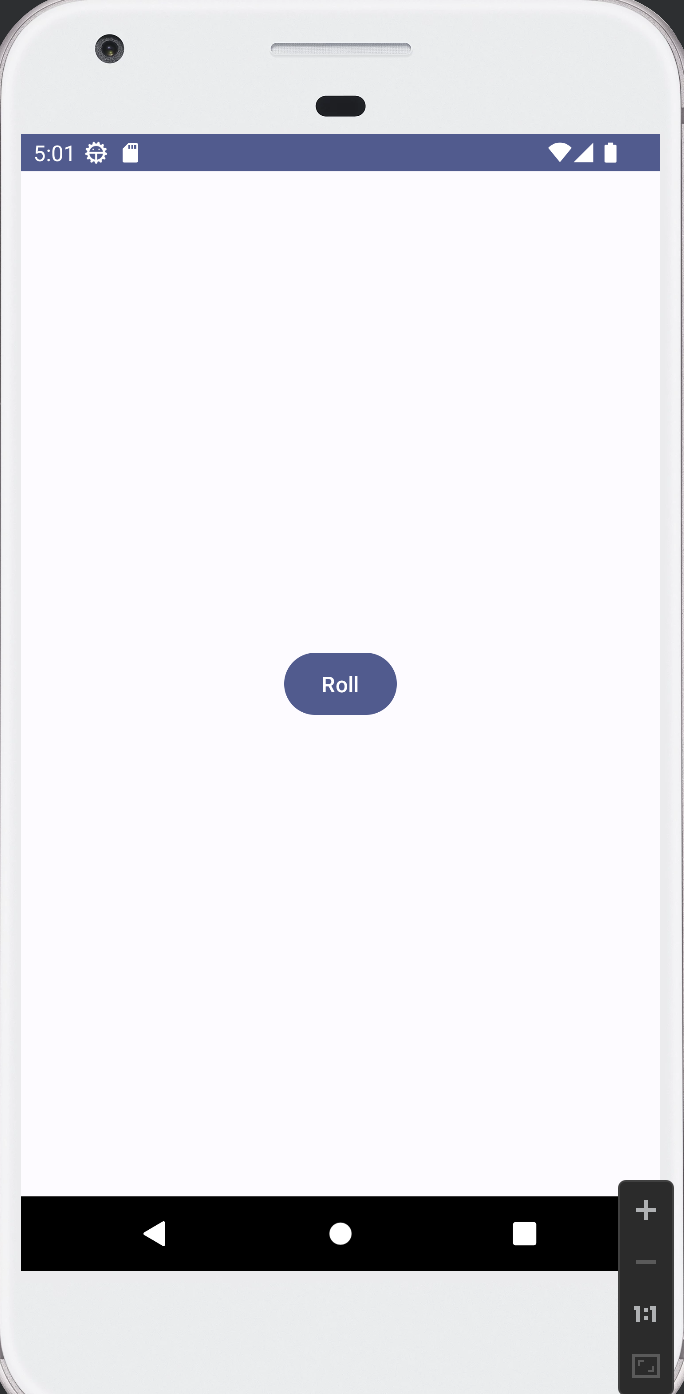
1. В  файле MainActivity.kt добавьте функцию Text() в лямбда-тело функции Button().
2. Передайте идентификатор строкового ресурса строки roll в функцию stringResource()и передайте результат в компонуемый объект Text.

MainActivity.kt



**Примечание** . Если вы выполняете автозаполнение функции Button(),аргумент onClick = { /\*TODO\*/ } появляется. Если вы не выполняете автозаполнение или Android Studio не позволяет вам это сделать, вы можете реализовать этот аргумент самостоятельно в качестве заполнителя.

При запуске вы увидите пока только кнопку. Проверьте работу в эмуляторе:



## [**6. Добавьте изображение**](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app#5)

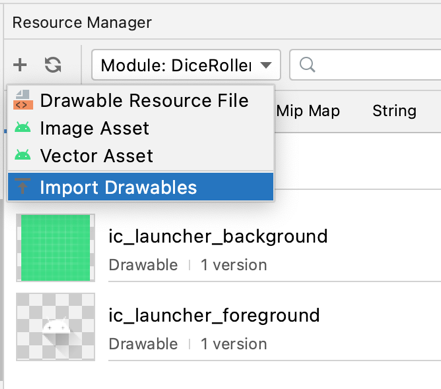
Еще одним важным компонентом приложения является изображение игральной кости, которое отображает результат, когда пользователь нажимает кнопку **«Бросить»**. Вы добавляете изображение с помощью составного элемента Image, но для этого требуется ресурс изображения, поэтому сначала вам необходимо загрузить несколько изображений, предоставленных для этого приложения.

