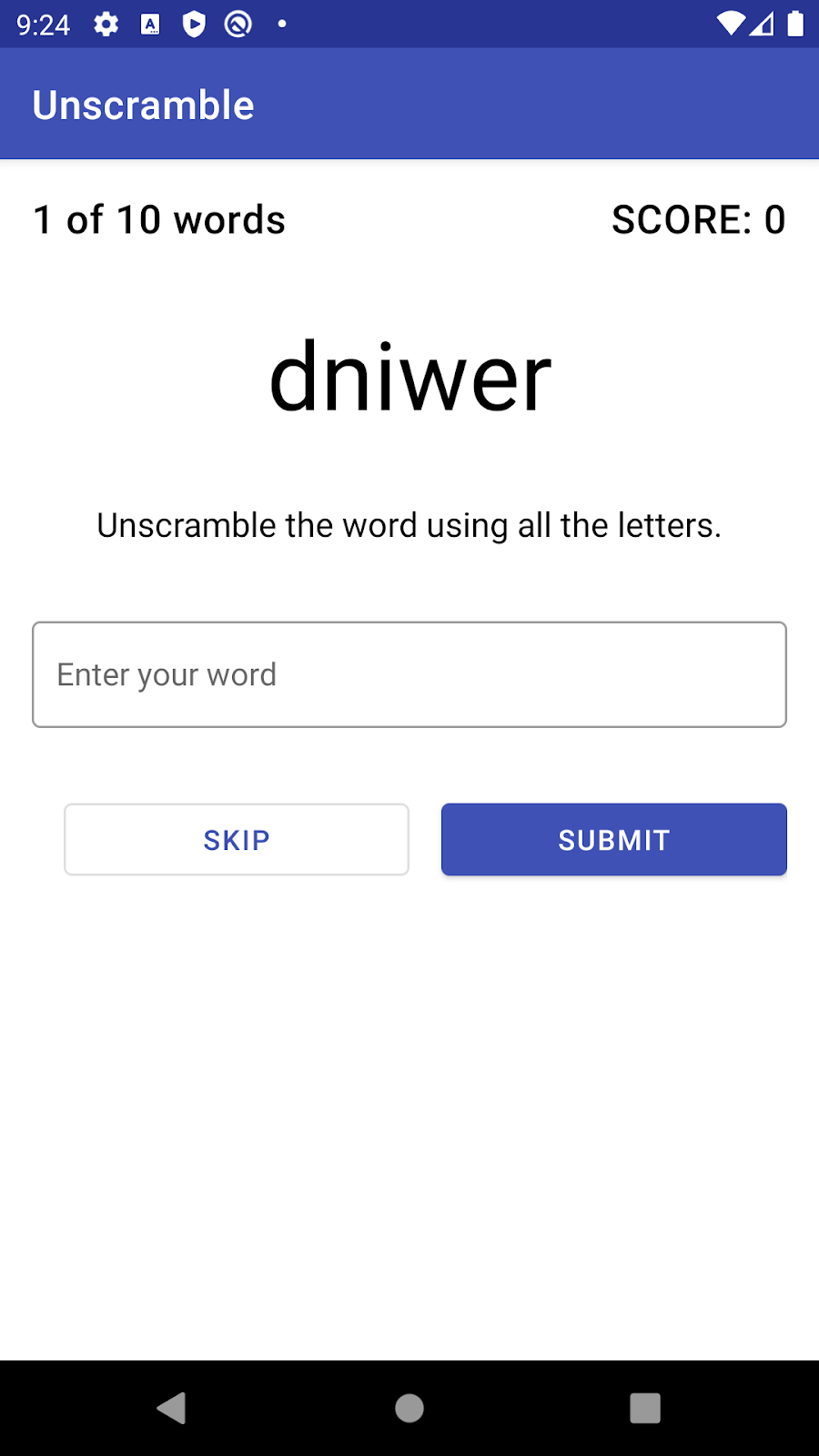
