# 1.Wprowadzenie

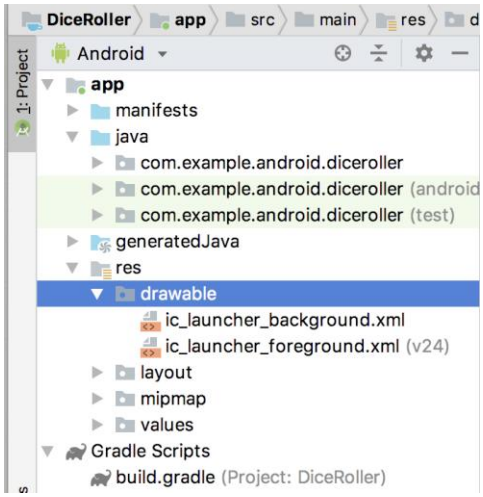
W tym ćwiczeniu poprawiasz aplikację DiceRoller i dowiesz się, jak dodawać i wykorzystywać zasoby graficzne w swojej aplikacji. Dowiesz się także o kompatybilności aplikacji z różnymi wersjami Androida i o tym, jak może pomóc Android Jetpack.

# 3. Zadanie: dodaj i zaktualizuj zasoby obrazu

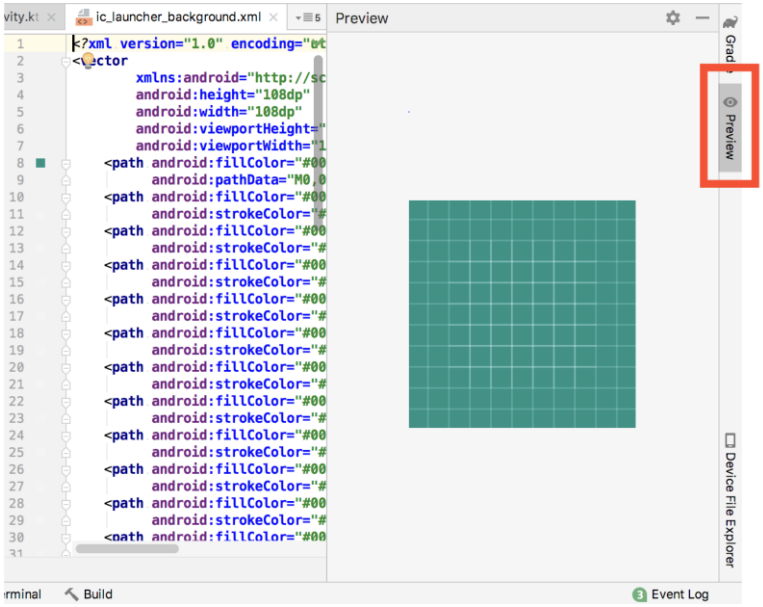
Pod koniec ostatniego kodu masz aplikację, która aktualizuje widok tekstu z liczbą od 1 do 6 za każdym razem, gdy użytkownik stuknie przycisk. Jednak aplikacja nazywa się DiceRoller, a nie generator liczb 1-6, więc byłoby fajnie, gdyby kości faktycznie wyglądały jak kości. W tym zadaniu dodajesz do swojej aplikacji kilka obrazów kości. Następnie zamiast aktualizować tekst po naciśnięciu przycisku, zamieniasz obraz dla każdego wyniku rzutu.

