FIAP ON

TÓPICOS AVANÇADOS DE DESENVOLVIMENTO ANDROID

GESTÃO DE DEPENDÊNCIAS E BUILD

HEIDER PINHOLI LOPES

2

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

2 GESTÃO DE DEPENDÊNCIAS E BUILD

No desenvolvimento de aplicativos Android, é necessário empacotar tudo o que foi desenvolvido antes da distribuição para o usuário. Para que isso aconteça, entram em ação os sistemas de builds.

Os builds são responsáveis por juntar todos os recursos (arquivos Java, Kotlin, XML, entre outros) e utilizar as ferramentas específicas nesse grupo de arquivos, empacotando tudo em um único arquivo conhecido como APK (Android Application Pack) que possui um formato semelhante à compressão que é feita em um arquivo ZIP no qual ficam todos os arquivos necessários para instalação de apps e jogos, podendo ser comparado com os arquivos proprietários de instalação de software do Windows, por exemplo, .exe.

Para esse compilamento, o Google adotou uma ferramenta chamada Gradle.

2.1 O que é o Gradle?

O Gradle (https://gradle.org) é uma ferramenta de automação de build, testes, publicação e deploy que une o melhor da flexibilidade do Ant (https://ant.apache.org) com o gerenciamento de dependências e convenções do Maven (https://maven.apache.org).

Ele utiliza uma DSL (*Domain Specific Language*) baseada em Groovy (https://groovy-lang.org) e Gráfico Acíclico Dirigido (DAG) para determinar em que ordem as tarefas devem ser executadas. Por meio dele, também é possível realizar o gerenciamento de dependências da aplicação e, assim como o Maven, baixar as dependências necessárias e adicionar no projeto.

Com o Gradle é possível definir configurações customizadas de builds, de formas simples e flexíveis, e realizar o build de diversos tipos de projetos, pois ele é baseado em plugins.

No caso do Android, ele é usado desde a primeira versão do Android Studio, porém, uma vez que ele é independente, podemos compilar aplicações Android sem mesmo ter o Android Studio instalado, dando a liberdade de executarmos o processo de build via terminal, sendo uma grande vantagem a compatibilidade com outras

ferramentas, como CI (*Continuous Integration* - Integração Contínua) e CD (*Continuous Delivery* - Entrega Contínua) muito dissolvida no processo de desenvolvimento de software.

2.1.1 Arquivos do Gradle

Nesta seção, o foco será nos arquivos utilizados pelo Gradle em seu processo de build.

> Não se preocupe em localizar esses recursos agora na sua IDE. Iremos desenvolver uma aplicação para explicar cada um deles.

2.1.1.1 Propriedade settings.gradle

Localizado na raiz do projeto, esse arquivo contém a relação dos módulos que o projeto possui.

2.1.1.2 Propriedade build.gradle

Os projetos Android possuem, ao menos, dois arquivos build.gradle. Um localizado na raiz (responsável por estabelecer as configurações que são aplicadas a todos os módulos) e outro para cada módulo (responsável pela configuração do módulo específico).

2.1.1.3 Propriedade gradle.properties

Encontram-se as configurações gerais do Gradle, por exemplo: tamanho da memória *heap*, nível de log a ser exibido, número de *threads*, estilo de código, entre outros.

2.1.1.4 Propriedade local.properties

Nesse arquivo ficam armazenadas configurações locais da máquina como o caminho do SDK (*Software Development Kit*) e do NDK (*Native Development Kit*). Seu conteúdo é gerado automaticamente pelo Android Studio.

2.1.1.5 Propriedade gradle-wrapper

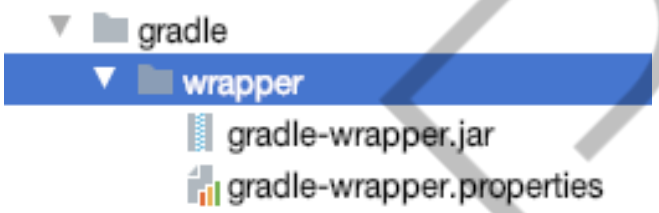


Figura 2.1 – Diretório gradle-wrapper
Fonte: Elaborado pelo autor (2020)

Encontra-se na raiz do projeto e contém os seguintes arquivos:

gradle-wrapper.jar é um script que invoca uma versão específica do Gradle, baixando-o, se necessário. Também possibilita compilar um projeto que utiliza o Gradle, sem ter a necessidade de instalá-lo manualmente.

gradle-wrapper.properties armazena informações sobre a distribuição do Gradle.

2.2 Processo de Build

O processo de build envolve diversas ferramentas e processos, porém, normalmente temos as seguintes etapas:

- Os recursos (strings, colors, styles, drawables, entre outros) são compilados em um arquivo *.arsc (Android Resources), e classe R é gerada;
- O compilador converte o código fonte **Kotlin(*.kt)** e/ou **Java(*.java)** em bytecodes com a extensão ***.class;**
- Esses arquivos são transformados em um ou mais arquivos Dalvik Executable (*.dex);

- O Android Application Pack (APK) é gerado contendo os arquivos gerados.

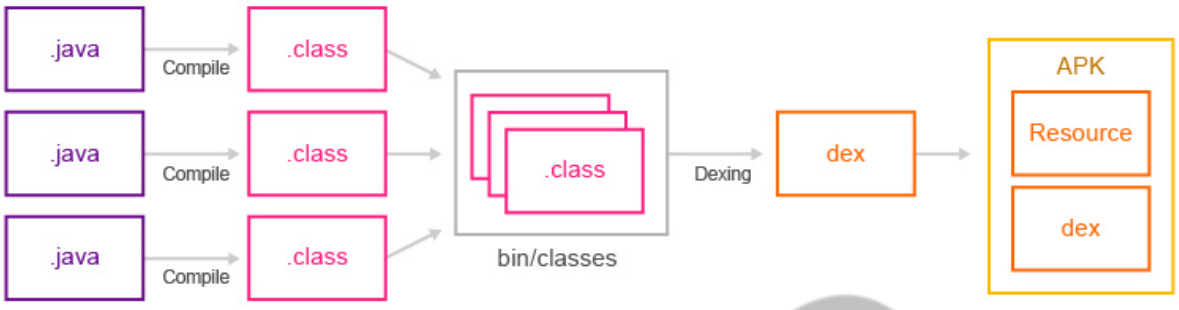


Figura 2.2 – Geração do Android Application Pack (APK)
Fonte: Dangizyan (2019)

2.3 O Projeto

Uma aplicação Android pode ser composta por um ou mais módulos e serem compilados juntos ou separados. Para este capítulo, iremos desenvolver o Jogo 21**,** **em** que cada carta numérica tem o valor do número nela presente. Valete (J), Dama (Q) e Reis (K) valem 10. O Ás vale 1.

O jogo possui como objetivo somar 21 pontos com as cartas ou chegar o mais próximo possível, sem ultrapassar esse valor.

Ganha o jogador que completar a pontuação ou aquele que fizer mais pontos sem estourar os 21 pontos.

2.4 Criando o projeto

Abra o Android Studio e clique em **Start a new Android Studio Project:**

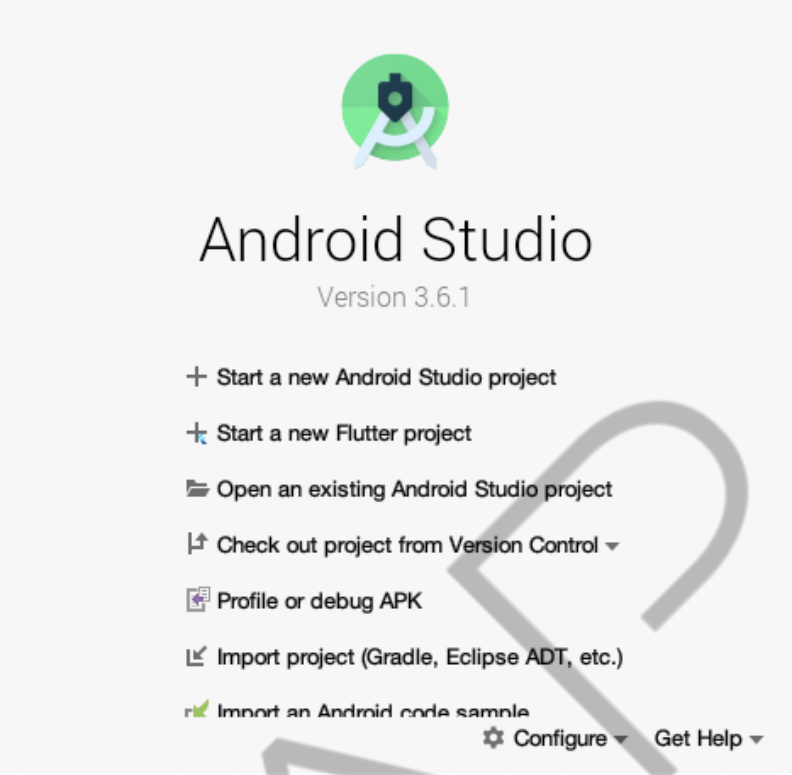


Figura 2.3 – Criação do projeto
Fonte: Elaborado pelo autor (2020)

Em seguida, selecione **Empty Activity** e clique em **Next:**

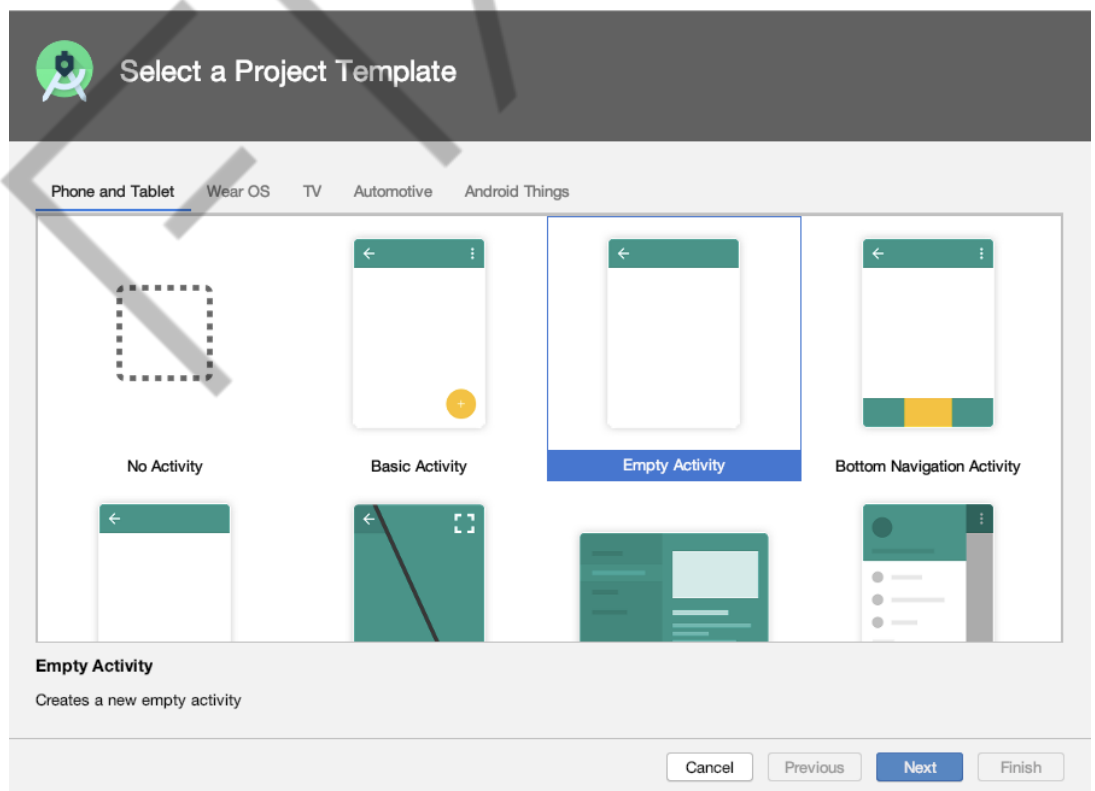


Figura 2.4 – Seleção de template do projeto
Fonte: Elaborado pelo autor (2020)

Configure o seu projeto definindo o Name, Package Name e Language, conforme a próxima imagem:

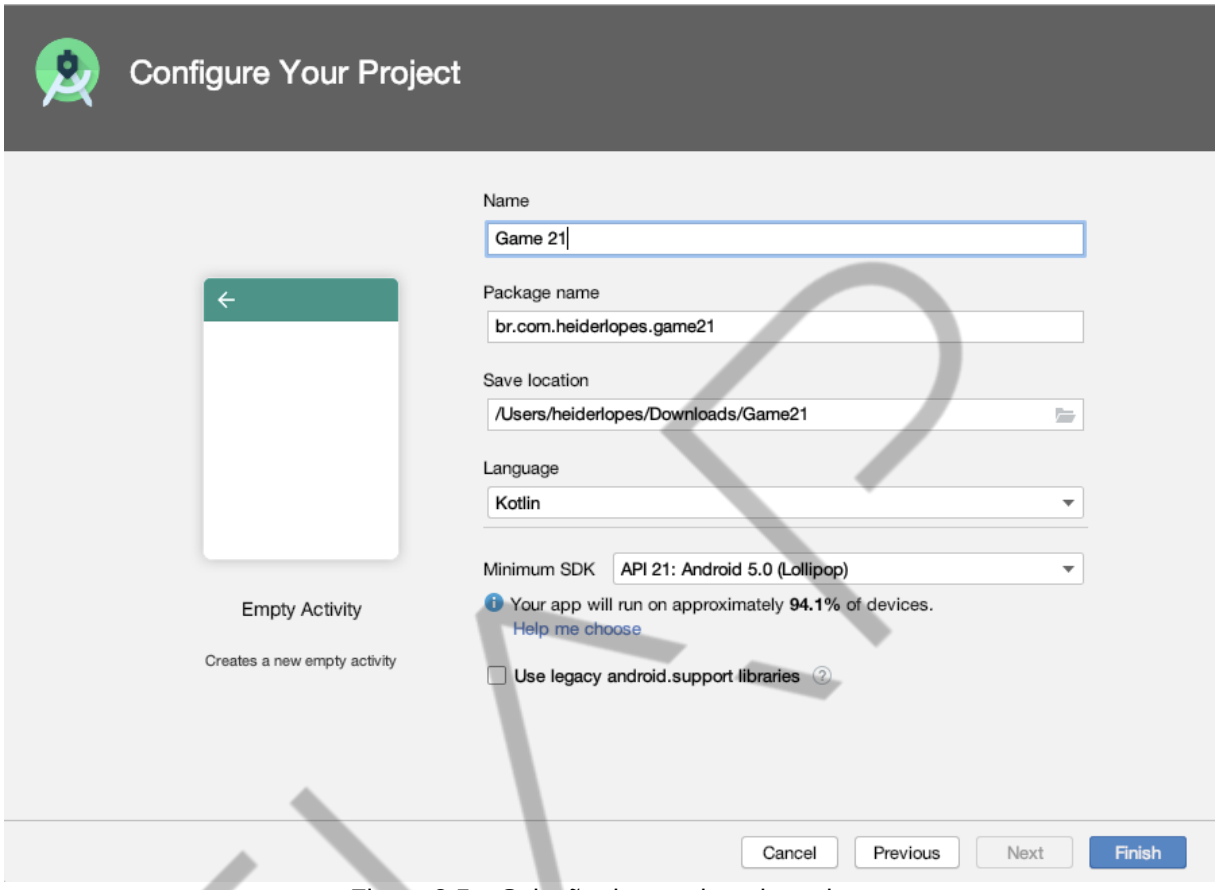


Figura 2.5 – Seleção de template do projeto
Fonte: Elaborado pelo autor (2020)

2.5 Módulos

Uma aplicação Android pode ser composta por um ou mais módulos e serem compilados juntos ou separados. Esses módulos podem ser: Java Library, Android Library ou Android Application.