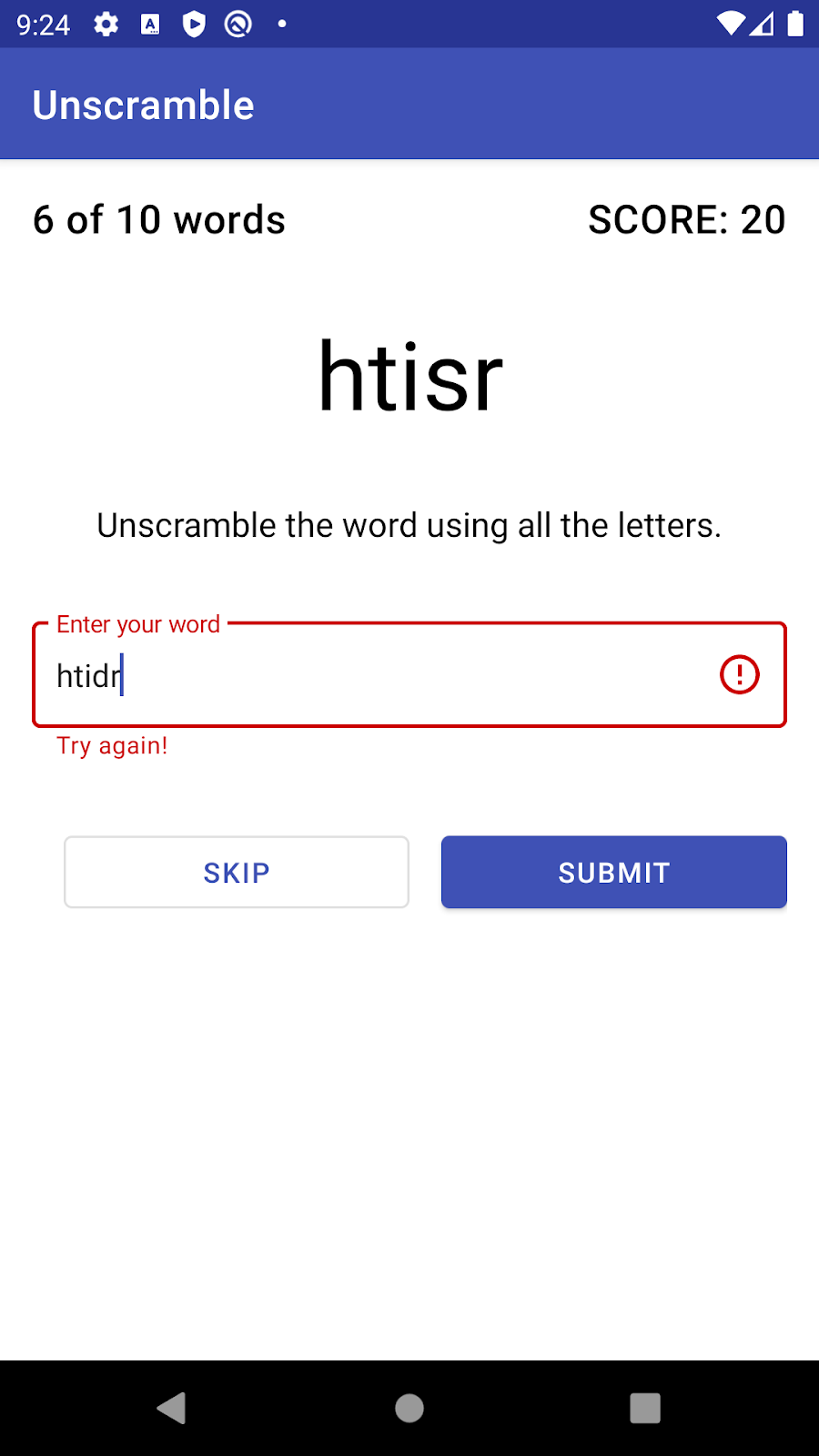