# Step 1: Dodawanie obrazów

1. W widoku Projekt> Android rozwiń folder res, a następnie rozwiń **drawable**.



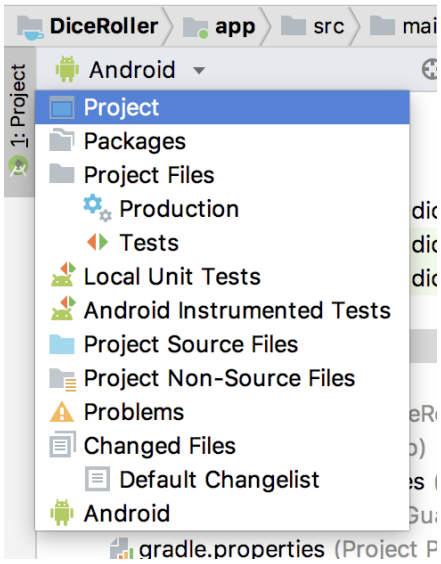

   Twoja aplikacja korzysta z wielu różnych zasobów, w tym: images, icons, colors, strings, and XML layouts. Wszystkie te zasoby są przechowywane w folderze `res`. Folder `drawable` jest miejscem gdzie powinny być umieszczone wszystkie zasoby obrazów . W tym momencie w folderze `drawable` można znaleźć zasoby dla ikon programu uruchamiającego aplikację.
2. Kliknij dwukrotnie ic_launcher_background.xml. Pamiętaj, że są to pliki XML opisujące ikonę jako obraz wektorowy. Wektory umożliwiają rysowanie obrazów w wielu różnych rozmiarach i rozdzielczościach. Obrazy bitmapowe, takie jak PNG lub GIF, mogą wymagać skalowania dla różnych urządzeń, co może spowodować pewną utratę jakości..
3. Kliknij opcję **Preview** (Podgląd) w prawej kolumnie edytora XML, aby wyświetlić wektor do rysowania w formie wizualnej.




4. Pobierz obrazy kości dla swojej aplikacji. Rozpakuj archiwum. Powinieneś mieć folder plików XML, który wygląda następująco:

| Name | Date Modified | Size | Kind |
|---|---|---|---|
| dice_1.xml | Aug 30, 2018 at 6:21 PM | 3 KB | XML Document |
| dice_2.xml | Aug 30, 2018 at 6:21 PM | 3 KB | XML Document |
| dice_3.xml | Aug 30, 2018 at 6:21 PM | 3 KB | XML Document |
| dice_4.xml | Aug 30, 2018 at 6:21 PM | 3 KB | XML Document |
| dice_5.xml | Aug 30, 2018 at 6:21 PM | 4 KB | XML Document |
| dice_6.xml | Aug 30, 2018 at 6:21 PM | 4 KB | XML Document |
| empty_dice.xml | Aug 30, 2018 at 6:21 PM | 921 bytes | XML Document |

6. W Android Studio kliknij menu rozwijane u góry widoku projektu, które obecnie pokazuje **Android**, i wybierz **Project**. Poniższy zrzut ekranu pokazuje, jak wygląda struktura Twojej aplikacji w systemie plików..