2.6 Java Library

Contém apenas código-fonte (Java e/ou Kotlin). Por meio dele, é gerado um arquivo JAR. Nesse caso, utiliza-se o plugin Java-Library:

2.6.1 Criando um módulo Java Library

Para criar esse tipo de módulo, selecione **File → New Module:**

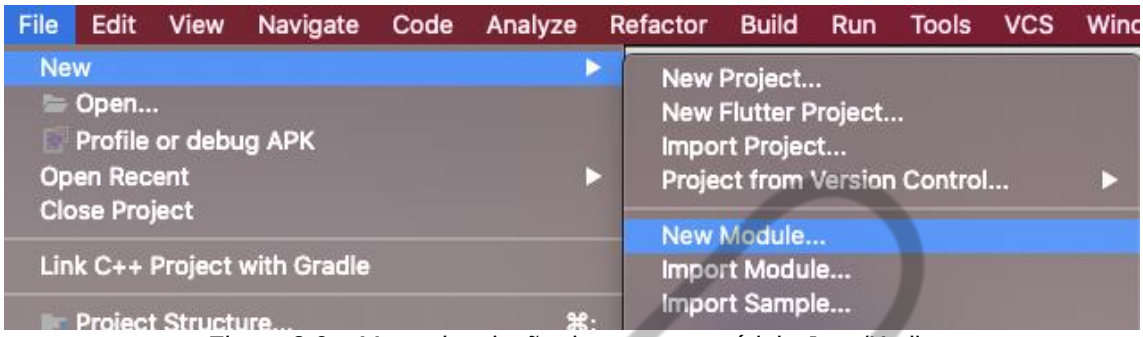


Figura 2.6 – Menu de criação de um novo módulo Java/Kotlin
Fonte: Elaborado pelo autor (2020)

Em Select a Module Type, escolha **Java or Kotlin Library** e clique em **Next:**

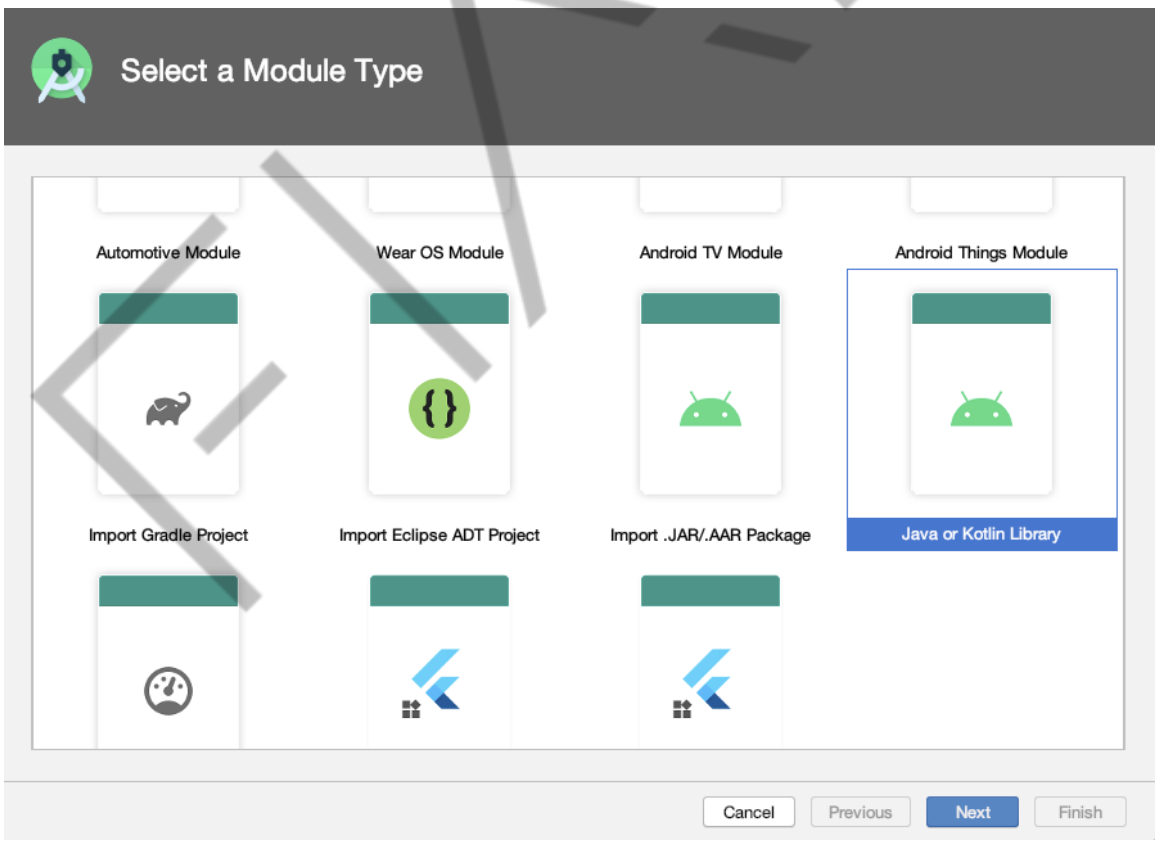


Figura 2.7 – Seleção para criação de um novo módulo Java/Kotlin
Fonte: Elaborado pelo autor (2020)

Defina o nome do módulo com **domain:**

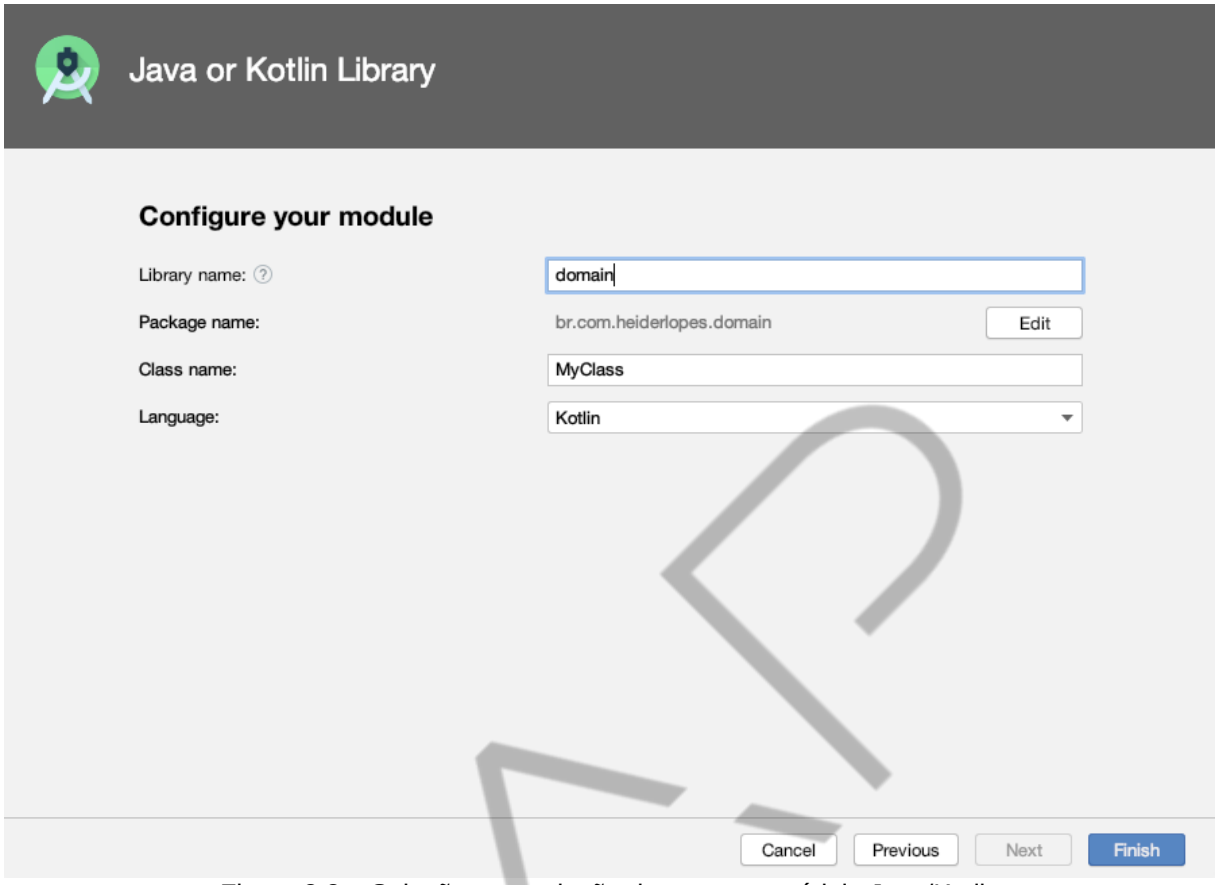


Figura 2.8 – Seleção para criação de um novo módulo Java/Kotlin
Fonte: Elaborado pelo autor (2020)

Após a criação do módulo, clique duas vezes na tecla **Shift** e pesquise o arquivo **build.gradle** (esse arquivo será abordado mais adiante) referente a esse módulo:

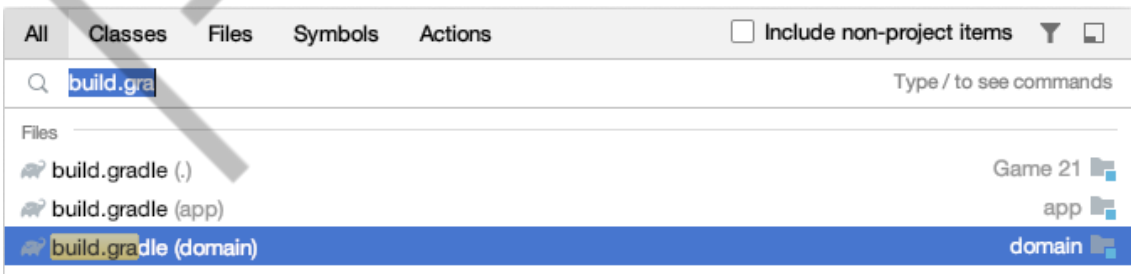


Figura 2.9 – Pesquisando o arquivo build.gradle: referente ao módulo domain
Fonte: Elaborado pelo autor (2020)

Nesse arquivo está sendo aplicado o plugin “java-library”, conforme mencionado anteriormente. Como o módulo dará suporte à linguagem Kotlin, observe que ele também aplica o plugin “kotlin":

```
apply plugin: 'java-library'
apply plugin: 'kotlin'

dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
}

sourceCompatibility = "7"
targetCompatibility = "7"
```

Código-fonte 2.1 – Arquivo build.gradle: referente ao módulo domain
Fonte: Elaborado pelo autor (2020)

Crie um pacote chamado **model:**

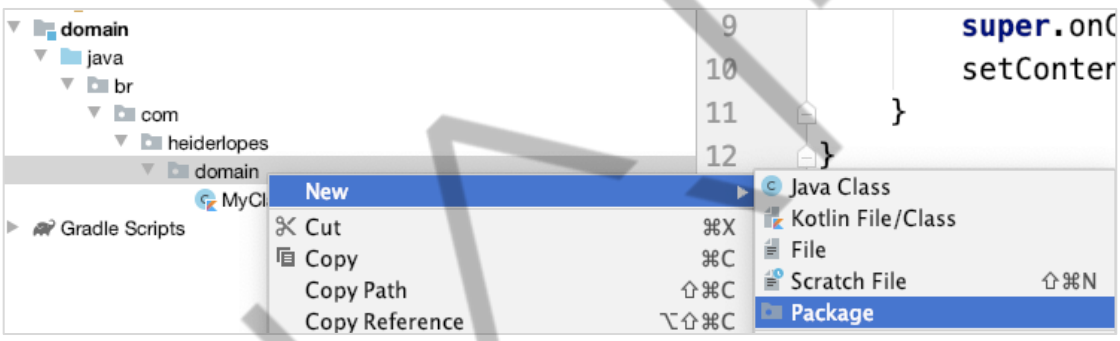


Figura 2.10 – Arquivo build.gradle: referente ao módulo domain
Fonte: Elaborado pelo autor (2020)

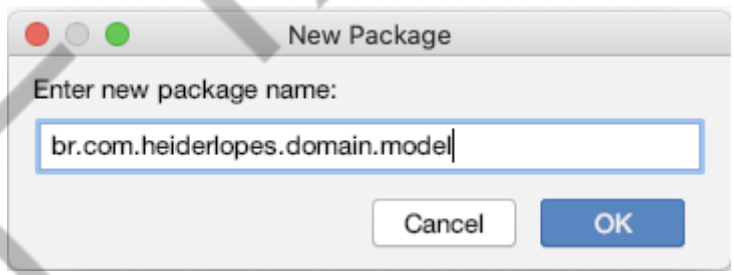


Figura 2.11 – Criação do pacote model
Fonte: Elaborado pelo autor (2020)

Dentro do pacote **model**, crie uma classe Kotlin chamada **Carta:**

Clique com o botão direito sobre model → New → Kotlin File Class:

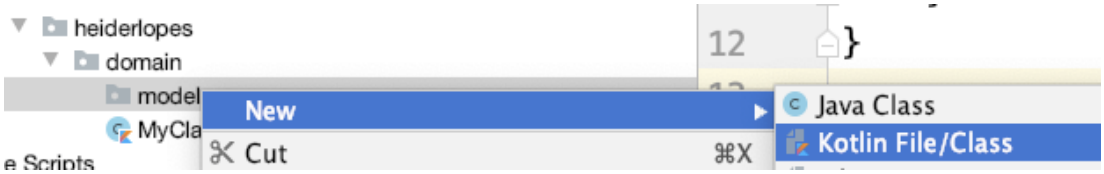


Figura 2.12 – Criação de uma nova classe Kotlin
Fonte: Elaborado pelo autor (2020)

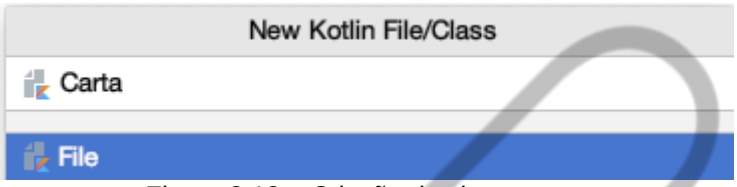


Figura 2.13 – Criação da classe carta
Fonte: Elaborado pelo autor (2020)

Nesse projeto, a carta será representada por um id do recurso (a imagem que será utilizada) e a sua pontuação:

```
data class Carta(
   var resourceId: Int,
   var pontuacao: Int
)
```

Código Fonte 2.2– Classe representando uma carta do jogo
Fonte: Elaborado pelo autor (2020)

2.7 Android Library

Neste tipo de biblioteca é possível ter código-fonte e recursos específicos do Android (por exemplo: layouts, drawables, estilos, entre outros). Ao ser compilado, será gerado um arquivo Android Archive (AAR).

Para este tipo de módulo, aplica-se o plugin ***com.android.library.***

2.7.1 Criando uma Android Library

Como o módulo anterior, este também pode ser criado dentro do mesmo projeto ou em um novo. Para este exemplo, será criada uma biblioteca de componentes que poderá ser reutilizada em outros projetos Android.