## **Загрузите изображения кубиков**

1. Откройте [этот URL-адрес](https://github.com/google-developer-training/basic-android-kotlin-compose-training-dice-roller/raw/main/dice_images.zip), чтобы загрузить zip-файл с изображениями игральных костей на свой компьютер или найдите в сетевой папке ресурсы к лабораторной работе DiceRoller.
2. Распакуйте zip-файл, чтобы создать новую папку dice\_images, содержащую шесть файлов изображений кубиков со значениями кубиков от 1 до 6.

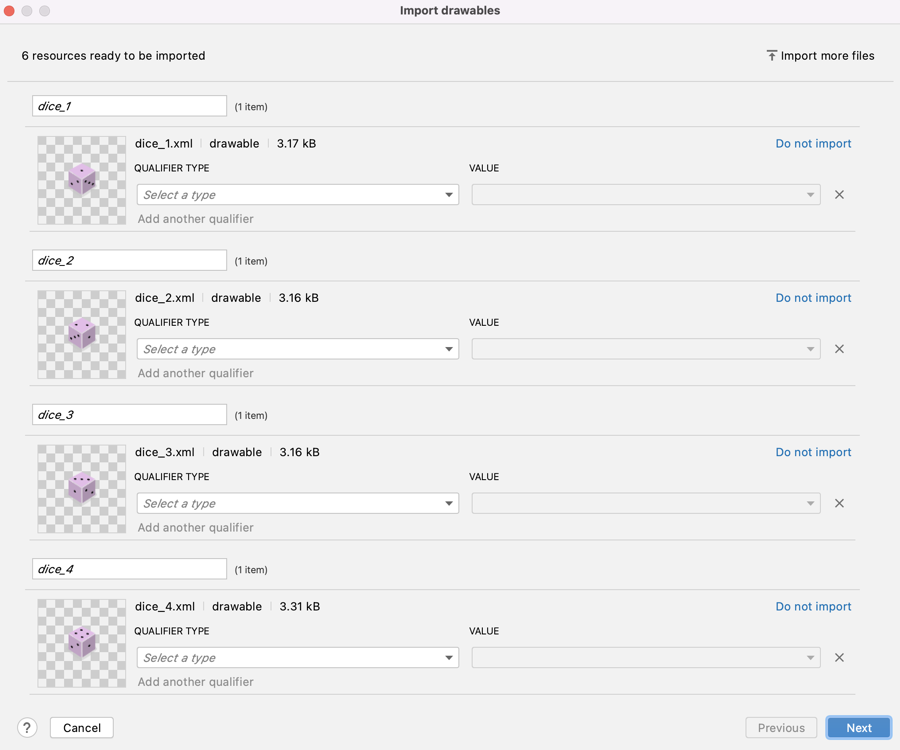
## **Добавьте изображения кубиков в свое приложение**

1. В Android Studio нажмите View > Tool Windows > Resource Manage или найдите вкладку Resource Manage слева в среде разработки (около структуры проекта).
2. Нажмите **+ >** Import drawings, чтобы открыть браузер файлов.