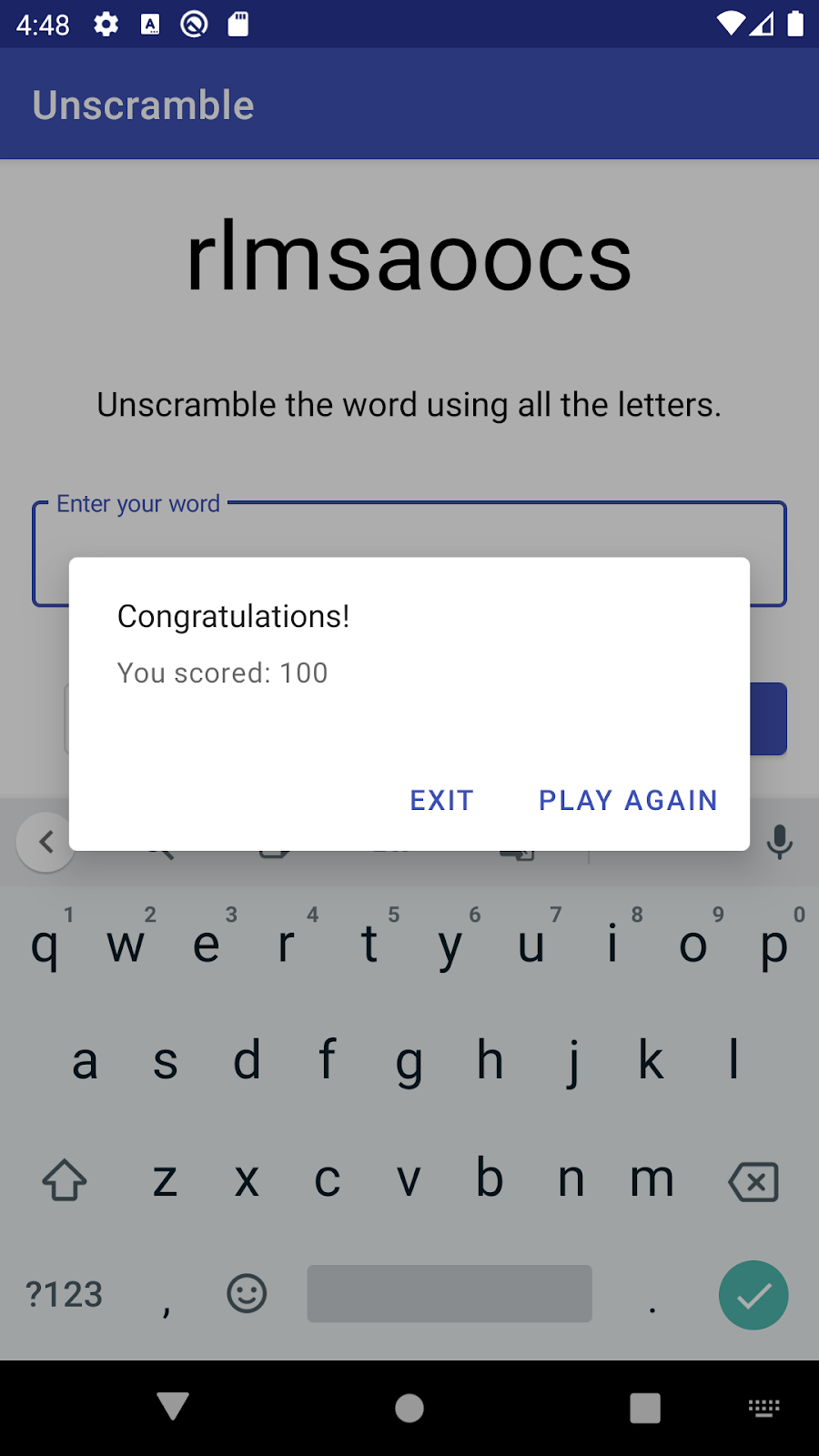
## Visão geral do jogo