Para isso, selecione **File → New → New Project:**

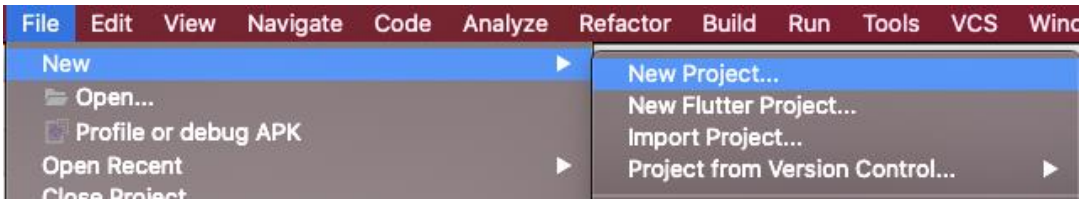


Figura 2.14 – Criando um novo projeto para uma Android Library
Fonte: Elaborado pelo autor (2020)

Selecione **No Activity** e clique em **Next:**

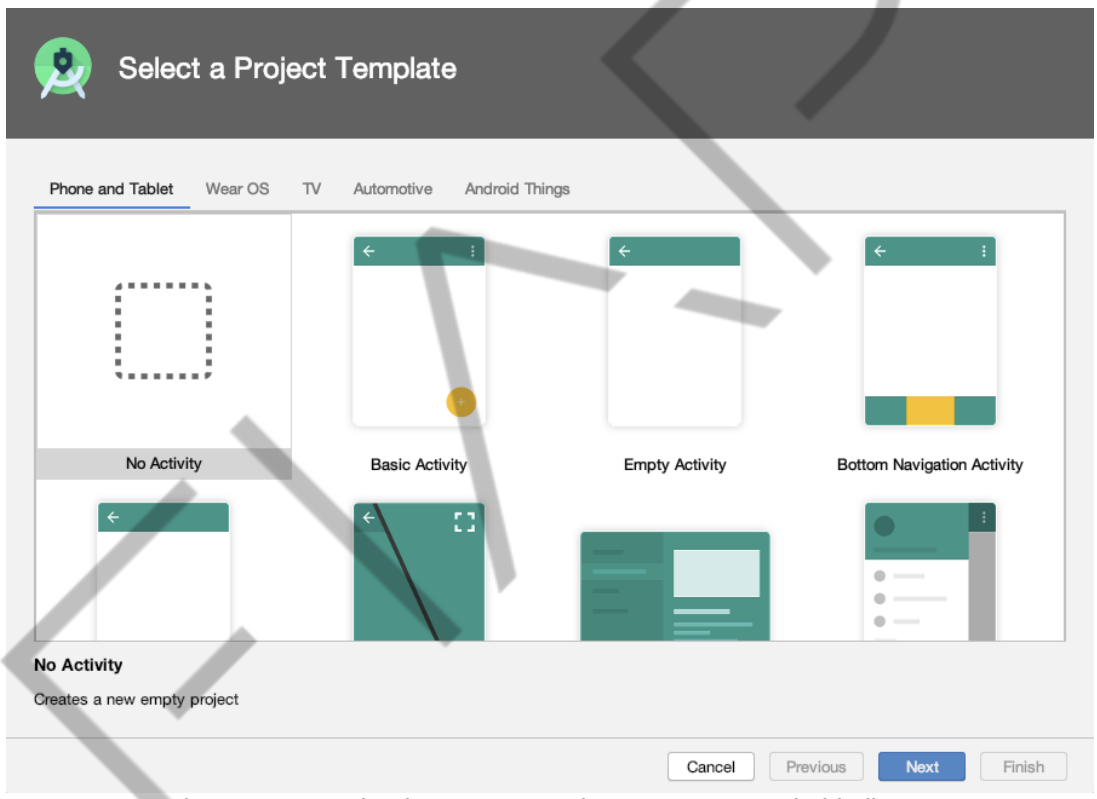


Figura 2.15 – Criando um novo projeto para uma Android Library
Fonte: Elaborado pelo autor (2020)

Configure o projeto de acordo com as configurações abaixo:

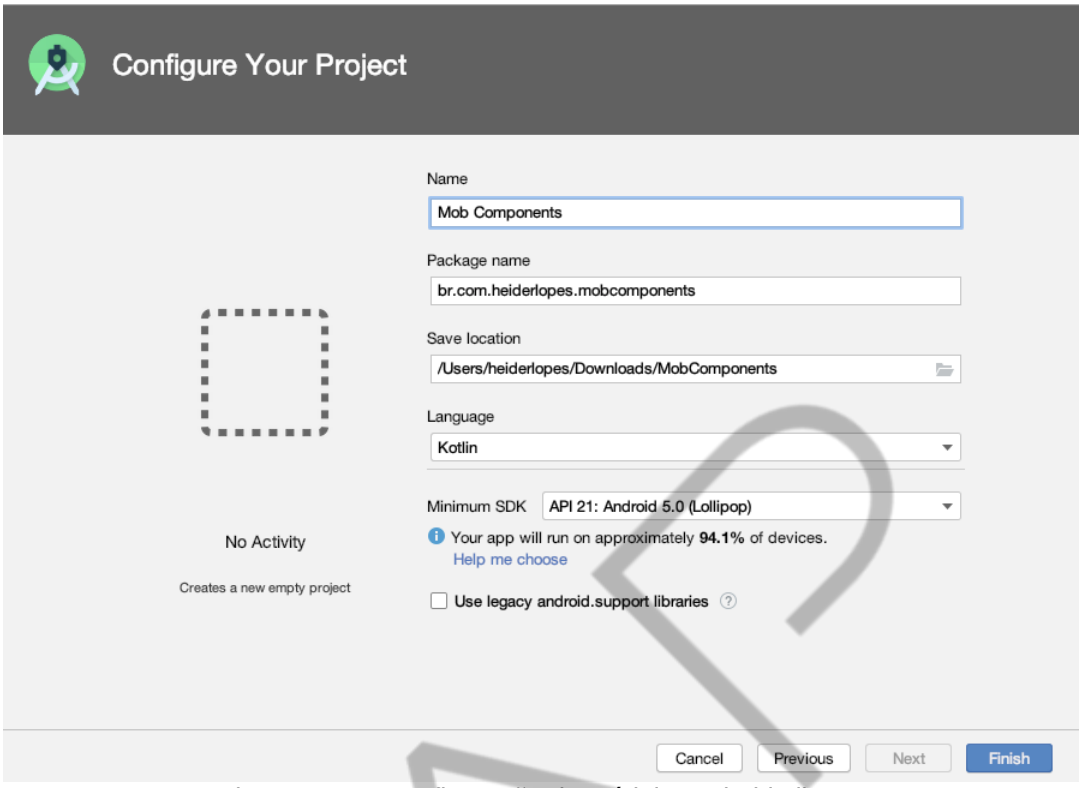


Figura 2.16 – Configuração do módulo Android Library
Fonte: Elaborado pelo autor (2020)

Após a criação do módulo, clique duas vezes na tecla **Shift** e pesquise o arquivo **build.gradle** (esse arquivo será abordado mais adiante) referente a este módulo. Após encontrá-lo, clique para abri-lo:

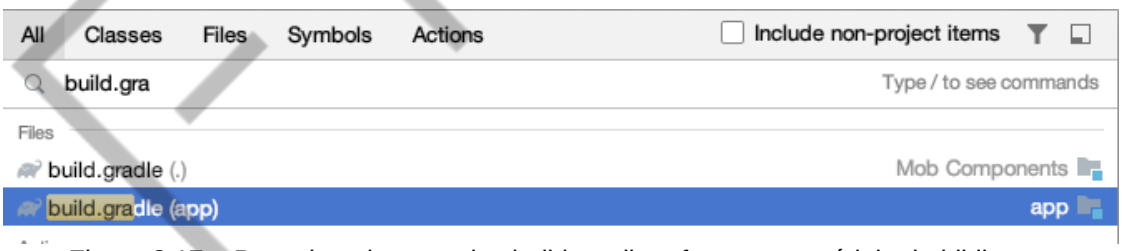


Figura 2.17 – Pesquisando o arquivo build.gradle referente ao módulo da biblioteca
Fonte: Elaborado pelo autor (2020)

Para indicarmos que este projeto é uma Android Library, remova as seguintes linhas em negrito:

apply plugin: 'com.android.application'

applicationId <APPLICATION_ID_DA_SUA_APLICAÇÃO>

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
   compileSdkVersion 29
   buildToolsVersion "29.0.3"

   defaultConfig {
      applicationId "br.com.heiderlopes.mobcomponents"
      minSdkVersion 21
      targetSdkVersion 29
      versionCode 1
      versionName "1.0"

      testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
   }

   buildTypes {
      release {
         minifyEnabled false
         proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
      }
   }

}

dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
   implementation 'androidx.appcompat:appcompat:1.1.0'
   implementation 'androidx.core:core-ktx:1.3.0'
   testImplementation 'junit:junit:4.12'
   androidTestImplementation 'androidx.test.ext:junit:1.1.1'
   androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

Código-fonte 2.3 – Arquivo build.gradle antes da configuração para Android Library
Fonte: Elaborado pelo autor (2020)

E adicione as seguintes linhas no início do arquivo:

apply plugin: 'com.android.library'

apply plugin: 'maven'

```
apply plugin: 'com.android.library'
apply plugin: 'maven'

apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
  compileSdkVersion 29
  buildToolsVersion "29.0.3"

  defaultConfig {
     applicationId "br.com.heiderlopes.mobcomponents"
     minSdkVersion 21
     targetSdkVersion 29
     versionCode 1
     versionName "1.0"

     testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
  }

  buildTypes {
     release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
     }
  }

}

dependencies {
  implementation fileTree(dir: 'libs', include: ['*.jar'])
  implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
  implementation 'androidx.appcompat:appcompat:1.1.0'
  implementation 'androidx.core:core-ktx:1.3.0'
  testImplementation 'junit:junit:4.12'
  androidTestImplementation 'androidx.test.ext:junit:1.1.1'
  androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

Código-fonte 2.4 – Arquivo build.gradle depois da configuração para Android Library
Fonte: Elaborado pelo autor (2020)

Após a configuração do arquivo **build.gradle (app),** clique sobre o botão **Sync Now,** no canto superior direito do Android Studio:

Figura 2.18 – Sincronização da Android Library
Fonte: Elaborado pelo autor (2020)

2.7.2 Criando um componente na Android Library

O componente que será utilizado neste projeto será uma **notificação toast customizada**. Esse recurso é muito útil no Android, pois ele serve para exibir mensagens rápidas e temporais. Ele deve ser utilizado para mensagens com informações simples. Lembrando que se forem mensagens mais críticas, para conter mais recursos, deve-se criar um *dialog* mais sofisticado. Para poder reaproveitá-lo, você também pode criá-lo dentro dessa biblioteca de componentes e reutilizá-lo em vários projetos.

Para esse exemplo, será criado os seguintes tipos de toasts:

- Padrão (Default)
- Erro (Error)
- Sucesso (Success)
- Atenção (Warning)
- Info (Info)

2.7.2.1 Definindo as cores do toast customizado

Para esse recurso, será necessário definir as cores que serão utilizadas como *backgrounds*, as cores dos textos e as formas (*shapes*) referentes ao alerta.

Abra o arquivo **colors.xml** e adicione as seguintes cores:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#6200EE</color>
  <color name="colorPrimaryDark">#3700B3</color>
  <color name="colorAccent">#03DAC5</color>

  <color name="defaultToastBackground">#3c3b39</color>
```

```
    <color name="successToastBackground">#49b150</color>
    <color name="errorToastBackground">#f5443c</color>
    <color name="infoToastBackground">#3e51b7</color>
    <color name="warningToastBackground">#ff9a00</color>

    <color name="defaultToastTextColor">#FFF</color>
    <color name="successToastTextColor">#FFF</color>
    <color name="errorToastTextColor">#FFF</color>
    <color name="infoToastTextColor">#FFF</color>
    <color name="warningToastTextColor">#FFF</color>

</resources>
```

Código-fonte 2.5 – Arquivo build.gradle depois da configuração para Android Library
Fonte: Elaborado pelo autor (2020)

2.7.2.2 Definindo as dimensões do toast customizado

Crie um arquivo chamado **dimens.xml** dentro da pasta **values**. Em seguida, clique, com o botão direito, sobre a pasta **values** → **New** → **Values Resource File:**

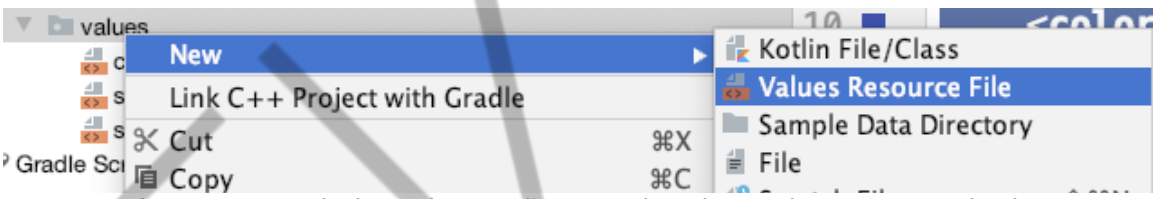


Figura 2.19 – Selecionando a opção para criar o layout do toast customizado
Fonte: Elaborado pelo autor (2020)

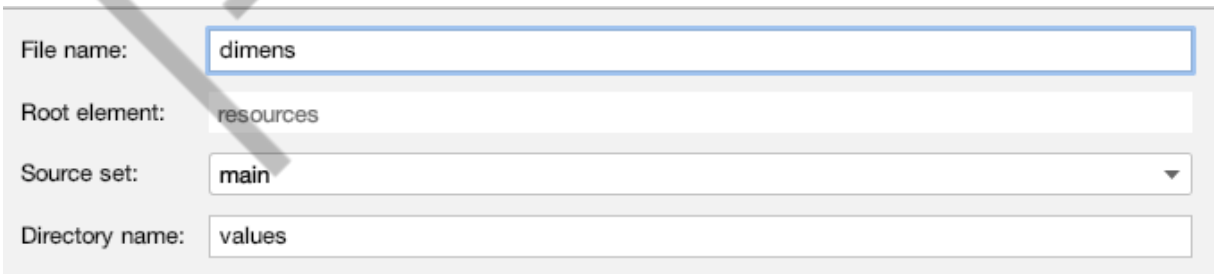


Figura 2.20 – Selecionando a opção para criar o layout do toast customizado
Fonte: Elaborado pelo autor (2020)

Dentro desse arquivo, adicione o radius que iremos utilizar no nosso Toast Customizado:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <dimen name="radius_custom_toast">5dp</dimen>
</resources>
```

Código-fonte 2.6 – Selecionando a opção para criar o layout do toast customizado
Fonte: Elaborado pelo autor (2020)

2.7.2.3 Criando o shape de sucesso

Agora crie o arquivo para o background referente ao sucesso. Clique com o botão direito sobre a pasta **drawable → New → Drawable Resource File:**

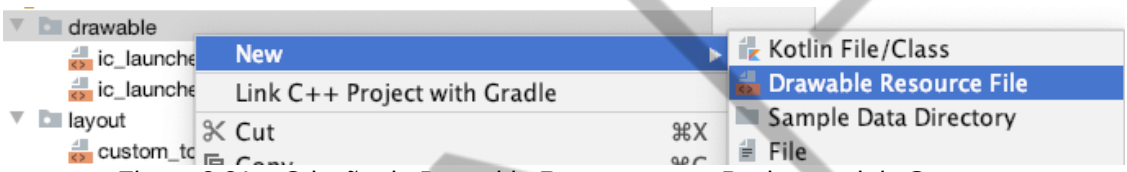


Figura 2.21 – Criação do Drawable Resource para Background de Sucesso
Fonte: Elaborado pelo autor (2020)

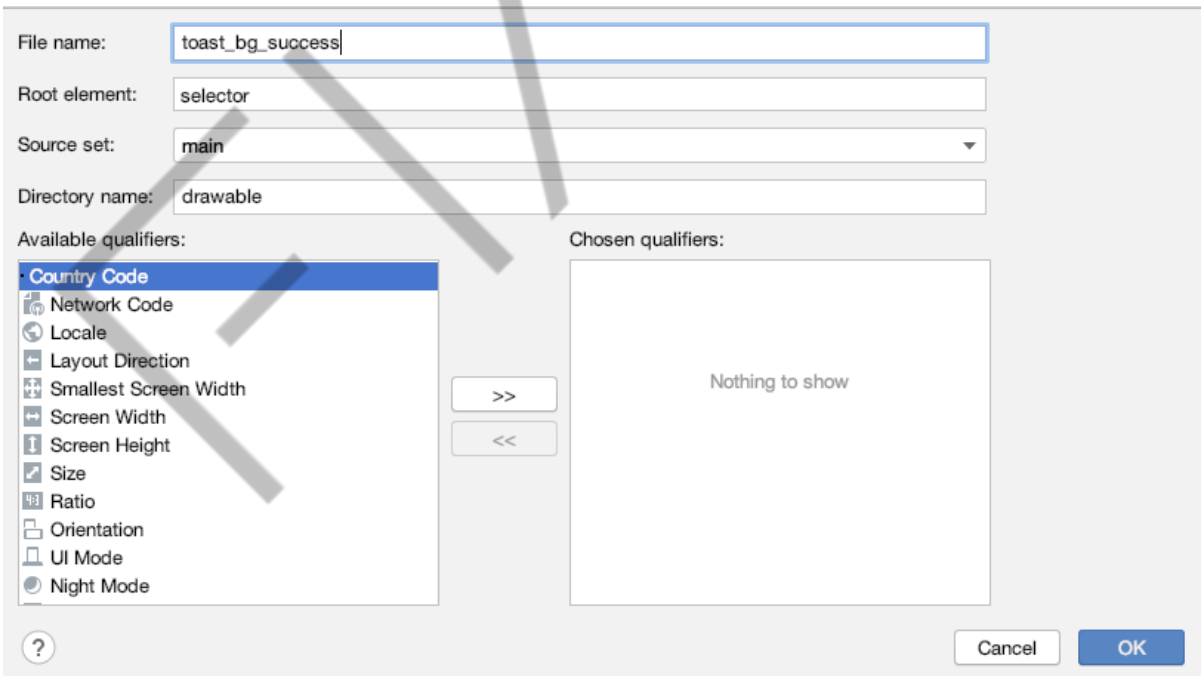


Figura 2.22 – Nomeação do Drawable Resource para Background de Sucesso
Fonte: Elaborado pelo autor (2020)

Adicione o seguinte código referente ao shape de sucesso:

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  android:shape="rectangle">

  <solid android:color="@color/successToastBackground" />

  <corners android:radius="@dimen/radius_custom_toast" />
</shape>
```