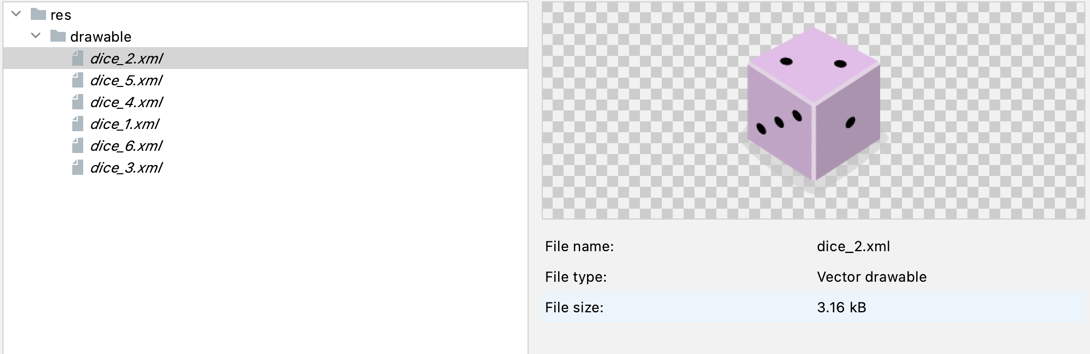


1. Найдите и выберите папку с изображениями шести кубиков и приступайте к их загрузке.

Загруженные изображения будут выглядеть следующим образом.

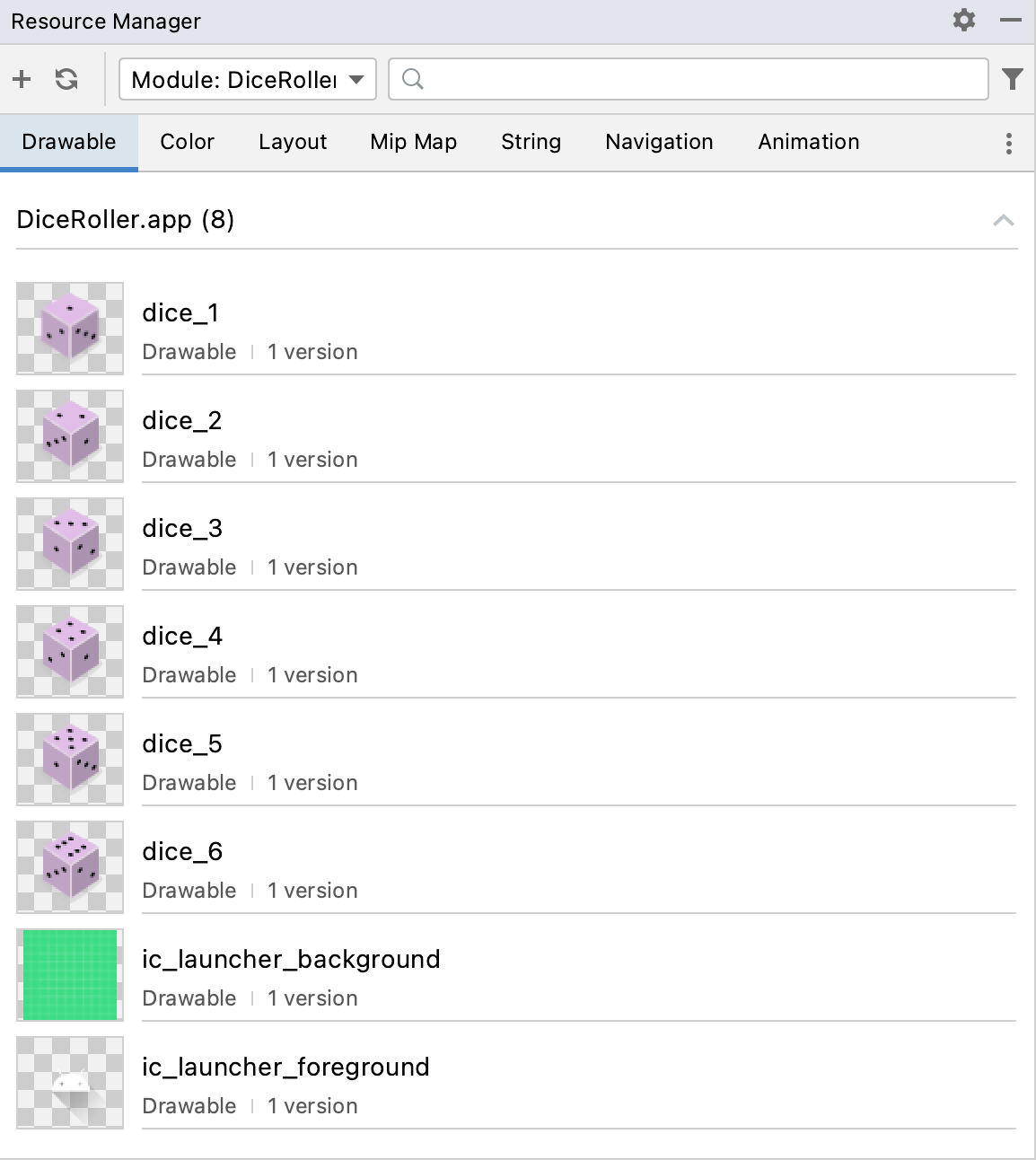


1. Нажмите **"Next** . Появится диалоговое окно **«**Importing drawings**»** , в котором будет показано, где в файловой структуре располагаются файлы ресурсов.