O app Unscramble é um jogo de palavras embaralhadas para um só jogador. O app exibe uma palavra embaralhada por vez, e o jogador precisa adivinhá-la usando todas as letras disponíveis. O jogador marcará pontos se a palavra estiver correta. Caso contrário, o jogador poderá tentar quantas vezes quiser. O app também tem a opção de pular a palavra atual. No canto superior esquerdo, o app exibe a contagem de palavras, que é o número de palavras jogadas na sessão atual. Cada partida tem 10 palavras.

### **Fazer o download do código inicial**

Este codelab oferece um código inicial para você se aprofundar nos recursos ensinados. O código inicial pode conter tanto um código familiar quanto desconhecido de codelabs anteriores. Você aprenderá mais sobre códigos desconhecidos nos próximos codelabs.

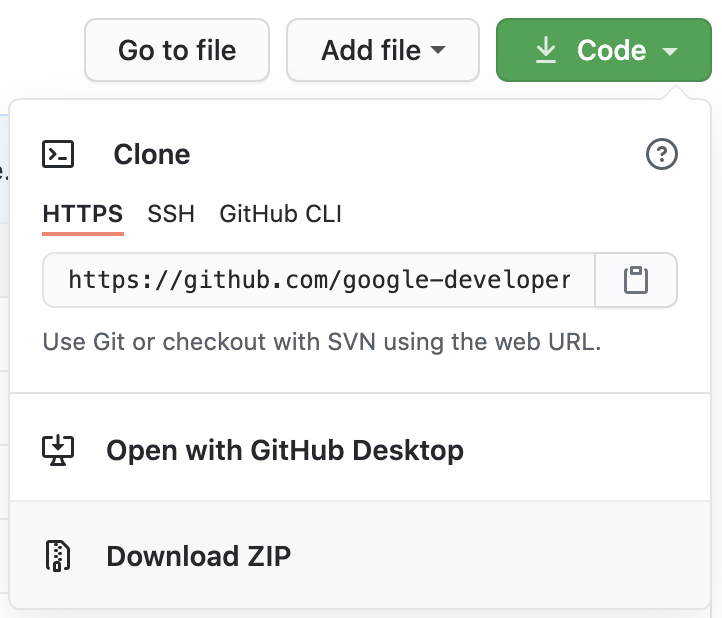
Se você estiver usando o código inicial do GitHub, o nome da pasta é android-basics-kotlin-unscramble-app-starter. Selecione essa pasta ao abrir o projeto no Android Studio.

**URL do código inicial**: <https://github.com/google-developer-training/android-basics-kotlin-unscramble-app/tree/starter> (link em inglês)

Para encontrar o código deste codelab e abri-lo no Android Studio, faça o seguinte.