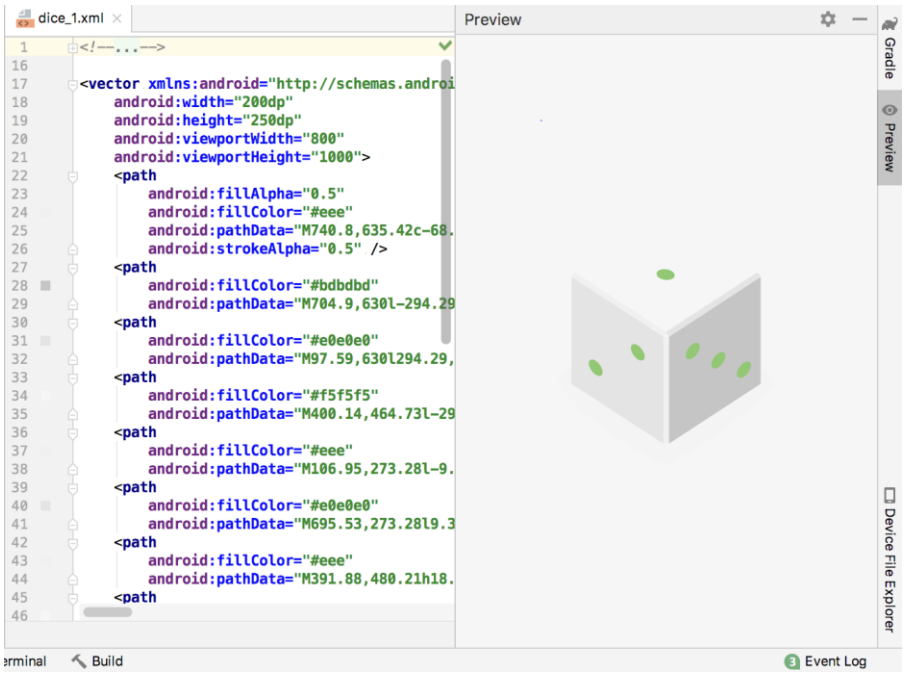



7. Rozwiń **DiceRoller** > **app** > **src** > **main** > **res** > **drawable**.

8. Przeciągnij wszystkie pojedyncze pliki XML z folderu DiceImages do Android Studio i do folderu **drawable**. Kliknij **OK**.

Uwaga: Upewnij się, że upuściłeś pliki w folderze **drawable**, a nie w folderze **drawable24**. Nie dołączaj również samego folderu DiceImages. Przeciągnij tylko pliki XML.

9. Przełącz projekt z powrotem do widoku Androida i zauważ, że pliki XML obrazu kostki znajdują się w folderze  drawable.
10. Kliknij dwukrotnie plik dice_1.xml i zwróć uwagę na kod XML tego obrazu. Kliknij przycisk Podgląd, aby uzyskać podgląd tego, jak faktycznie wygląda ten rysowany wektor.




# Krok 2: Zaktualizuj układ, aby używać obrazów

Teraz, gdy masz pliki obrazów kości w folderze res / drawables, możesz uzyskać dostęp do tych plików z układu aplikacji i kodu. W tym kroku zamieniasz TextView, który wyświetla liczby, na ImageView, aby wyświetlić obrazy..

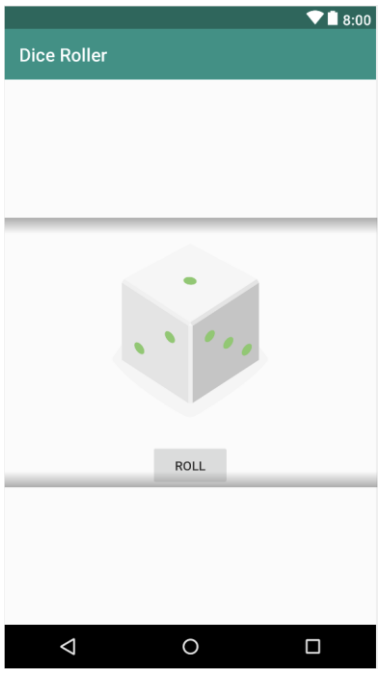
1. Otwórz plik `activity_main.xml` layout.  Kliknij zakładkę **Text** aby zwyswietlić kod XML.
2. Usuń element `<TextView>` .
3. Dodaj element `<ImageView>` z następującymi atrybutami:

```
<ImageView
    android:id="@+id/dice_image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:src="@drawable/dice_1" />
```

Za pomocą ImageView możesz wyświetlić obraz w swoim układzie. Jedynym nowym atrybutem tego elementu jest android: src, który wskazuje zasób źródłowy obrazu. W takim

przypadku źródło obrazu `@drawable/dice_1` oznacza, że Android powinien poszukać w zasobach do rysowania (`res/drawable`) obrazu o nazwie `dice_1`.

4. Kliknij przycisk **Preview**, aby wyświetlić podgląd układu. To powinno wyglądać tak::




# Krok 3: Zaktualizuj kod

1. Otwórz `MainActivity`. Oto jak do tej pory wygląda funkcja `rollDice()`:

```
private fun rollDice() {
   val randomInt = Random().nextInt(6) + 1

   val resultText: TextView = findViewById(R.id.result_text)
   resultText.text = randomInt.toString()
}
```

Zauważ, że odniesienie do R.id.result_text może być podświetlone na czerwono - to dlatego, że usunąłeś TextView z układu i ten identyfikator już nie istnieje.