1. Нажмите **«**Import**»** , чтобы подтвердить, что вы хотите импортировать шесть изображений.

Изображения должны появиться на панели **«**Resource Manager**»** .



**Важный!**Вы можете ссылаться на эти изображения в своем коде Kotlin по их идентификаторам ресурсов:

* R.drawable.dice\_1
* R.drawable.dice\_2
* R.drawable.dice\_3
* R.drawable.dice\_4
* R.drawable.dice\_5
* R.drawable.dice\_6

Хорошая работа! В следующей задаче вы используете эти изображения в своем приложении.

## **Добавить  составной элемент Image**

Изображение игральной кости должно появиться над кнопкой **«Roll»**. Compose по своей сути размещает компоненты пользовательского интерфейса последовательно. Другими словами, какой бы составной объект ни был объявлен первым, он отображается первым. Это может означать, что первое объявление отображается выше или перед компонуемым объектом, объявленным после него. Составные элементы внутри составного объекта Column будут отображаться на устройстве сверху/под друг другом. В этом приложении вы используете Column для вертикального штабелирования Composables, поэтому тот Composable, который объявлен первым внутри функции Column(),  отображается перед составным объектом, объявленным после него в той же  функции Column().

Чтобы добавить составной объект Image:

1. В  теле функции Column() создайте функцию Image() перед  функцией Button().

MainActivity.kt

Column(  
    modifier = modifier,  
    horizontalAlignment = Alignment.CenterHorizontally  
) {  
**Image()**    Button(onClick = { /\*TODO\*/ }) {  
      Text(stringResource(R.string.roll))  
    }  
}

**Примечание.** Оператор импорта для составного объекта Image—

import androidx.compose.foundation.Image.

1. Передайте  функции Image() аргумент painter, а затем присвойте ей значение painterResource, которое принимает аргумент идентификатора ресурса. На данный момент передайте следующий идентификатор ресурса: аргумент R.drawable.dice\_1.

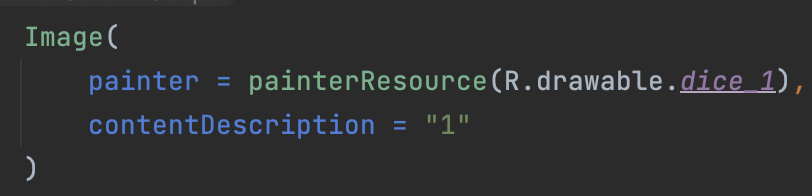
MainActivity.kt

Image(  
**painter = painterResource(R.drawable.dice\_1)**)

**Примечание** . Позже вы измените значение, переданное для идентификатора ресурса. На данный момент ему следует передать значение по умолчанию, чтобы код компилировался для целей предварительного просмотра.

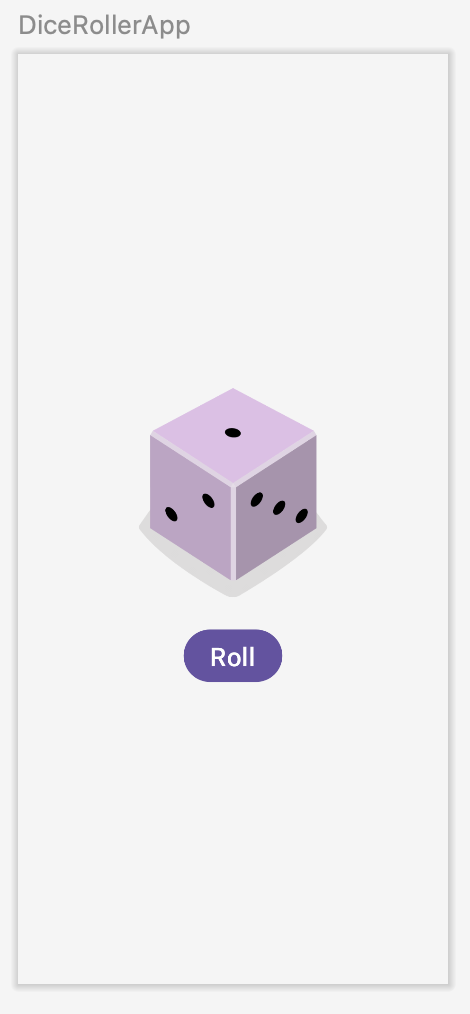
1. Каждый раз, когда вы создаете изображение в своем приложении, вы должны предоставить так называемое «описание контента». Описания контента — важная часть разработки Android. Они прикрепляют описания к соответствующим компонентам пользовательского интерфейса, чтобы повысить доступность.  Вы можете передать описание содержимого изображению в качестве параметра.