## Buscar o código

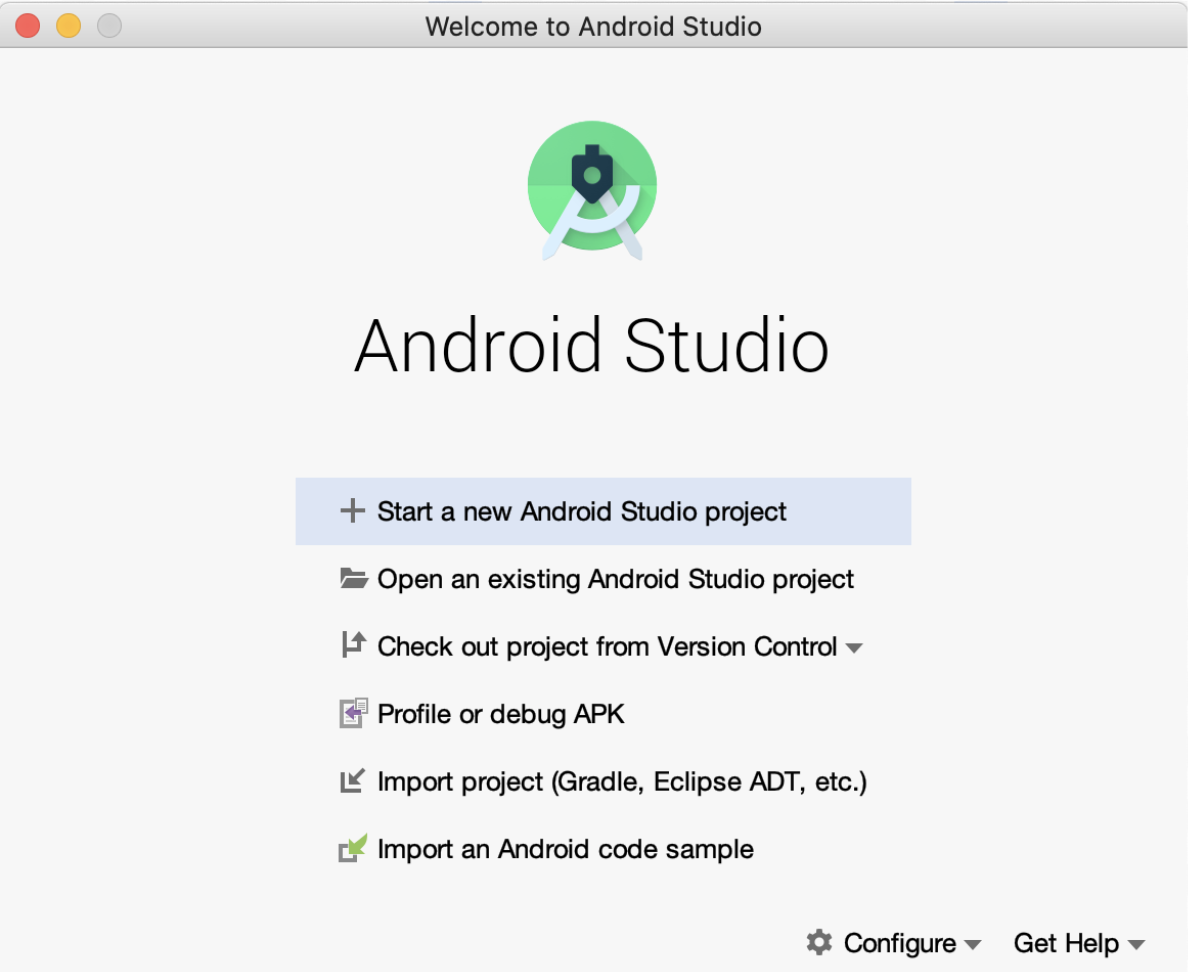
1. Clique no URL fornecido. Isso abrirá a página do GitHub referente ao projeto em um navegador.
2. Na página do GitHub do projeto, clique no botão **Code**, que exibirá uma caixa de diálogo.



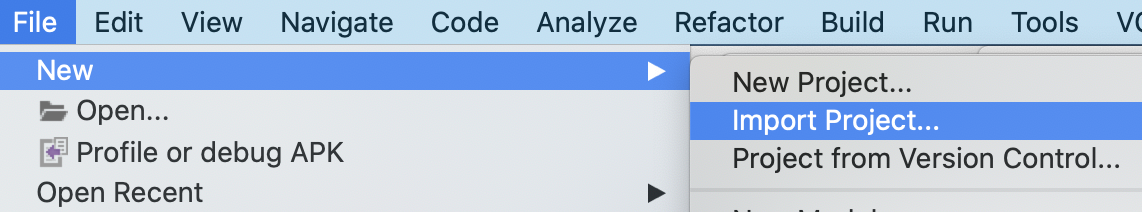
1. Na caixa de diálogo, clique no botão **Download ZIP** para salvar o projeto no seu computador. Aguarde a conclusão do download.
2. Localize o arquivo no computador, que provavelmente está na pasta **Downloads**.
3. Clique duas vezes no arquivo ZIP para descompactá-lo. Isso criará uma nova pasta com os arquivos do projeto.