Código-fonte 2.7 – Drawable Resource para Background de Sucesso
Fonte: Elaborado pelo autor (2020)

2.7.2.4 Criando o shape de erro

Agora crie o arquivo para o background referente ao sucesso. Clique com o botão direito sobre a pasta **drawable → New → Drawable Resource File:**

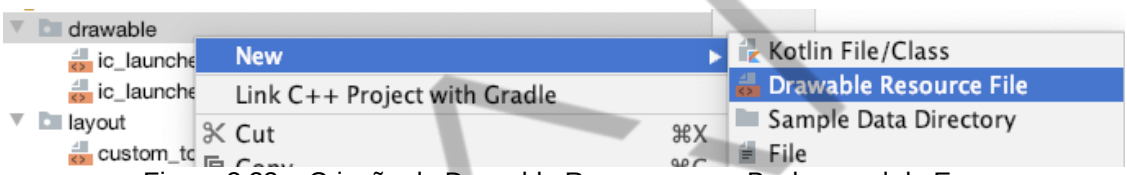


Figura 2.23 – Criação do Drawable Resource para Background de Erro
Fonte: Elaborado pelo autor (2020)

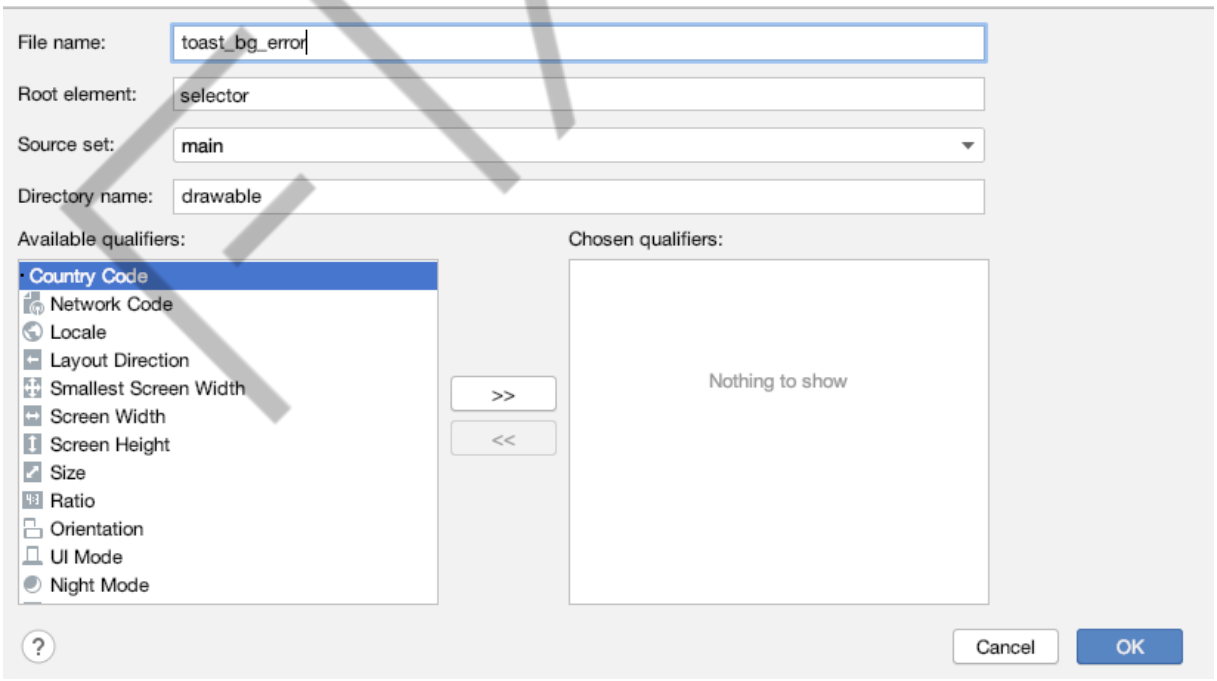


Figura 2.24 – Nomeação do Drawable Resource para Background de Erro
Fonte: Elaborado pelo autor (2020)

Adicione o seguinte código referente ao shape de erro:

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">

  <solid android:color="@color/errorToastBackground" />

  <corners android:radius="@dimen/radius_custom_toast" />

</shape>
```

Código-fonte 2.8 – Drawable Resource para Background de Erro
Fonte: Elaborado pelo autor (2020)

2.7.2.5 Criando o shape de warning

Agora crie o arquivo para o background referente ao sucesso. Clique com o botão direito sobre a pasta **drawable → New → Drawable Resource File:**

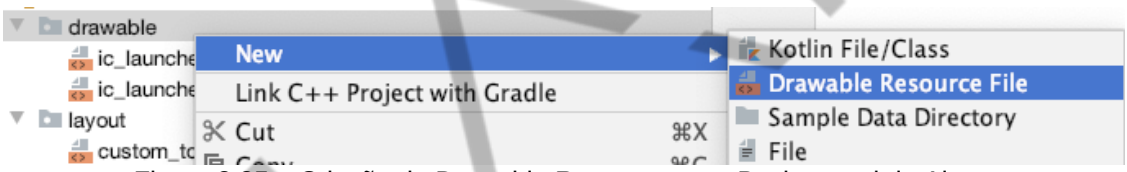


Figura 2.25 – Criação do Drawable Resource para Background de Alerta
Fonte: Elaborado pelo autor (2020)

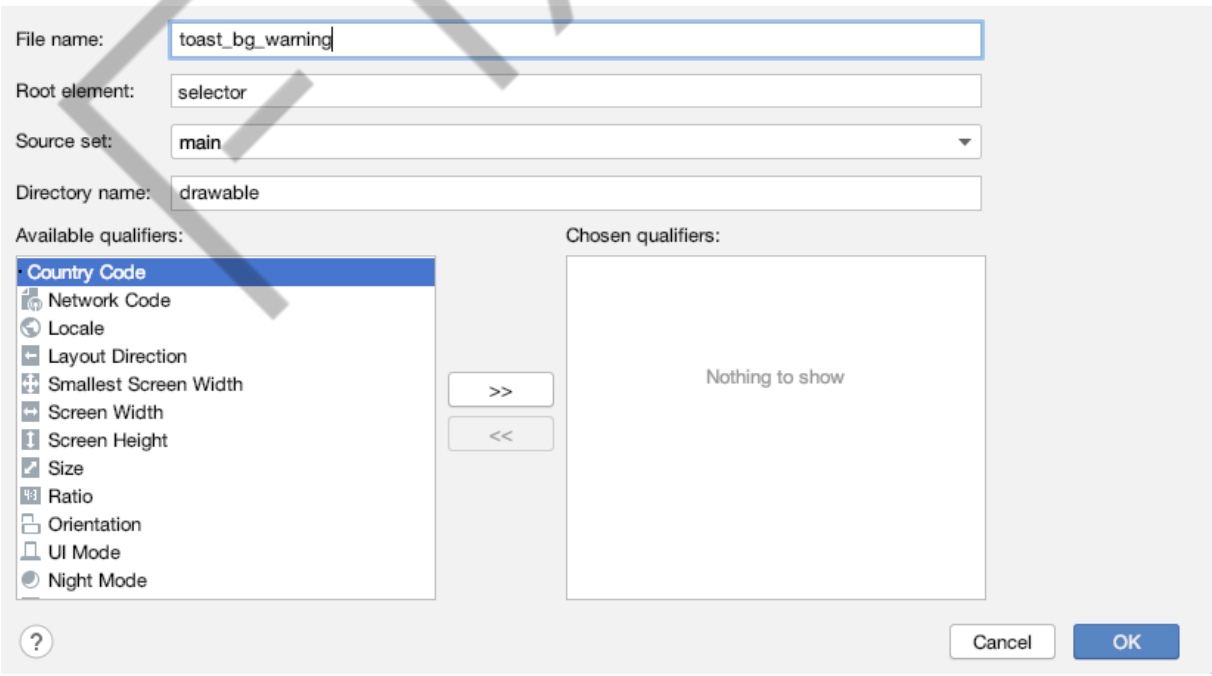


Figura 2.26 – Nomeação do Drawable Resource para Background de Alerta
Fonte: Elaborado pelo autor (2020)

Adicione o seguinte código referente ao shape de atenção:

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">

  <solid android:color="@color/warningToastBackground" />

  <corners android:radius="@dimen/radius_custom_toast" />

</shape>
```

Código-fonte 2.9 – Drawable Resource para Background de Alerta
Fonte: Elaborado pelo autor (2020)

2.7.2.6 Criando o shape de default

Agora crie o arquivo para o background referente ao sucesso. Clique com o botão direito sobre a pasta **drawable → New → Drawable Resource File:**

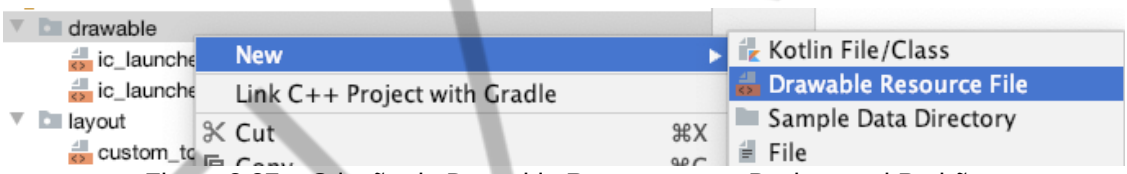


Figura 2.27 – Criação do Drawable Resource para Background Padrão
Fonte: Elaborado pelo autor (2020)

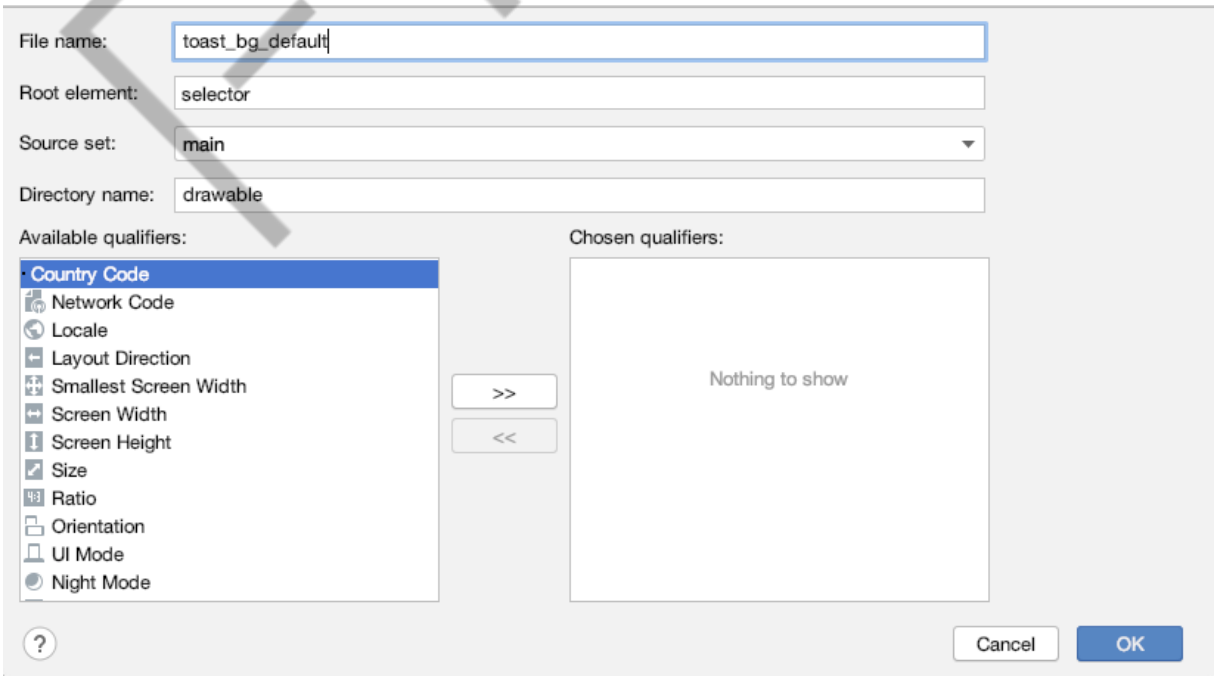


Figura 2.28 – Nomeação do Drawable Resource para Background Padrão

Fonte: Elaborado pelo autor (2020)

Adicione o seguinte código referente ao shape de padrão:

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">

  <solid android:color="@color/defaultToastBackground" />

  <corners android:radius="@dimen/radius_custom_toast" />

</shape>
```

Código-fonte 2.10 – Drawable Resource para Background Padrão
Fonte: Elaborado pelo autor (2020)

2.7.2.7 Criando o shape de info

Agora crie o arquivo para o background referente ao sucesso. Clique com o botão direito sobre a pasta **drawable → New → Drawable Resource File:**

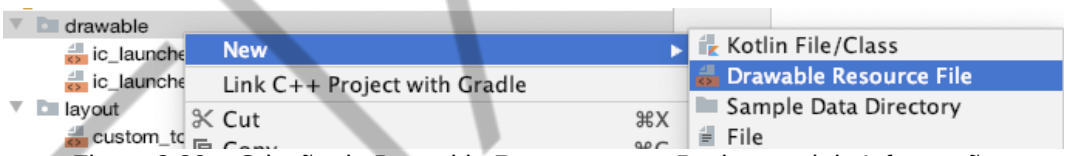


Figura 2.29 – Criação do Drawable Resource para Background de Informação
Fonte: Elaborado pelo autor (2020)

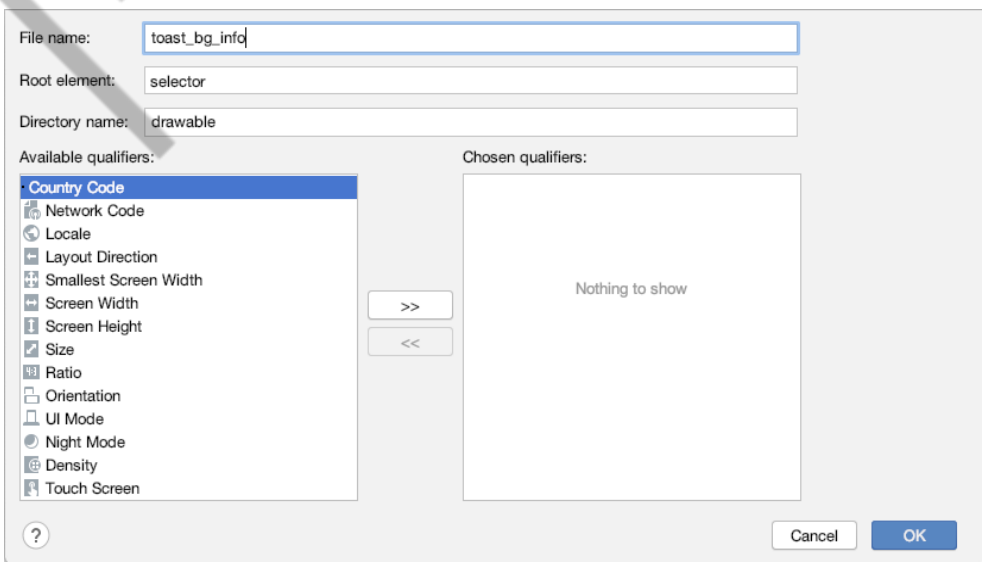


Figura 2.30 – Nomeação do Drawable Resource para Background de Informação
Fonte: Elaborado pelo autor (2020)

Adicione o seguinte código referente ao shape de info:

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">

  <solid android:color="@color/infoToastBackground" />

  <corners android:radius="@dimen/radius_custom_toast" />

</shape>
```

Código-fonte 2.11 – Drawable Resource para Background de Informação
Fonte: Elaborado pelo autor (2020)

2.7.2.8 Adicionando as imagens

Adicione as imagens que serão utilizadas em nosso Toast Customizado. Você deve baixá-las do repositório:

https://github.com/FIAPON/AndroidToastIconsCustom

E, em seguida, adicioná-las na pasta drawable do projeto:

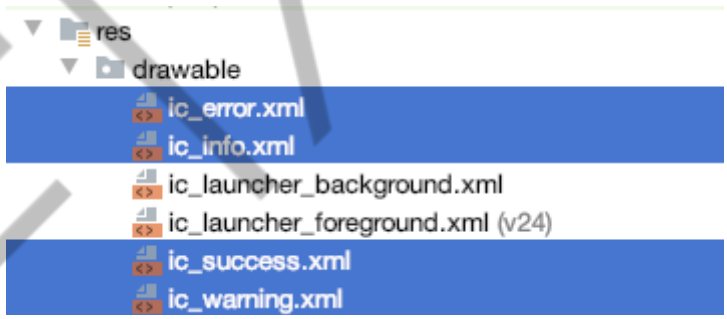


Figura 2.31 – Inclusão das imagens utilizadas no Toast
Fonte: Elaborado pelo autor (2020)

2.7.2.9 Criando o layout do toast customizado

Agora crie o arquivo referente ao layout do toast customizado. Clique com o botão direito sobre a pasta **res → New → Android Resource File:**

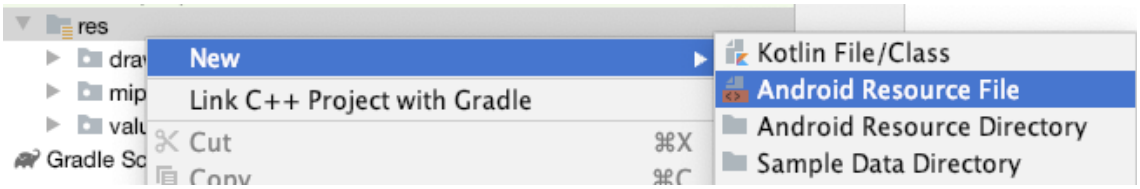


Figura 2.32 – Selecionando a opção para criar o layout do toast customizado
Fonte: Elaborado pelo autor (2020)

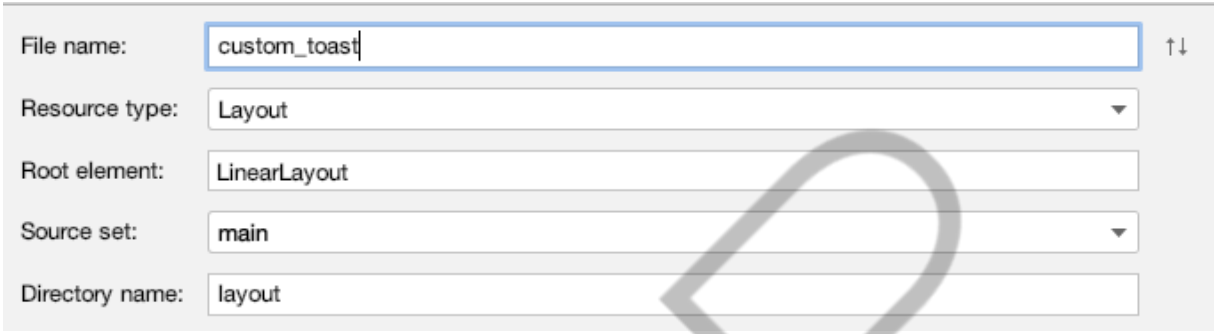


Figura 2.33 – Configurando o layout do toast customizado
Fonte: Elaborado pelo autor (2020)

Em seguida, adicione o seguinte código referente ao layout do toast customizado:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_margin="8dp"
  android:paddingHorizontal="8dp"
  android:background="@drawable/toast_bg_success"
  android:orientation="horizontal">

  <ImageView
    android:id="@+id/ivIconToast"
    android:padding="8dp"
    android:layout_width="48dp"
    android:layout_height="48dp"
    android:layout_gravity="center"
    android:src="@drawable/ic_success" />

  <TextView
    android:id="@+id/tvMessageToast"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    tools:text="Mensagem do toast"
    android:textColor="@color/successToastTextColor" />
```

```
</LinearLayout>
```

Código-fonte 2.12 – Configurando o layout do toast customizado
Fonte: Elaborado pelo autor (2020)

Clique, com o botão direito, sobre o pacote **mobcomponets** e crie uma classe Kotlin, chamada **CustomToast.kt:**

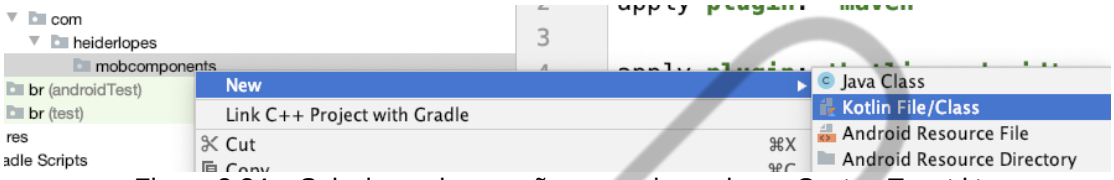


Figura 2.34 – Selecionando a opção para criar a classe CustomToast.kt
Fonte: Elaborado pelo autor (2020)

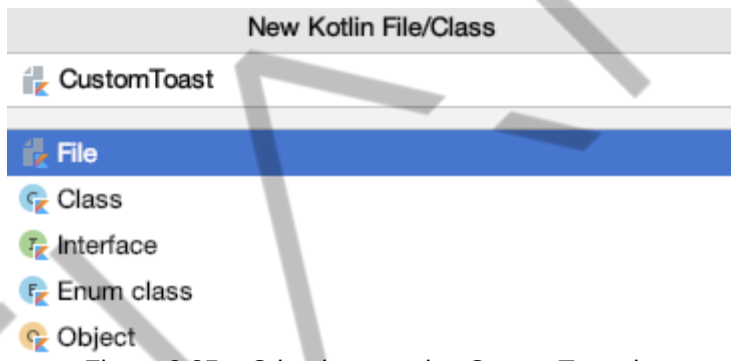


Figura 2.35 – Criando o arquivo CustomToast.kt
Fonte: Elaborado pelo autor (2020)

Segue o código responsável pelo gerenciamento do nosso toast customizado:

```
import android.app.Activity
import android.graphics.drawable.Drawable
import android.view.View
import android.widget.ImageView
import android.widget.TextView
import android.widget.Toast
import androidx.core.content.ContextCompat

object CustomToast {

   private fun showToast(
      activity: Activity,
      backgroundToast: Drawable?,
      icon: Drawable?,
      message: String,
```

```
        duration: Int = Toast.LENGTH_SHORT
    ) {
        val toastLayout: View = activity.layoutInflater.inflate(R.layout.custom_toast,
null)
        val toast = Toast(activity)
        toast.view = toastLayout
        toast.view.background = backgroundToast
        toastLayout.findViewById<TextView>(R.id.tvMessageToast).text = message
        val ivIconToast = toastLayout.findViewById<ImageView>(R.id.ivIconToast)
        if (icon == null) {
            ivIconToast.visibility = View.GONE
        } else {
            ivIconToast.visibility = View.VISIBLE
            ivIconToast.setImageDrawable(icon)
        }
        toast.duration = duration
        toast.show()
    }

    fun success(activity: Activity, message: String) {
        showToast(
            activity,
            ContextCompat.getDrawable(activity, R.drawable.toast_bg_success),
            ContextCompat.getDrawable(activity, R.drawable.ic_success),
            message
        )
    }

    fun warning(activity: Activity, message: String) {
        showToast(
            activity,
            ContextCompat.getDrawable(activity, R.drawable.toast_bg_warning),
            ContextCompat.getDrawable(activity, R.drawable.ic_warning),
            message
        )
    }

    fun error(activity: Activity, message: String) {
        showToast(
            activity,
            ContextCompat.getDrawable(activity, R.drawable.toast_bg_error),
            ContextCompat.getDrawable(activity, R.drawable.ic_error),
            message
        )
    }

    fun info(activity: Activity, message: String) {
        showToast(
            activity,
```

```
            ContextCompat.getDrawable(activity, R.drawable.toast_bg_info),
            ContextCompat.getDrawable(activity, R.drawable.ic_info),
            message
        )
    }

    fun default(activity: Activity, message: String) {
        showToast(
            activity,
            ContextCompat.getDrawable(activity, R.drawable.toast_bg_default),
            null,
            message
        )
    }
}
```

Código-fonte 2.13 – Classe CustomToast.kt completa
Fonte: Elaborado pelo autor (2020)

2.7.3 Publicando a biblioteca

Para que o projeto possa ser utilizado em outros projetos, a lib pode ser disponibilizada em um servidor de arquivos. Nesse exemplo, iremos versionar o projeto no **GitHub** (https://www.github.com) e, em seguida, iremos publicá-lo no **JitPack** (https://www.jitpack.io).

O JitPack é um repositório de pacotes para projetos Android e outras linguagens baseadas em JVM, no qual qualquer desenvolvedor ou organização pode publicar os próprios pacotes e bibliotecas. Uma vez publicado um pacote no JitPack, ele ficará disponível para qualquer pessoa utilizar.

2.7.3.1 Versionamento da biblioteca no GitHub

Para enviar a biblioteca para o GitHub, podemos fazer o share do Projeto. Para isso, clique no menu **VCS → Import into Version Control → Share Project on GitHub.**

Você também pode usar os comandos Git em linha de comando (CLI).

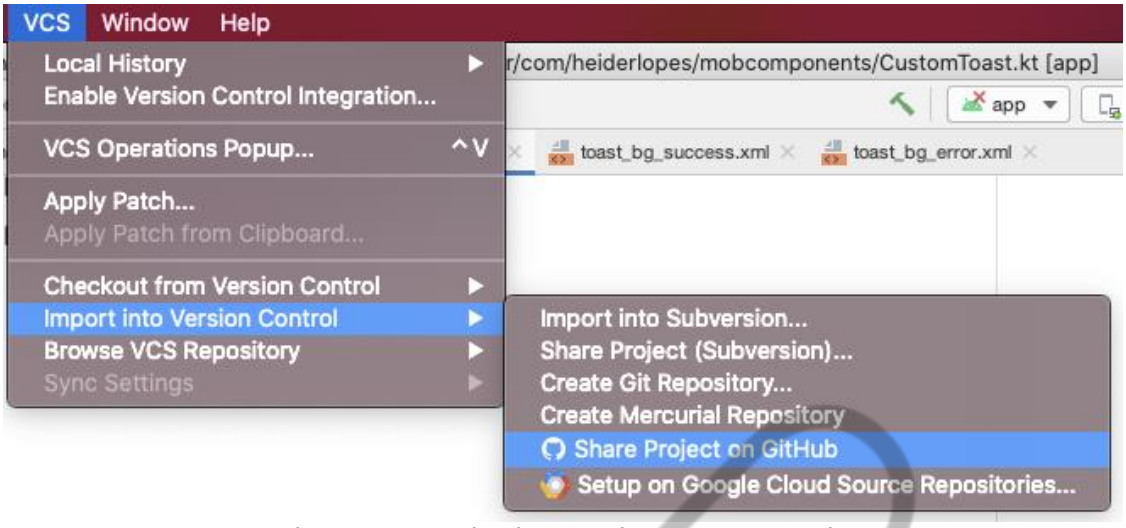


Figura 2.36 – Criando o arquivo CustomToast.kt
Fonte: Elaborado pelo autor (2020)

Defina o nome do repositório e clique em **Share:**

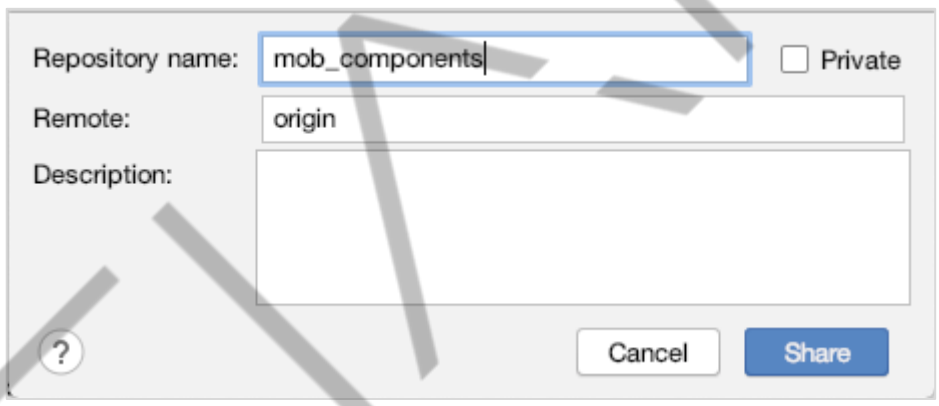


Figura 2.37 – Nome do repositório
Fonte: Elaborado pelo autor (2020)

Adicione todos os arquivos do projeto e a mensagem referente ao commit e, em seguida, clique em **Add:**

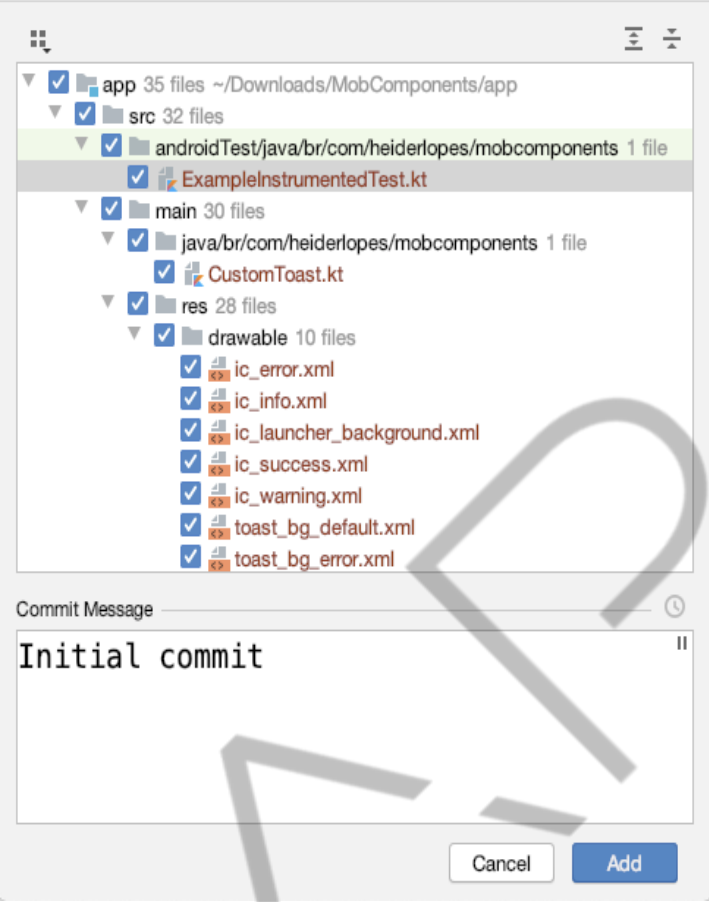


Figura 2.38 – Arquivos versionados
Fonte: Elaborado pelo autor (2020)

2.7.3.2 Criando Tag no GitHub

Acesse o repositório criado no GitHub, clique em **Master → Tags → View all tags:**

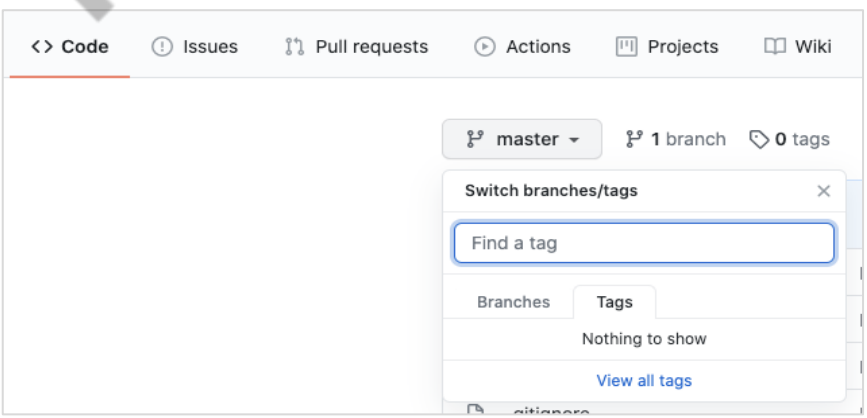


Figura 2.39 – Tags no repositório do GitHub
Fonte: Elaborado pelo autor (2020)

Clique sobre o botão: **Create a new release:**

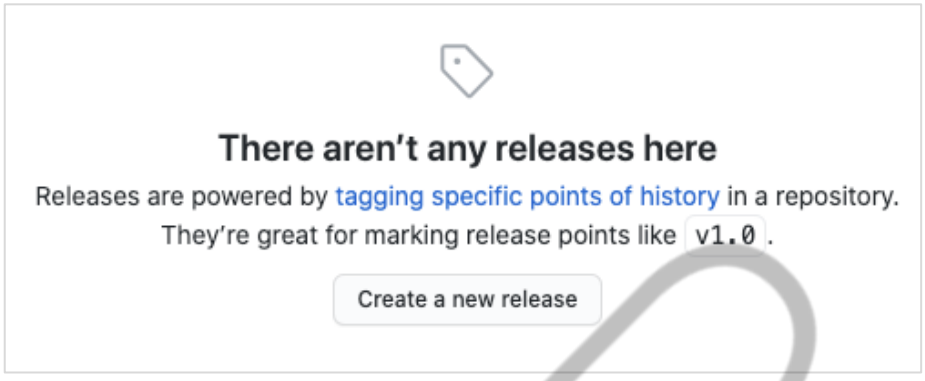


Figura 2.40 – Criando um novo release do componente no GitHub
Fonte: Elaborado pelo autor (2020)

Preencha os dados referente à criação do release e clique no botão **Publish release:**

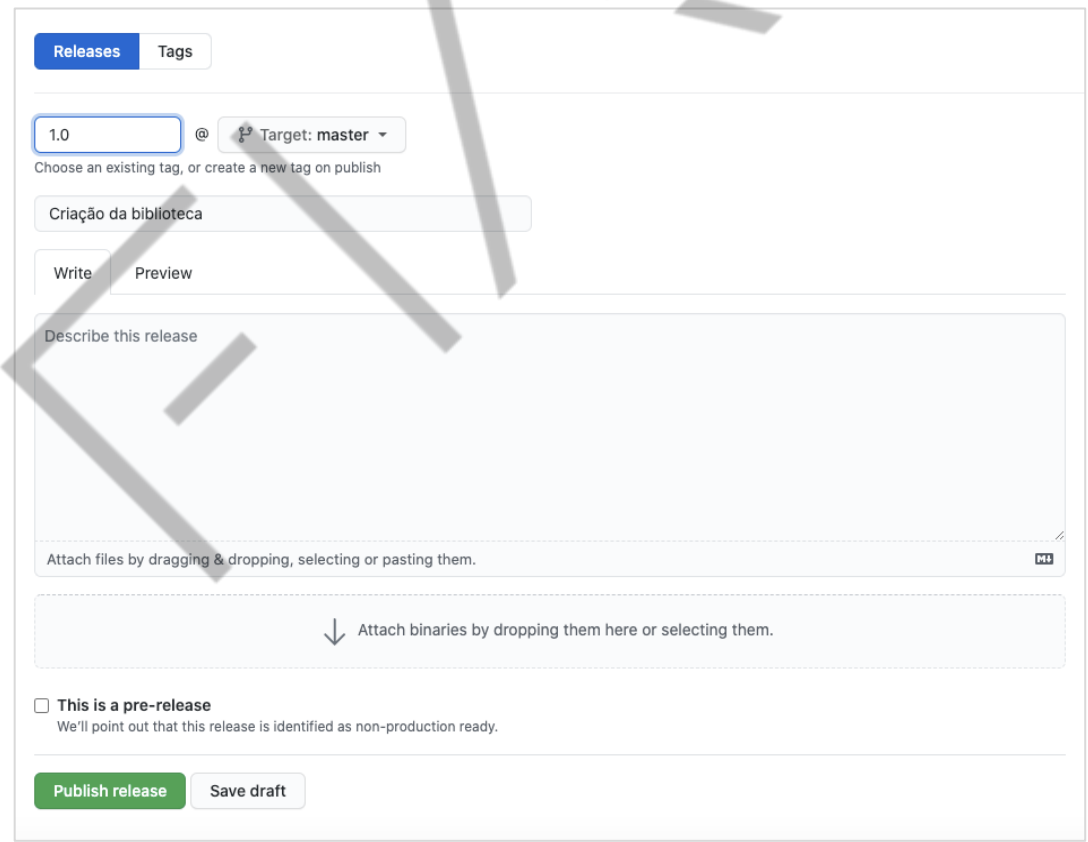


Figura 2.41 – Publicando o novo release no GitHub
Fonte: Elaborado pelo autor (2020)

2.7.3.3 Publicando no JitPack

Acesse o JitPack em: https://jitpack.io

Figura 2.42 – Site do JitPack
Fonte: Elaborado pelo autor (2020)

Faça login com seu GitHub, clicando sobre **Sign In:**

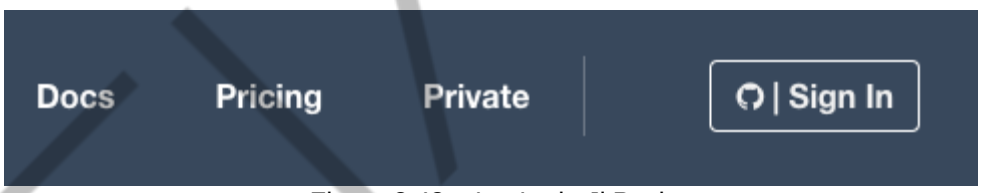


Figura 2.43 – LogIn do JitPack
Fonte: Elaborado pelo autor (2020)

Em seguida, digite o nome da biblioteca e clique em **Look up:**

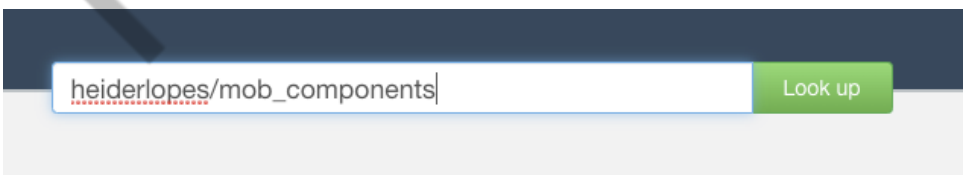


Figura 2.44 – Pesquisa do repositório
Fonte: Elaborado pelo autor (2020)

Após clicar em Look up, o JitPack mostra como adicionar a dependência em seu projeto, provendo instruções específicas no Gradle, dentre elas:

1. Como adicionar o JitPack como provedor de pacotes.

2. Como adicionar a dependência específica a partir do repositório do GitHub.

Mais adiante, essa biblioteca será utilizada.

2.7.4 Android Application

É a aplicação que será executada em dispositivos Android, podendo depender de outros módulos ou de bibliotecas locais ou externas. Normalmente, a maioria dos projetos possuem apenas um módulo dessa categoria. Mas nada impede de ter um módulo de application contendo uma versão de Auto, Wear ou TV. O plugin aplicado é o **apply plugin: 'com.android.application'.**

No projeto Game 21, abra o arquivo **build.gradle (app)** e observe que o plugin mencionado acima está sendo aplicado:

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
// restante do arquivo
```