MainActivity.kt



**Примечание.** Приведенное выше описание контента на данный момент является заполнителем. Это будет обновлено в следующем разделе этой лаборатории кода.

Теперь все необходимые компоненты пользовательского интерфейса присутствуют. Но они Button и Imageнемного теснят друг друга.



1. Чтобы это исправить, добавьте составной объект [Spacer](https://developer.android.com/reference/kotlin/androidx/compose/foundation/layout/package-summary#Spacer(androidx.compose.ui.Modifier)) между составными объектами Button и Image. A принимает Spacer в качестве параметра Modifier. В данном случае значок Image находится над Button, поэтому между ними должно быть вертикальное пространство. Таким образом, высоту Modifier можно установить так, чтобы она применялась к Spacer. Попробуйте установить высоту 16.dp. Обычно размеры dp изменяются с шагом 4.dp.

MainActivity.kt

Spacer(modifier = Modifier.height(16.dp))

**Примечание.** Импорт для составного элемента Spacer, модификатора height и dp:

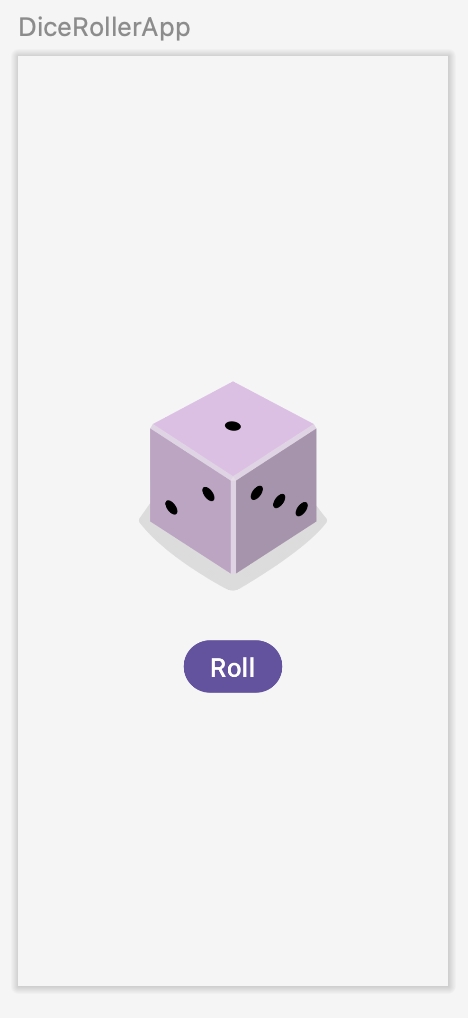
import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.Spacer

import androidx.compose.ui.unit.dp

1. На панели **Preview** нажмите **«Build & Refresh»** .

Вы должны увидеть что-то похожее на это изображение:



## [**7. Постройте логику броска кубиков**](https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-build-a-dice-roller-app#6)

Теперь, когда все необходимые компоненты присутствуют, вы модифицируете приложение так, чтобы нажатие кнопки бросало кости.