2. Usuń dwa wiersze na końcu funkcji, które definiują zmienną resultText i ustaw jej właściwość text. Nie używasz już TextView w układzie, więc nie potrzebujesz żadnej linii.
3. Użyj findViewByID (), aby uzyskać odniesienie do nowego ImageView w układzie według ID (R.id.dice_image), i przypisz ten widok do nowej zmiennej diceImage:

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

4. Dodaj blok `when`, aby wybrać konkretny obraz matrycy na podstawie wartości `randomInteger`:

```
val drawableResource = when (randomInt) {
   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
}
```

Podobnie jak w przypadku identyfikatorów, możesz odwoływać się do obrazów kości w folderze drawable, z wartościami w klasie R. Tutaj R.drawable odnosi się do folderu aplikacji, drawable, a dice_1 jest konkretnym zasobem obrazu matrycy w tym folderze.

5. Zaktualizuj źródło ImageView za pomocą metody setImageResource () i odwołania do właśnie znalezionego obrazu matrycy.

```
diceImage.setImageResource(drawableResource)
```

6. Skompiluj i uruchom aplikację. Teraz po kliknięciu przycisku Roll obraz powinien zostać zaktualizowany o odpowiedni obraz.

# 4. Zadanie: wydajne wyszukiwanie widoków

Wszystko w Twojej aplikacji działa, ale tworzenie aplikacji to coś więcej niż tylko kod, który działa. Powinieneś także zrozumieć, jak pisać wydajne i dobrze zachowujące się aplikacje. Oznacza to, że aplikacje powinny działać poprawnie, nawet jeśli użytkownik nie ma najdroższego urządzenia z Androidem ani najlepszej łączności sieciowej. Twoje aplikacje powinny również nadal działać płynnie, gdy dodajesz więcej funkcji, a kod powinien być czytelny i dobrze zorganizowany.

W tym zadaniu poznasz jeden ze sposobów na zwiększenie wydajności aplikacji.

1. Otwórz MainActivity, jeśli nie jest jeszcze otwarta. W metodzie rollDice () zwróć uwagę na deklarację zmiennej diceImage:

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

Ponieważ rollDice () to moduł obsługi kliknięcia przycisku Roll, za każdym razem, gdy użytkownik stuknie ten przycisk, aplikacja wywołuje funkcję findViewById () i otrzymuje kolejne odwołanie do tego ImageView. Najlepiej byłoby zminimalizować liczbę wywołań funkcji findViewById (), ponieważ system Android przeszukuje całą hierarchię widoków za każdym razem, a to jest kosztowna operacja.

W małej aplikacji takiej jak ta nie jest to ogromny problem. Jeśli używasz bardziej skomplikowanej aplikacji na wolniejszym telefonie, ciągłe wywoływanie funkcji findViewById () może powodować opóźnienie aplikacji. Zamiast tego najlepszym rozwiązaniem jest jednorazowe wywołanie findViewById () i zapisanie obiektu View w polu. Zachowanie odwołania do ImageView w polu pozwala systemowi na bezpośredni dostęp do Widoku w dowolnym momencie, co poprawia wydajność.

2. U góry klasy, przed onCreate (), utwórz pole do przechowywania ImageView.

```
var diceImage : ImageView? = null
```

Idealnie byłoby zainicjować tę zmienną tutaj, kiedy jest zadeklarowana, lub w konstruktorze - ale działania Androida nie używają konstruktorów. W rzeczywistości widoki w układzie w ogóle nie są dostępnymi obiektami w pamięci, dopóki nie zostaną nadmuchane w metodzie onCreate () przez wywołanie setContentView (). Nie możesz w ogóle zainicjować zmiennej diceImage, dopóki to się nie stanie.