Código-fonte 2.14 – Arquivo build.gradle (app)
Fonte: Elaborado pelo autor (2020)

2.7.5 Dependências

Um projeto Android poderá utilizar classes ou recursos que não estejam necessariamente no código do seu aplicativo. Com isso, seu projeto pode conter: dependências de módulos, dependências locais ou dependências remotas.

2.7.5.1 Dependências de módulos

Ocorre quando um módulo inclui em seu build.gradle dependência de outro módulo. Quando um módulo for compilado, a dependência também será compilada e incluída no arquivo final.

Neste projeto, o módulo **domain** deverá ser adicionado no módulo app. Para isso, abra o arquivo build.gradle (app) e adicione a dependência, conforme destaque abaixo:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation project(":domain")

    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    implementation 'com.github.heiderlopes:mob_components:1.0'
}
```

Código-fonte 2.15 – Arquivo build.gradle do módulo domain
Fonte: Elaborado pelo autor (2020)

2.7.5.2 Dependências locais

Ocorre quando o projeto possui arquivos binários (por exemplo: jar) no computador. Nesse caso, também devemos declarar a dependência no build.gradle.

2.7.5.3 Dependências remotas

Existe uma grande quantidade de bibliotecas disponíveis em repositórios remotos (igual à biblioteca de componentes criada anteriormente que foi disponibilizada no JitPack), os mais populares são JCenter e o Maven, que são suportados pelo Android Studio. Dessa forma, basta informar onde está a biblioteca seguinte, normalmente com o padrão grupo:nome:versão.

Para incluir a biblioteca Mob Components no projeto Game 21 (ou qualquer outro projeto Android), abra o arquivo **build.gradle (.)** e adicione a seguinte linha em destaque:

```
allprojects {
   repositories {
      google()
      jcenter()
      maven { url 'https://jitpack.io' }
   }
}
```

Código-fonte 2.16 – Inclusão do repositório JitPack
Fonte: Elaborado pelo autor (2020)

Abra o arquivo **build.gradle (app)** e adicione a seguinte linha:

```
dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
   implementation 'androidx.appcompat:appcompat:1.1.0'
   implementation 'androidx.core:core-ktx:1.3.0'
   implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
   testImplementation 'junit:junit:4.12'
   androidTestImplementation 'androidx.test.ext:junit:1.1.1'
   androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

   implementation 'com.github.heiderlopes:mob_components:1.0'
}
```

Código-fonte 2.17 – Inclusão da biblioteca Mob Components criada
Fonte: Elaborado pelo autor (2020)

E clique em **Sync now** para baixar as dependências necessárias:

Figura 2.45 – Sincronização do gradle
Fonte: Elaborado pelo autor (2020)

2.7.6 Programando o jogo

Baixe as imagens que serão utilizadas no jogo, encontradas no repositório: <https://github.com/FIAPON/AndroidJogo21>. Adicione as imagens na pasta **drawable,** da pasta do projeto, (estamos usando o nome “Game 21”, mas, originalmente, se você clonar o repositório, o nome da pasta raiz será “AndroidJogo21”):

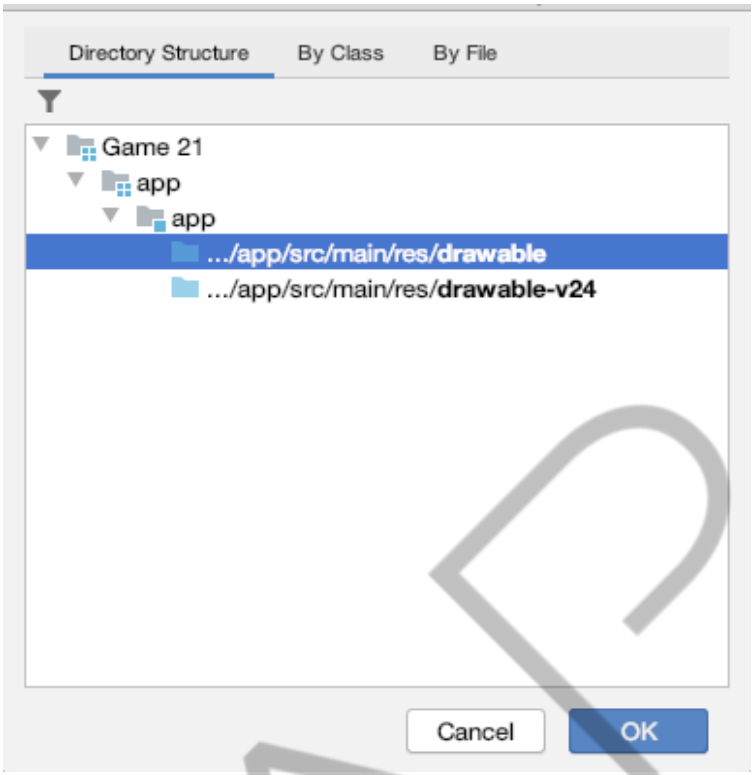


Figura 2.46 – Inclusão das imagens do jogo
Fonte: Elaborado pelo autor (2020)

Abra o arquivo **activity_main.xml** e adicione o seguinte código:

```
<androidx.constraintlayout.widget.ConstraintLayout
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:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:layout_editor_absoluteY="81dp">


   <ImageView
      android:id="@+id/ivCarta"
      android:layout_width="0dp"
      android:layout_height="0dp"
      android:layout_marginStart="32dp"
      android:layout_marginTop="32dp"
      android:layout_marginEnd="32dp"
      android:layout_marginBottom="32dp"
      app:layout_constraintBottom_toTopOf="@+id/btProximaCarta"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toBottomOf="@+id/textView"
      tools:src="@drawable/as_de_espada" />
```

```
<Button
    android:id="@+id/btProximaCarta"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="16dp"
    android:text="Próxima Carta"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toEndOf="@+id/btRecomecar" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:text="Sua pontuação:"
    android:textSize="18sp"
    app:layout_constraintEnd_toStartOf="@+id/tvPontuacao"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/tvPontuacao"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:text="21"
    android:textColor="@android:color/holo_blue_dark"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/btRecomecar"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="16dp"
    android:text="Recomeçar o Jogo"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/btProximaCarta"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Código Fonte 2.18 – Layout do jogo 21
Fonte: Elaborado pelo autor (2020)

Abra o arquivo **MainActivity.kt** e adicione as variáveis que irão realizar o bind com os componentes do layout **activity_main.xml:**

```
class MainActivity : AppCompatActivity() {

   private lateinit var btRecomecar: Button
   private lateinit var btProximaCarta: Button
   private lateinit var tvPontuacao: TextView
   private lateinit var ivCarta: ImageView

   override fun onCreate(savedInstanceState: Bundle?) {

   }
}
```

Código-fonte 2.19 – Declaração das views do jogo 21
Fonte: Elaborado pelo autor (2020)

Crie um método que será responsável por gerar o baralho com as cartas do jogo:

```
private fun getBaralho() :  MutableList<Carta> {

   return mutableListOf<Carta>(
      Carta(R.drawable.as_de_espada, 1),
      Carta(R.drawable.dois_de_espada, 2),
      Carta(R.drawable.tres_de_espada, 3),
      Carta(R.drawable.quatro_de_espada, 4),
      Carta(R.drawable.cinco_de_espada, 5),
      Carta(R.drawable.seis_de_espada, 6),
      Carta(R.drawable.sete_de_espada, 7),
      Carta(R.drawable.oito_de_espada, 8),
      Carta(R.drawable.nove_de_espada, 9),
      Carta(R.drawable.dez_de_espada, 10),
      Carta(R.drawable.valete_de_espada, 10),
      Carta(R.drawable.dama_de_espada, 10),
      Carta(R.drawable.rei_de_espada, 10)
   )
}
```

Código-fonte 2.20 – Criação do baralho com as cartas e sua pontuação
Fonte: Elaborado pelo autor (2020)

Agora crie um método chamado **setUpView,** e nele serão realizados os binds. Em seguida, execute o método dentro de **onCreate**:

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)

   setUpView()

}

private fun setUpView() {
   btRecomecar = findViewById(R.id.btRecomecar)
   btProximaCarta = findViewById(R.id.btProximaCarta)
   tvPontuacao = findViewById(R.id.tvPontuacao)
   ivCarta = findViewById(R.id.ivCarta)
}
```

Código-fonte 2.21 – Bind das views
Fonte: Elaborado pelo autor (2020)

Para captar o clique do botão, crie um método chamado **setListeners** e o execute dentro do método **onCreate:**

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)

   setUpView()
   setListeners()

}

private fun setListeners() {
   btProximaCarta.setOnClickListener {

   }

   btRecomecar.setOnClickListener {

   }
}
```

Código-fonte 2.22 – Declaração dos listeners
Fonte: Elaborado pelo autor (2020)

Ao clicar no botão Recomeçar (btRecomecar), o jogo será reiniciado. Para isso, crie uma variável chamada **cartas,** de forma global, e em seguida, crie um método

chamado **iniciarPartida**. Execute o método dentro do setOnClickListener do **btRecomercar**. Obs: Adicione somente o que está em negrito:

```
class MainActivity : AppCompatActivity() {

    private lateinit var btRecomecar: Button
    private lateinit var btProximaCarta: Button
    private lateinit var tvPontuacao: TextView
    private lateinit var ivCarta: ImageView

    private var cartas: MutableList<Carta> = mutableListOf()
    private val gerador = Random()

    private fun getBaralho(): MutableList<Carta> {...}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setUpView()
        setListeners()

        iniciarPartida()
    }

    private fun setUpView() {
        btRecomecar = findViewById(R.id.btRecomecar)
        btProximaCarta = findViewById(R.id.btProximaCarta)
        tvPontuacao = findViewById(R.id.tvPontuacao)
        ivCarta = findViewById(R.id.ivCarta)
    }

    private fun setListeners() {
        btProximaCarta.setOnClickListener {

        }

        btRecomecar.setOnClickListener {
            iniciarPartida()
        }
    }

    private fun iniciarPartida() {
        tvPontuacao.text = "0"
        cartas = getBaralho()
        ivCarta.setImageDrawable(ContextCompat.getDrawable(this,
R.drawable.logo))
    }
```

```
}
```