## **Сделайте кнопку интерактивной**

1. В функции DiceWithButtonAndImage() перед функцией Column() создайте переменную result и присвойте ей значение 1.
2. Взгляните на композицию Button. Вы заметите, что ему передается параметр onClick, который имеет форму пары фигурных скобок с комментарием /\*TODO\*/ внутри скобок. В данном случае фигурные скобки представляют собой так называемую лямбду, причем область внутри фигурных скобок является телом лямбды. Когда функция передается в качестве аргумента, ее также можно назвать « [обратным вызовом](https://en.wikipedia.org/wiki/Callback_(computer_programming)) ».

MainActivity.kt

Button(onClick = { /\*TODO\*/ })

Лямбда — это функциональный литерал, который является такой же функцией, как и любая другая, но вместо того, чтобы объявляться отдельно с ключевым словом fun, он записывается в строке и передается как выражение. Составной объект Button ожидает, что функция будет передана в качестве параметра onClick. Это идеальное место для использования лямбды, и в этом разделе вы будете писать тело лямбды.

1. В Button() удалите комментарий /\*TODO\*/ из значения лямбда-тела параметра onClick.
2. Бросок кубика случаен. Чтобы отразить это в коде, вам нужно использовать правильный синтаксис для генерации случайного числа. В Котлине вы можете использовать метод random() для диапазона чисел. В теле лямбды onClick задайте для переменной result диапазон от 1 до 6, а затем вызовите метод random()в этом диапазоне. Помните, что в Котлине диапазоны обозначаются двумя точками между первым числом диапазона и последним числом диапазона.

MainActivity.kt

fun DiceWithButtonAndImage(modifier: Modifier = Modifier) {  
    var result = 1  
    Column(  
        modifier = modifier,  
        horizontalAlignment = Alignment.CenterHorizontally  
    ) {  
        Image(painter = painterResource(imageResource), contentDescription = result.toString())  
        Button(onClick = { **result = (1..6).random()** }) {  
            Text(stringResource(R.string.roll))  
        }  
    }  
}

Теперь кнопку можно нажимать, но нажатие кнопки пока не приведет к каким-либо визуальным изменениям, поскольку вам еще нужно создать эту функциональность.

### **Добавьте условие в приложение для игры в кости**

Вы создали переменную result и жестко запрограммировали ее значение 1. В конечном итоге значение переменной result сбрасывается при нажатии кнопки **«Roll» , и оно должно определять, какое изображение будет показано.**

По умолчанию составные объекты не имеют состояния, что означает, что они не содержат значения и могут быть перекомпонованы системой в любое время, что приводит к сбросу значения. Однако Compose предоставляет удобный способ избежать этого. Составные функции могут хранить объект в памяти с помощью composable функции remember.

1. Сделайте переменную result составной remember.

Компонуемый remember требует передачи функции.

1. В  составном теле remember передайте функцию mutableStateOf(), а затем передайте ей аргумент 1.

Функция mutableStateOf() возвращает наблюдаемую. Вы узнаете больше о наблюдаемых позже, но сейчас это в основном означает, что при изменении значения переменной result запускается рекомпозиция, значение результата отражается и пользовательский интерфейс обновляется.

MainActivity.kt

var result by remember { mutableStateOf(1) }

**Примечание** . Операторы

import androidx.compose.runtime.mutableStateOf

import androidx.compose.runtime.remember

импортируют пакеты, необходимые для функции mutableStateOf() и компонуемого объекта remember.

Следующие операторы импорта также необходимы для импорта необходимых функций расширения State:

import androidx.compose.runtime.getValue

import androidx.compose.runtime.setValue

Теперь при нажатии кнопки переменная result обновляется значением случайного числа.

Теперь переменную result можно использовать для определения того, какое изображение показывать.

1. Под созданием переменной result создайте неизменяемую переменную imageResource, заданную для выражения when, которое принимает переменную result, а затем установите каждый возможный результат в его рисуемую форму.

MainActivity.kt

val imageResource = when (result) {  
    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  
}

1. Измените идентификатор, переданный в параметр painterResource компонуемого объекта Image, с рисуемого R.drawable.dice\_1 на переменную imageResource.
2. Измените  параметр contentDescription компонуемого объекта Image, чтобы он отражал значение переменной result, преобразовав переменную result в строку с помощью toString() и передав ее как contentDescription.

MainActivity.kt

Image(painter = painterResource(id = imageResource), contentDescription = result.toString())

1. Запустите свое приложение.

Ваше приложение **Dice Roller** теперь должно полностью работать!