Jedną z opcji jest zdefiniowanie zmiennej diceImage jako wartości zerowej, jak w tym przykładzie. Ustaw go na null, gdy zostanie zadeklarowany, a następnie przypisz go do rzeczywistego ImageView w onCreate () za pomocą findViewById (). Skomplikuje to jednak twój kod, ponieważ teraz musisz sprawdzać wartość zerową za każdym razem, gdy chcesz użyć diceImage. Jest lepszy sposób..

3. Zmień deklarację diceImage, aby użyć słowa kluczowego lateinit, i usuń przypisanie zerowe:

```
lateinit var diceImage : ImageView
```

Słowo kluczowe lateinit obiecuje kompilatorowi Kotlin, że zmienna zostanie zainicjowana, zanim kod wywoła jakiekolwiek operacje na niej. Dlatego nie musimy tutaj inicjować zmiennej do wartości null i możemy ją traktować jako zmienną, która nie ma wartości null, kiedy jej używamy. Najlepszą praktyką jest używanie lateinit z polami, które przechowują widoki w ten właśnie sposób.

4. W onCreate (), po metodzie setContentView (), użyj findViewById (), aby uzyskać ImageView..

```
diceImage = findViewById(R.id.dice_image)
```

5. Usuń starą linię w rollDice (), która deklaruje i pobiera ImageView. Ten wiersz zastąpiłeś wcześniej deklaracją pola.

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

6. Uruchom aplikację ponownie, aby zobaczyć, że nadal działa zgodnie z oczekiwaniami.

# 5. Zadanie: użyj domyślnego obrazu

W tej chwili używasz kości_1 jako początkowego obrazu kości. Powiedzmy, że nie chciałeś wyświetlać żadnego obrazu, dopóki kostka nie zostanie rzucona po raz pierwszy. Istnieje kilka sposobów na osiągnięcie tego.

1. Otwórz Activity_layout.xml w zakładce Tekst.
2. W elemencie `<ImageView>` ustaw atrybut `android:src` na: `"@drawable/empty_dice"`:

```
android:src="@drawable/empty_dice"
```

Obraz empty_dice był jednym z obrazów pobranych i dodanych do folderu, który można wyciągać. Ma taki sam rozmiar jak inne obrazy kości, tyle że jest pusty. Ten obraz zostanie wyświetlony przy pierwszym uruchomieniu aplikacji.

3. Kliknij kartę **Design**. Obraz matrycy jest teraz pusty, ale nie jest również widoczny w podglądzie.


   Dość często treść projektu może być definiowana dynamicznie w czasie wykonywania - na przykład każda aplikacja, która pobiera dane z Internetu, powinna prawdopodobnie zaczynać się pustym lub pustym ekranem. Jest to jednak przydatne, gdy projektujesz aplikację tak, aby zawierała w układzie jakieś dane zastępcze, abyś wiedział, co planujesz.
4. W activity_layout.xml skopiuj linię android: src i wklej drugą kopię. Zmień słowo „android” na „narzędzia”, aby Twoje dwa atrybuty wyglądały tak:

```
android:src="@drawable/empty_dice"
tools:src="@drawable/empty_dice" />
```

Tutaj zmieniłeś przestrzeń nazw XML tego atrybutu z domyślnej przestrzeni nazw Androida na przestrzeń nazw narzędzi. Przestrzeń nazw narzędzi jest używana, gdy chcesz zdefiniować symbol zastępczy, który jest używany tylko w podglądzie lub edytorze projektu w Android Studio. Atrybuty korzystające z przestrzeni nazw narzędzi są usuwane podczas kompilowania aplikacji.

Przestrzenie nazw służą do rozwiązywania niejednoznaczności w odniesieniu do atrybutów o tej samej nazwie. Na przykład oba te atrybuty w znaczniku <ImageView> mają tę samą nazwę (src), ale przestrzeń nazw jest inna..

5. Sprawdź element <LinearLayout> w katalogu głównym pliku układu i zwróć uwagę na dwie zdefiniowane tutaj przestrzenie nazw.