Código-fonte 2.23 – Criação do método para iniciar a partida
Fonte: Elaborado pelo autor (2020)

O próximo passo será programar a jogada ao clicar no botão próxima carta. Adicione o método chamado realizar jogada, executando-o dentro do setOnClickListener do botão próxima carta (btProximaCarta):

```
private fun setListeners() {
   btProximaCarta.setOnClickListener {
      realizarJogada()
   }

   btRecomecar.setOnClickListener {
      iniciarPartida()
   }
}

private fun realizarJogada() {

   val posicaoCartaSelecionada = gerador.nextInt(cartas.size)
   val cartaSelecionada = cartas.get(posicaoCartaSelecionada)

   val pontuacaoAtualizada = tvPontuacao.text.toString().toInt() +
cartaSelecionada.pontuacao

   tvPontuacao.text = pontuacaoAtualizada.toString()

   if (pontuacaoAtualizada > 21) {
      iniciarPartida()
   } else {
      cartas.removeAt(posicaoCartaSelecionada)
      ivCarta.setImageDrawable(
         ContextCompat.getDrawable(
            this,
            cartaSelecionada.resourceId
         )
      )
   }
}
```

Código-fonte 2.24 – Método para realizar a jogada
Fonte: Elaborado pelo autor (2020)

Para finalizar, iremos exibir os Toasts Customizados, criados na nossa biblioteca de componentes. Crie um método chamado **exibirMensagem** com o

seguinte código, e o execute dentro do método realizar jogada. Para que o usuário não fique clicando antes do término da exibição das mensagens, iremos adicionar o método para controle dos botões:

```
private fun exibeMensagem(pontuacao: Int) {
  when {
     pontuacao == 21 -> {
        CustomToast.success(this, "Você atingiu a melhor pontuação. Hora de
parar :)")
     }
     pontuacao > 21 -> {
        CustomToast.error(this, "Você perdeu fez $pontuacao e perdeu")
     }
     pontuacao > 11 -> {
        CustomToast.warning(
           this,
           "Cuidado, dependendo da carta que comprar você poderá perder"
        )
     }
     else -> {
        CustomToast.info(this, "Você ainda pode jogar com segurança")
     }
  }
  controlarBotoes()
}

private fun controlarBotoes() {
  val time = Toast.LENGTH_LONG
  habilitarBotoes(false)
  Handler().postDelayed({
     habilitarBotoes(true)
  }, 2500)
}

private fun habilitarBotoes(habilitar: Boolean) {
  btProximaCarta.isEnabled = habilitar
  btRecomecar.isEnabled = habilitar
}


private fun realizarJogada() {

  val posicaoCartaSelecionada = gerador.nextInt(cartas.size)
  val cartaSelecionada = cartas.get(posicaoCartaSelecionada)

  val pontuacaoAtualizada = tvPontuacao.text.toString().toInt() +
cartaSelecionada.pontuacao
```

```
    tvPontuacao.text = pontuacaoAtualizada.toString()

    exibeMensagem(pontuacaoAtualizada)

    if (pontuacaoAtualizada > 21) {
        iniciarPartida()
    } else {
        cartas.removeAt(posicaoCartaSelecionada)
        ivCarta.setImageDrawable(
            ContextCompat.getDrawable(
                this,
                cartaSelecionada.resourceId
            )
        )
    }
}
```

Código-fonte 2.25 – Método para exibir mensagem de ajuda
Fonte: Elaborado pelo autor (2020)

A seguir, temos o código completo da classe criada:

```
class MainActivity : AppCompatActivity() {

    private lateinit var btRecomecar: Button
    private lateinit var btProximaCarta: Button
    private lateinit var tvPontuacao: TextView
    private lateinit var ivCarta: ImageView

    private var cartas: MutableList<Carta> = mutableListOf()
    private val gerador = Random()

    private fun getBaralho(): MutableList<Carta> {

        return mutableListOf<Carta>(
            Carta(R.drawable.as_de_espada, 1),
            Carta(R.drawable.dois_de_espada, 2),
            Carta(R.drawable.tres_de_espada, 3),
            Carta(R.drawable.quatro_de_espada, 4),
            Carta(R.drawable.cinco_de_espada, 5),
            Carta(R.drawable.seis_de_espada, 6),
            Carta(R.drawable.sete_de_espada, 7),
            Carta(R.drawable.oito_de_espada, 8),
            Carta(R.drawable.nove_de_espada, 9),
            Carta(R.drawable.dez_de_espada, 10),
            Carta(R.drawable.valete_de_espada, 10),
            Carta(R.drawable.dama_de_espada, 10),
            Carta(R.drawable.rei_de_espada, 10)
```

```
        )
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setUpView()
        setListeners()

        iniciarPartida()
    }

    private fun setUpView() {
        btRecomecar = findViewById(R.id.btRecomecar)
        btProximaCarta = findViewById(R.id.btProximaCarta)
        tvPontuacao = findViewById(R.id.tvPontuacao)
        ivCarta = findViewById(R.id.ivCarta)
    }

    private fun setListeners() {
        btProximaCarta.setOnClickListener {
            realizarJogada()
        }

        btRecomecar.setOnClickListener {
            iniciarPartida()
        }
    }


    private fun exibeMensagem(pontuacao: Int) {
        when {
            pontuacao == 21 -> {
                CustomToast.success(this, "Você atingiu a melhor pontuação. Hora
de parar :)")
            }
            pontuacao > 21 -> {
                CustomToast.error(this, "Você perdeu fez $pontuacao e perdeu")
            }
            pontuacao > 11 -> {
                CustomToast.warning(
                    this,
                    "Cuidado, dependendo da carta que comprar você poderá perder"
                )
            }
            else -> {
                CustomToast.info(this, "Você ainda pode jogar com segurança")
            }
```

```
        }
        controlarBotoes()
    }

    private fun controlarBotoes() {
        val time = Toast.LENGTH_LONG
        habilitarBotoes(false)
        Handler().postDelayed({
            habilitarBotoes(true)
        }, 2500)
    }

    private fun habilitarBotoes(habilitar: Boolean) {
        btProximaCarta.isEnabled = habilitar
        btRecomecar.isEnabled = habilitar
    }


    private fun realizarJogada() {

        val posicaoCartaSelecionada = gerador.nextInt(cartas.size)
        val cartaSelecionada = cartas.get(posicaoCartaSelecionada)

        val pontuacaoAtualizada = tvPontuacao.text.toString().toInt() +
cartaSelecionada.pontuacao

        tvPontuacao.text = pontuacaoAtualizada.toString()

        exibeMensagem(pontuacaoAtualizada)

        if (pontuacaoAtualizada > 21) {
            iniciarPartida()
        } else {
            cartas.removeAt(posicaoCartaSelecionada)
            ivCarta.setImageDrawable(
                ContextCompat.getDrawable(
                    this,
                    cartaSelecionada.resourceId
                )
            )
        }
    }

    private fun iniciarPartida() {
        tvPontuacao.text = "0"
        cartas = getBaralho()
        ivCarta.setImageDrawable(ContextCompat.getDrawable(this,
R.drawable.logo))
    }
```

```
}
```

Código-fonte 2.26 – Código completo do jogo 21 com as mensagens
Fonte: Elaborado pelo autor (2020)

2.7.7 Product Flavors e seus builds variants

Um poderoso recurso que o Android possui é a capacidade de gerar múltiplas versões customizadas de uma única aplicação, usando o mesmo projeto. Um mesmo aplicativo pode ter diferentes versões, e a mudança entre eles pode ser visual (ícones, cores, textos…) e/ou por funcionalidade. Esse mecanismo é chamado de **Product Flavors**, cuja ideia é que você tenha vários "sabores" do mesmo aplicativo.

Com flavors reduzimos: a necessidade de duplicar código, os múltiplos versionamentos, os possíveis erros no famoso copy/paste de códigos e os recursos.

Normalmente esse recurso é utilizado quando:

- o aplicativo tem a versão paga e a versão gratuita (com menos funcionalidades).
- precisa lançar uma versão experimental antes da oficial.
- há produtos white label, com os quais é possível trocar cor, nome, ícone, fluxo ou novas funcionalidades, de acordo com o cliente.

Para conseguir atingir os objetivos acima, é preciso definir as variações do aplicativo com base no *build type* e no *product flavor*, compondo assim um build variant.

2.7.7.1 Build type

Define propriedades utilizadas pelo Gradle ao compilar e empacotar a aplicação. Normalmente é utilizado em diferentes estágios do ciclo de desenvolvimento. Por padrão, um projeto Android possui dois builds types: debug (utilizado durante o desenvolvimento do app, com o qual podemos utilizar a depuração

do código) e o release (utilizado na publicação do aplicativo, em que ele é assinado com chave diferente e depuração desabilitada).

2.7.7.2 Product Flavor

Representa as diferentes versões do mesmo aplicativo, por exemplo: versão gratuita e paga, versão de desenvolvimento, homologação ou produção. Eles são opcionais e criados explicitamente nos projetos. Por padrão, os projetos Android possuem um flavor chamado de main.

2.7.7.3 Build Variants

É a combinação entre o build type e o product flavors. Permite a geração de múltiplos APKs com diferentes configurações para o mesmo projeto. Por exemplo, se o projeto tiver dois flavors "gratuito" e "pago", o projeto terá ao menos os seguintes builds variants: debugGratuito, releaseGratuito, debugPago, releasePago. Para a compilação do projeto, o Gradle utiliza uma combinação do código disponível em diferentes diretórios.

2.7.7.4 Aplicando Flavors na prática

Para utilizar os flavors na prática, abra o arquivo build.gradle (app) e adicione product flavors, conforme o código abaixo:

```
android {
  compileSdkVersion 29
  buildToolsVersion "29.0.3"

  defaultConfig {
    applicationId "br.com.heiderlopes.game21"
    minSdkVersion 21
    targetSdkVersion 29
    versionCode 1
    versionName "1.0"
```

```
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    flavorDimensions "app"

    productFlavors {
        pago {
            dimension "app"

        }

        gratuito {
            dimension "app"
            applicationIdSuffix ".gratuito"
            versionNameSuffix "-gratuito"
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }

}
```

Código-fonte 2.27 – Criação dos flavors
Fonte: Elaborado pelo autor (2020)

Pode ser definido um sufixo para o id da aplicação por meio do applicationIdSuffix. Assim, é possível instalar os dois flavors no mesmo dispositivo. Com o uso do versionNameSuffix é possível alterar o nome de acordo com a versão.

Altere a visão do projeto de Android para Project:

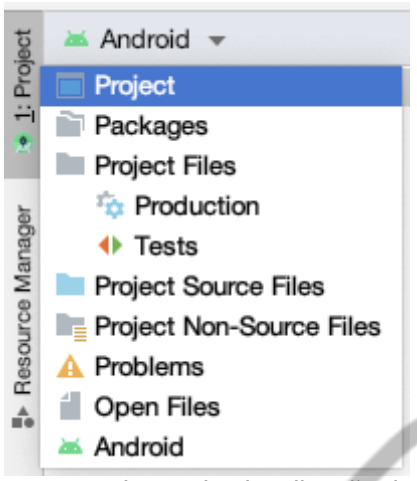


Figura 2.47 – Mudança da visualização do projeto
Fonte: Elaborado pelo autor (2020)

Crie um diretório chamado **gratuito** (**app** → **src**) dentro uma pasta chamada **res**:

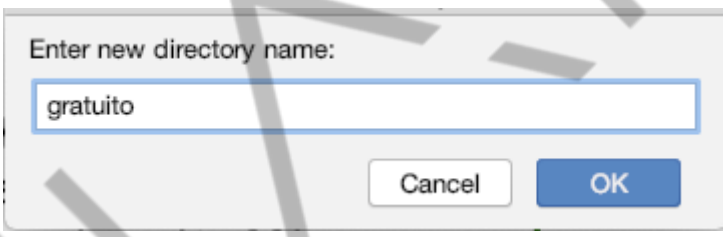


Figura 2.48 – Criação do pacote gratuito
Fonte: Elaborado pelo autor (2020)

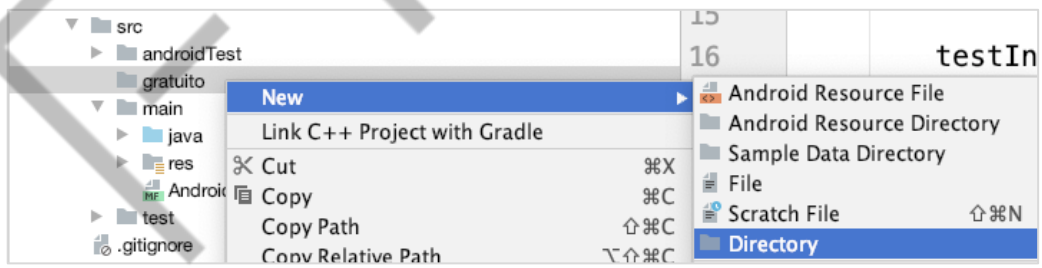


Figura 2.49 – Criação de um novo diretório
Fonte: Elaborado pelo autor (2020)

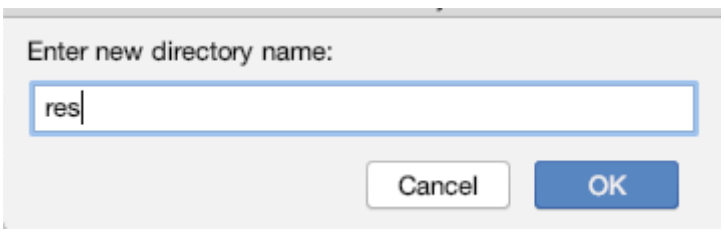


Figura 2.50 – Criação de um novo diretório para armazenar resources
Fonte: Elaborado pelo autor (2020)

Dentro desse diretório, é possível colocar os recursos referentes à versão gratuita do aplicativo, por exemplo: *strings*, cores, ícones, entre outros. Dessa mesma maneira, é possível adicionar código java/kotlin e funcionalidades diferentes conforme o flavor.

Neste exemplo, o ícone será alterado de acordo com a versão. Copie os ícones do mipmap (disponíveis em: https://github.com/FIAPON/AndroidJogo21 /tree/master/mipmap_gratuito) dentro do diretório criado:

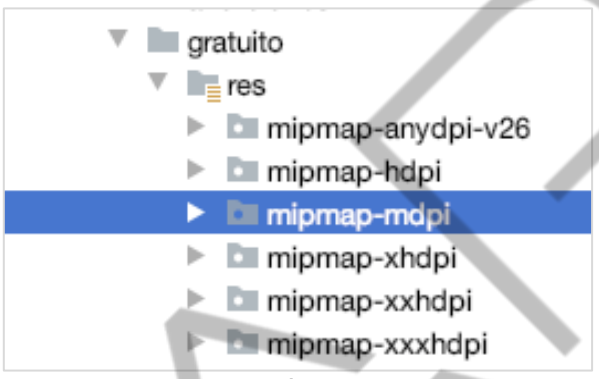


Figura 2.51 – Adição dos ícones do aplicativo gratuito
Fonte: Elaborado pelo autor (2020)

Faça o mesmo para o padrão. Os ícones estão disponíveis em: https://github.com/FIAPON/AndroidJogo21 /tree/master/mipmap_default:

Figura 2.52 – Adição dos ícones do aplicativo pago
Fonte: Elaborado pelo autor (2020)

Para alterar o **build variant**, selecione a aba **Build Variants** no Android Studio e escolha a opção desejada:

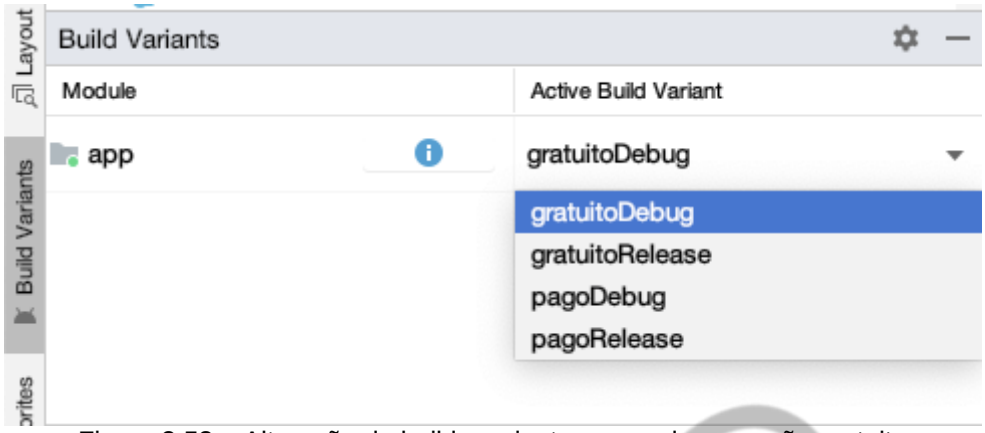


Figura 2.53 – Alteração de builds variants para rodar a versão gratuita
Fonte: Elaborado pelo autor (2020)

Após selecionar, coloque o aplicativo para rodar.