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   ...
```

6. Zmień atrybut `tools:src` w tagu `ImageView` na `dice_1` zamiast `empty_dice`:

```
android:src="@drawable/empty_dice"
tools:src="@drawable/dice_1" />
```

Zauważ, że obraz dice_1 jest teraz na miejscu jako obraz zastępczy w podglądzie.

7. Skompiluj i uruchom aplikację. Zauważ, że obraz matrycy jest pusty w rzeczywistej aplikacji, dopóki nie klikniesz lub nie stukniesz opcji Roll.

# 6. Zadanie: poznanie poziomów API i kompatybilności

Jedną z wielkich zalet programowania na Androida jest sama liczba urządzeń, na których kod może działać - od Nexusa One do Pixela, poprzez takie urządzenia jak tablety, Pixelbooki, zegarki, telewizory i samochody.

Pisząc dla systemu Android, nie piszesz całkowicie osobnych aplikacji dla każdego z tych różnych urządzeń - nawet aplikacje działające na radykalnie różnych urządzeniach, takich jak zegarki i telewizory, mogą współdzielić kod. Ale nadal istnieją ograniczenia i strategie zgodności, o których należy pamiętać, aby wspierać to wszystko.

W tym zadaniu nauczysz się, jak kierować aplikację na określone poziomy (wersje) interfejsu API Androida, i jak używać bibliotek Android Jetpack do obsługi starszych urządzeń.

# Krok 1: poznaj poziomy API

W poprzednim codelabie podczas tworzenia projektu wskazano konkretny poziom interfejsu API systemu Android, który powinna obsługiwać Twoja aplikacja. System operacyjny Android ma różne numery wersji, nazwane od smacznych smakołyków, które są uporządkowane alfabetycznie. Każda wersja systemu operacyjnego jest dostarczana z nowymi funkcjami. Na przykład Android Oreo jest dostarczany z obsługą aplikacji Picture-in-picture apps, a Android Pie wprowadza introduced Slices. The API levels correspond to the Android versions. Poziomy API odpowiadają wersjom Androida. Na przykład API 19 odpowiada Androidowi 4.4 (KitKat).

Ze względu na szereg czynników, w tym na obsługiwany sprzęt, to, czy użytkownicy zdecydują się na aktualizację swoich urządzeń oraz czy producenci obsługują różne poziomy systemu operacyjnego, użytkownicy nieuchronnie kończą na urządzeniach z różnymi wersjami systemu operacyjnego..

Podczas tworzenia projektu aplikacji określasz minimalny poziom interfejsu API obsługiwany przez aplikację. Oznacza to, że określasz najstarszą wersję Androida obsługiwaną przez aplikację. Twoja aplikacja ma również poziom, do którego jest skompilowana, i poziom, na który jest kierowana. Każdy z tych poziomów jest parametrem konfiguracyjnym w plikach kompilacji Gradle.

1. Rozwiń folder **Gradle Scripts** i otwórz plik **build.gradle (Module: app)**.

Ten plik określa parametry kompilacji i zależności specyficzne dla modułu aplikacji. Plik build.gradle (Project: DiceRoller) definiuje parametry kompilacji dla całego projektu. W wielu przypadkach moduł aplikacji jest jedynym modułem w projekcie, więc podział ten może wydawać się arbitralny. Ale jeśli Twoja aplikacja stanie się bardziej złożona i podzielisz ją na kilka części lub jeśli aplikacja obsługuje platformy takie jak Android Watch, możesz napotkać różne moduły w tym samym projekcie.

2. Sprawdź sekcję Androida w górnej części pliku build.gradle. (Poniższy przykład nie jest całą sekcją, ale zawiera to, co najbardziej Cię interesuje dla tego kodu).

```
android {
   compileSdkVersion 28
   defaultConfig {
       applicationId "com.example.android.diceroller"
       minSdkVersion 19
       targetSdkVersion 28
       versionCode 1
       versionName "1.0"
   }
```

3. Sprawdź parametr `compileSdkVersion` .

```
compileSdkVersion 28
```

Ten parametr określa poziom interfejsu API Androida, którego Gradle powinien użyć do skompilowania aplikacji. To jest najnowsza wersja Androida, którą Twoja aplikacja może obsługiwać. Oznacza to, że Twoja aplikacja może korzystać z funkcji API zawartych na tym poziomie API i niższych. W takim przypadku aplikacja obsługuje interfejs API 28, który odpowiada systemowi Android 9 (Pie).

4. Sprawdź parametr `targetSdkVersion` który znajduje się w sekcji `defaultConfig`:

```
targetSdkVersion 28
```

Ta wartość jest najnowszym interfejsem API, na którym przetestowano aplikację. W wielu przypadkach jest to ta sama wartość, co `compileSdkVersion`.

5. Sprawdź parametr `minSdkVersion`.

```
minSdkVersion 19
```

Ten parametr jest najważniejszy z trzech, ponieważ określa najstarszą wersję Androida, na której będzie działać Twoja aplikacja. Urządzenia z systemem operacyjnym Android starszym niż ten poziom API nie mogą w ogóle uruchamiać Twojej aplikacji.

Wybór minimalnego poziomu interfejsu API dla Twojej aplikacji może być trudny. Ustawiony zbyt niski poziom interfejsu API, ograniczy nowsze funkcje systemu operacyjnego Android. Ustawiony za wysoko, sprawi ze aplikacja może działać tylko na nowszych urządzeniach.

Po skonfigurowaniu projektu i przejściu do miejsca, w którym określasz minimalny poziom interfejsu API swojej aplikacji, kliknij opcję **Help me chooseaby wyswietlić** okno dialogowe **API Version Distribution**. Okno dialogowe zawiera informacje o tym, ile urządzeń używa

różnych poziomów systemu operacyjnego oraz funkcje, które zostały dodane lub zmienione na poziomach systemu operacyjnego. Możesz także sprawdzić informacje o wersji dokumentacji i pulpit nawigacyjny dla systemu Android, które zawierają dodatkowe informacje na temat konsekwencji obsługi różnych poziomów API..

# Krok 2: Sprawdź zgodność

Pisanie dla różnych poziomów API Androida jest powszechnym wyzwaniem, przed którym stają twórcy aplikacji, więc zespół platformy Android wykonał wiele pracy, aby Ci pomóc.

W 2011 roku zespół wydał pierwszą bibliotekę wsparcia, bibliotekę opracowaną przez Google, która oferuje klasy kompatybilne wstecz i pomocne funkcje. W 2018 roku Google ogłosił Android Jetpack, który jest zbiorem bibliotek, który zawiera wiele poprzednich klas i funkcji biblioteki wsparcia, a także rozszerza bibliotekę wsparcia.

1. Otwórz. `MainActivity`.
2. Zauważ, że twoja klasa MainActivity nie pochodzi z samego działania, ale z AppCompatActivity..

```
class MainActivity : AppCompatActivity() {
...
```

`AppCompatActivity` to klasa zgodności, która zapewnia, że twoja aktywność wygląda tak samo na różnych poziomach systemu operacyjnego..

3. Kliknij symbol + obok linii rozpoczynającej się od importu, aby rozwinąć import dla swojej klasy. Pamiętaj, że klasa AppCompatActivity jest importowana z pakietu androidx.appcompat.app. Przestrzeń nazw dla bibliotek Jetpack dla Androida to Androidx.
4. Otwórz **build.gradle (Module: app)** i przewiń w dół do sekcji zależności.