Altere o flavor para outra versão e execute novamente:

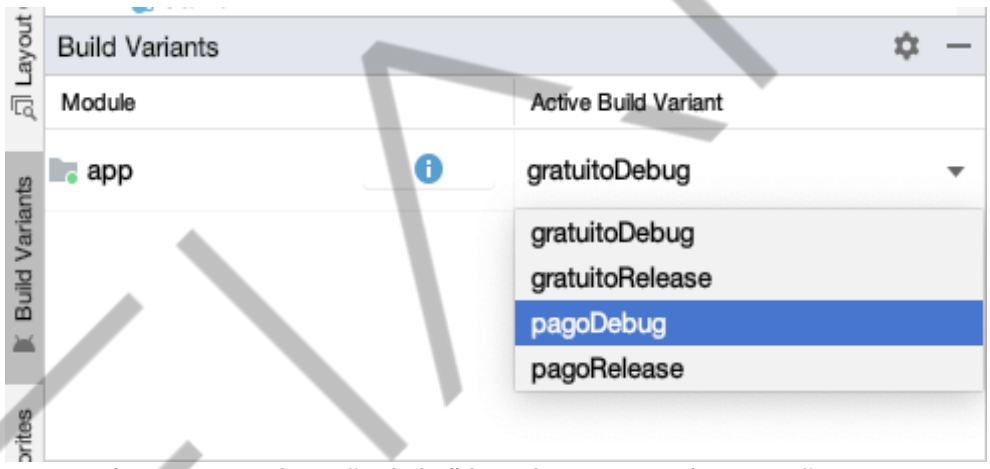


Figura 2.54 – Alteração de builds variants para rodar a versão paga
Fonte: Elaborado pelo autor (2020)

Observe que os dois aplicativos estão instalados:

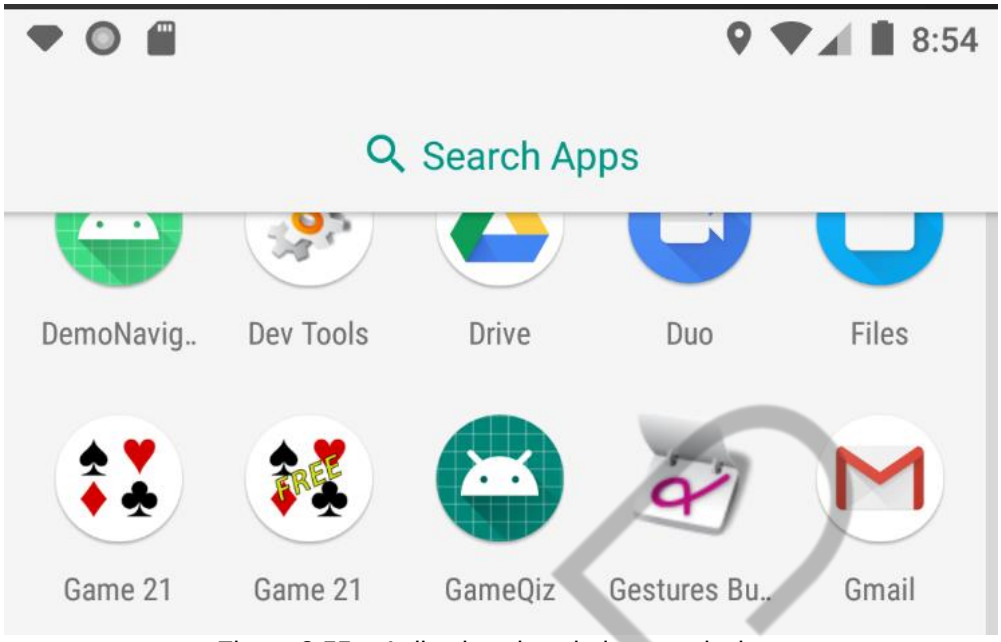


Figura 2.55 – Aplicativos instalados nos devices
Fonte: Elaborado pelo autor (2020)

Para o próximo exemplo, será adicionada uma propaganda na versão gratuita.

Existem várias formas de realizar essa ação, uma abordagem seria criar o pacote dentro do flavor, conforme feito nos ícones. Porém será feito por outra abordagem, verificando a versão do build.

Primeiro será alterado o layout para adicionar o banner:

```
<androidx.constraintlayout.widget.ConstraintLayout
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:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:layout_editor_absoluteY="81dp">

  <ImageView
    android:id="@+id/ivCarta"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="32dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="32dp"
    android:layout_marginBottom="32dp"
    app:layout_constraintBottom_toTopOf="@+id/btProximaCarta"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
```

```
    tools:src="@drawable/as_de_espada" />

<Button
    android:id="@+id/btProximaCarta"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="8dp"
    android:text="Próxima Carta"
    app:layout_constraintBottom_toTopOf="@+id/containerPropaganda"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toEndOf="@+id/btRecomecar" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:text="Sua pontuação:"
    android:textSize="18sp"
    app:layout_constraintEnd_toStartOf="@+id/tvPontuacao"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/tvPontuacao"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:text="21"
    android:textColor="@android:color/holo_blue_dark"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/btRecomecar"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="8dp"
    android:text="Recomeçar o Jogo"
    app:layout_constraintBottom_toTopOf="@+id/containerPropaganda"
    app:layout_constraintEnd_toStartOf="@+id/btProximaCarta"
```

```
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent" />


<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/containerPropaganda"
    android:background="#EEE"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent">

    <TextView
        android:id="@+id/tvAcaoPropaganda"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_marginEnd="16dp"
        android:text="Saiba Mais"
        android:gravity="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ImageView
        android:id="@+id/ivPropaganda"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:layout_marginStart="12dp"
        android:layout_marginTop="12dp"
        android:layout_marginBottom="12dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/logo" />

    <TextView
        android:id="@+id/tvTituloPropaganda"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:text="Anuncie aqui"
        android:textSize="16sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toStartOf="@+id/tvAcaoPropaganda"
        app:layout_constraintStart_toEndOf="@+id/ivPropaganda"
        app:layout_constraintTop_toTopOf="@+id/ivPropaganda" />
```

```
    <TextView
        android:id="@+id/tvMensagemPropaganda"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Me pergunte como e irei te mostrar"
        app:layout_constraintEnd_toEndOf="@+id/tvTituloPropaganda"
        app:layout_constraintStart_toStartOf="@+id/tvTituloPropaganda"
        app:layout_constraintTop_toBottomOf="@+id/tvTituloPropaganda" />
</androidx.constraintlayout.widget.ConstraintLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Código-fonte 2.28 - Layout com banner
Fonte: Elaborado pelo autor (2020)

Em seguida, deve-se alterar o código para exibir ou não a propaganda, de acordo com a versão. Adicione o código em negrito abaixo:

```
class MainActivity : AppCompatActivity() {

    private lateinit var btRecomecar: Button
    private lateinit var btProximaCarta: Button
    private lateinit var tvPontuacao: TextView
    private lateinit var ivCarta: ImageView

    private lateinit var containerPropaganda: ConstraintLayout

    private var cartas: MutableList<Carta> = mutableListOf()
    private val gerador = Random()

    private fun getBaralho(): MutableList<Carta> {
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setUpView()
        setListeners()

        iniciarPartida()

        showBanner()
    }

    private fun showBanner() {
        containerPropaganda.visibility = if (isGratuito()) View.VISIBLE else
```

```
View.GONE
  }

  private fun isGratuito(): Boolean {
    return packageName == "br.com.heiderlopes.game21.gratuito"
  }

  private fun setUpView() {
    btRecomecar = findViewById(R.id.btRecomecar)
    btProximaCarta = findViewById(R.id.btProximaCarta)
    tvPontuacao = findViewById(R.id.tvPontuacao)
    ivCarta = findViewById(R.id.ivCarta)
    containerPropaganda = findViewById(R.id.containerPropaganda)
  }
  //Restante do código
}
```

Código-fonte 2.29 – Lógica para exibir o banner de acordo com a versão do aplicativo
Fonte: Elaborado pelo autor (2020)

Segue abaixo o código completo da classe MainActivity:

```
class MainActivity : AppCompatActivity() {

  private lateinit var btRecomecar: Button
  private lateinit var btProximaCarta: Button
  private lateinit var tvPontuacao: TextView
  private lateinit var ivCarta: ImageView

  private lateinit var containerPropaganda: ConstraintLayout

  private var cartas: MutableList<Carta> = mutableListOf()
  private val gerador = Random()

  private fun getBaralho(): MutableList<Carta> {

    return mutableListOf<Carta>(
      Carta(R.drawable.as_de_espada, 1),
      Carta(R.drawable.dois_de_espada, 2),
      Carta(R.drawable.tres_de_espada, 3),
      Carta(R.drawable.quatro_de_espada, 4),
      Carta(R.drawable.cinco_de_espada, 5),
      Carta(R.drawable.seis_de_espada, 6),
      Carta(R.drawable.sete_de_espada, 7),
      Carta(R.drawable.oito_de_espada, 8),
      Carta(R.drawable.nove_de_espada, 9),
      Carta(R.drawable.dez_de_espada, 10),
      Carta(R.drawable.valete_de_espada, 10),
      Carta(R.drawable.dama_de_espada, 10),
```

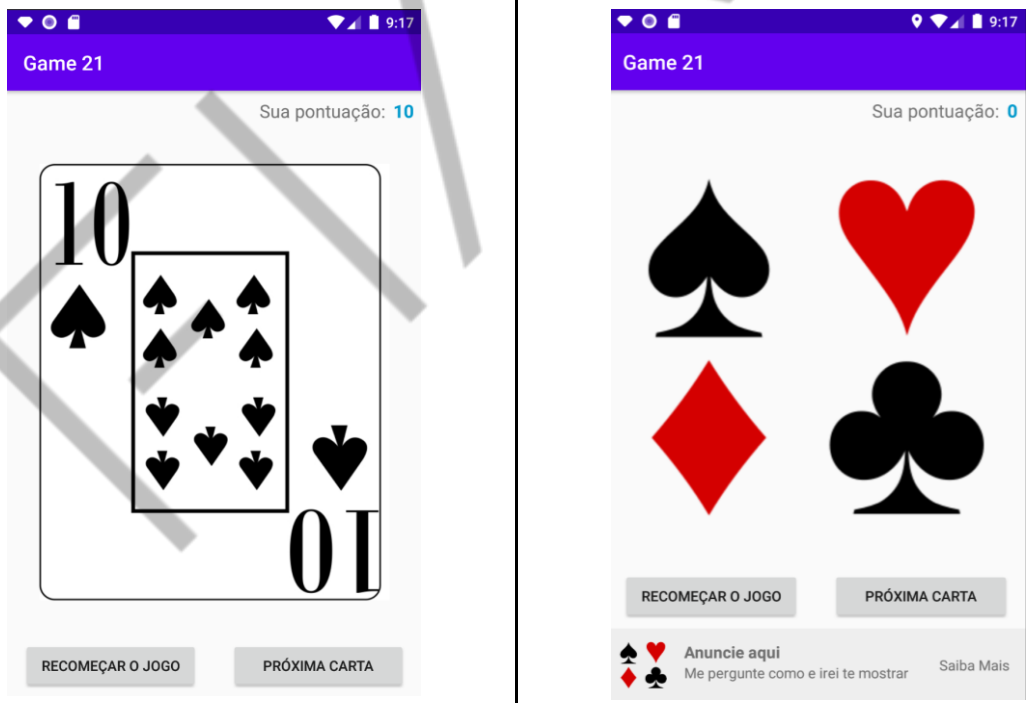
```
            Carta(R.drawable.rei_de_espada, 10)
        )
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setUpView()
        setListeners()

        iniciarPartida()

        showBanner()
    }

    private fun showBanner() {
        containerPropaganda.visibility = if (isGratuito()) View.VISIBLE else
View.GONE
    }

    private fun isGratuito(): Boolean {
        return packageName == "br.com.heiderlopes.game21.gratuito"
    }

    private fun setUpView() {
        btRecomecar = findViewById(R.id.btRecomecar)
        btProximaCarta = findViewById(R.id.btProximaCarta)
        tvPontuacao = findViewById(R.id.tvPontuacao)
        ivCarta = findViewById(R.id.ivCarta)
        containerPropaganda = findViewById(R.id.containerPropaganda)
    }

    private fun setListeners() {
        btProximaCarta.setOnClickListener {
            realizarJogada()
        }

        btRecomecar.setOnClickListener {
            iniciarPartida()
        }
    }


    private fun exibeMensagem(pontuacao: Int) {
        when {
            pontuacao == 21 -> {
                CustomToast.success(this, "Você atingiu a melhor pontuação. Hora de
parar :)")
```

```
            }
            pontuacao > 21 -> {
                CustomToast.error(this, "Você perdeu fez $pontuacao e perdeu")
            }
            pontuacao > 11 -> {
                CustomToast.warning(
                    this,
                    "Cuidado, dependendo da carta que comprar você poderá perder"
                )
            }
            else -> {
                CustomToast.info(this, "Você ainda pode jogar com segurança")
            }
        }
        controlarBotoes()
    }

    private fun controlarBotoes() {
        val time = Toast.LENGTH_LONG
        habilitarBotoes(false)
        Handler().postDelayed({
            habilitarBotoes(true)
        }, 2500)
    }

    private fun habilitarBotoes(habilitar: Boolean) {
        btProximaCarta.isEnabled = habilitar
        btRecomecar.isEnabled = habilitar
    }


    private fun realizarJogada() {

        val posicaoCartaSelecionada = gerador.nextInt(cartas.size)
        val cartaSelecionada = cartas[posicaoCartaSelecionada]

        val pontuacaoAtualizada = tvPontuacao.text.toString().toInt() +
cartaSelecionada.pontuacao

        tvPontuacao.text = pontuacaoAtualizada.toString()

        exibeMensagem(pontuacaoAtualizada)

        if (pontuacaoAtualizada > 21) {
            iniciarPartida()
        } else {
            cartas.removeAt(posicaoCartaSelecionada)
            ivCarta.setImageDrawable(
                ContextCompat.getDrawable(
```

```
                this,
                cartaSelecionada.resourceId
            )
        )
    }
  }

  private fun iniciarPartida() {
      tvPontuacao.text = "0"
      cartas = getBaralho()
      ivCarta.setImageDrawable(ContextCompat.getDrawable(this,
R.drawable.logo))
  }
}
```

Código-fonte 2.30 – Jogo completo
Fonte: Elaborado pelo autor (2020)

Tem-se, portanto, o resultado:



Versão Paga	Versão Gratuita

Figura 2.56 – Resultado dos aplicativos
Fonte: Elaborado pelo autor (2020)

CONCLUSÃO

Neste capítulo, foi possível conhecer o Gradle e a possibilidade que ele oferece de: gerenciar as dependências do projeto, gerar versões diferentes (Product Flavors) e assinar aplicativos para melhorar ainda mais a vida do desenvolvedor.

Observou-se que, além de usar o Gradle para configurar dependências específicas dos projetos, também foi possível aprender como criar um pacote que seja utilizado por outras aplicações, com o uso do JitPack.

E que esses recursos são úteis, quando se trabalha com reutilização e padronização de componentes.

REFERÊNCIAS

LEAL, N. G. de V. **Dominando o Android com Kotlin**. 3. ed. São Paulo: Novatec, 2019.