```
dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   implementation"org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
   implementation 'androidx.appcompat:appcompat:1.0.0-beta01'
   implementation 'androidx.core:core-ktx:1.0.1'
   implementation 'androidx.constraintlayout:constraintlayout:1.1.2'
   testImplementation 'junit:junit:4.12'
   androidTestImplementation 'androidx.test:runner:1.1.0-alpha4'
   androidTestImplementation
        'androidx.test.espresso:espresso-core:3.1.0-alpha4'
}
```

Zwróć uwagę na zależność od biblioteki appcompat, która jest częścią systemu Androidx i zawiera klasę AppCompatActivity.

**Tip:** Ogólnie rzecz biorąc, jeśli aplikacja może korzystać z klasy zgodności z bibliotek Jetpack, powinna używać jednej z tych klas, ponieważ klasy te zapewniają obsługę możliwie największej liczby funkcji i urządzeń.

# Krok 3: Dodaj zgodność dla wektorowych elementów rysunkowych (drawables)

Wykorzystasz swoją nową wiedzę o przestrzeniach nazw, stopniowaniu i kompatybilności, aby dokonać ostatecznej korekty aplikacji, która zoptymalizuje rozmiar aplikacji na starszych platformach.

1. Rozwiń folder res, a następnie rozwiń **drawable**. Kliknij dwukrotnie jeden z obrazów kostki.

   Jak się dowiedziałeś wcześniej, wszystkie obrazy kości są w rzeczywistości plikami XML, które określają kolory i kształty kości. Tego rodzaju pliki nazywane są wektorowymi drawables. Zaletą rysunków wektorowych w porównaniu do formatów bitmapowych, takich jak PNG, jest to, że rysunki wektorowe można skalować bez utraty jakości. Ponadto plik do rysowania wektorowego jest zwykle znacznie mniejszym plikiem niż ten sam obraz w formacie bitmapy.

   Ważną rzeczą, na którą należy zwrócić uwagę w odniesieniu do wektorowych drawable, jest to, że są one obsługiwane w API 21 i późniejszych. Ale minimalny zestaw SDK aplikacji jest ustawiony na API 19. Jeśli wypróbowałeś aplikację na urządzeniu lub emulatorze API 19, zobaczysz, że aplikacja wydaje się działać poprawnie. Jak to działa?

   Podczas budowania aplikacji proces kompilacji Gradle generuje plik PNG z każdego z plików wektorowych, a te pliki PNG są używane na dowolnym urządzeniu z Androidem poniżej 21. Te dodatkowe pliki PNG zwiększają rozmiar aplikacji. Niepotrzebnie duże aplikacje nie są świetne - spowalniają pobieranie plików przez użytkowników i zajmują więcej miejsca na urządzeniach. Duże aplikacje mają również większą szansę na odinstalowanie, a użytkownicy nie pobiorą lub nie anulują pobierania tych aplikacji.

   Dobrą wiadomością jest to, że istnieje biblioteka kompatybilności z Androidem X dla wektorowych drawable aż do poziomu API 7.

2. Otwórz **build.gradle (Module: app)**. Dodaj ten wiersz do sekcji `defaultConfig`:

```
vectorDrawables.useSupportLibrary = true
```

3. Kliknij przycisk **Sync Now**. Za każdym razem, gdy modyfikowany jest plik build.gradle, musisz zsynchronizować pliki build z projektem.
4. Otwórz plik układu main_activity.xml. Dodaj tę przestrzeń nazw do znacznika root <LinearLayout> poniżej przestrzeni nazw narzędzi::

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

Przestrzeń nazw app dotyczy atrybutów pochodzących z niestandardowego kodu lub z bibliotek, a nie z podstawowej struktury systemu Android..

5. Zmień atrybut `android:src` w elemencie `<ImageView>` na `app:srcCompat`.

```
app:srcCompat="@drawable/empty_dice"
```

Atrybut app: srcCompat korzysta z biblioteki Androida X do obsługi rysunków wektorowych w starszych wersjach Androida, z powrotem do poziomu API 7.

6. Zbuduj i uruchom swoją aplikację. Na ekranie nie zobaczysz nic innego, ale teraz Twoja aplikacja nie musi używać wygenerowanych plików PNG dla obrazów kości bez względu na to, gdzie się uruchamia, co oznacza mniejszy plik aplikacji.

Wyzwanie: zmodyfikuj aplikację DiceRoller, aby miała dwie kości. Gdy użytkownik stuknie przycisk Rzuć, każda kość powinna mieć wartość niezależną od drugiej.