## Abrir o projeto no Android Studio

1. Inicie o Android Studio.
2. Na janela **Welcome to Android Studio**, clique em **Open an existing Android Studio project**.



Observação: caso o Android Studio já esteja aberto, selecione a opção **File** > **New** > **Import Project**.



1. Na caixa de diálogo **Import Project**, vá até a pasta do projeto descompactada, que provavelmente está na pasta **Downloads**.
2. Clique duas vezes nessa pasta do projeto.
3. Aguarde o Android Studio abrir o projeto.
4. Clique no botão **Run** j7ptomO2PEQNe8jFt4nKCOw_Oc_Aucgf4l_La8fGLCMLy0t9RN9SkmBFGOFjkEzlX4ce2w2NWq4J30sDaxEe4MaSNuJPpMgHxnsRYoBtIV3-GUpYYcIvRJ2HrqR27XGuTS4F7lKCzg para criar e executar o app. Confira se ele funciona da forma esperada.
5. Procure os arquivos do projeto na janela de ferramentas **Project** para ver como o app foi implementado.

## Visão geral do código inicial

1. Abra o projeto com o código inicial no Android Studio.
2. Execute o app em um dispositivo Android ou em um emulador.
3. Jogue uma partida com algumas palavras, tocando nos botões **Submit** e **Skip**. Tocar nos botões exibe a próxima palavra e aumenta a contagem de palavras.
4. A pontuação aumenta apenas ao tocar no botão **Submit**.

## Problemas com o código inicial

Enquanto jogava, é possível que você tenha percebido os seguintes bugs:

1. Ao clicar no botão **Submit**, o app não verifica a palavra do jogador. O jogador sempre marca pontos.
2. Não há como encerrar o jogo. O app permite que você jogue mais de 10 palavras.
3. A tela do jogo mostra uma palavra embaralhada, a pontuação dos jogadores e a contagem de palavras. Gire o dispositivo ou emulador para mudar a orientação da tela. A palavra, a pontuação e a contagem de palavras atuais são perdidas e o jogo reinicia do início.

### **Principais problemas no app**

O app inicial não salva e restaura o estado e os dados dele durante mudanças na configuração, como quando a orientação do dispositivo muda.

Resolva esse problema usando o callback [onSaveInstanceState()](https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt-br" \l "save-simple-lightweight-ui-state-using-onsaveinstancestate" \t "_blank). No entanto, o uso do método onSaveInstanceState() exige que você escreva um código extra para salvar o estado em um pacote e implementar uma lógica para acessar esse estado. Além disso, a quantidade de dados que pode ser armazenada é mínima.

É possível resolver esses problemas usando os [Componentes da arquitetura do Android](https://developer.android.com/topic/libraries/architecture?hl=pt-br) que você aprenderá neste programa.

## Passo a passo do código inicial

O código inicial que foi transferido por download tem o layout da tela de jogo pré-criado para você. Nesse módulo, você se concentrará em implementar a lógica do jogo. Você usará os Componentes da arquitetura para implementar a arquitetura recomendada do app e resolver os problemas mencionados acima. Veja a seguir um breve tutorial sobre alguns dos arquivos para começar.

### **game\_fragment.xml**

* Abra res/layout/game\_fragment.xml na visualização **Design**.
* Esse arquivo contém o layout da única tela do app, que é a tela do jogo.
* Esse layout contém um campo de texto para a palavra do jogador, além de TextViews para exibir a pontuação e a contagem de palavras. Há também instruções e botões (**Submit** e **Skip**) para jogar.

### **main\_activity.xml**

Define o layout da atividade principal com um único fragmento de jogo.

### **Pasta res/values**

Você já conhece os arquivos de recurso nesta pasta.

* O arquivo colors.xml contém as cores de tema usadas no app.
* O arquivo strings.xml contém todas as strings que o app precisa.
* As pastas themes e styles contêm a personalização da IU para o app

MainActivity.kt

Contém o código padrão gerado pelo modelo para definir a visualização do conteúdo da atividade como main\_activity.xml.

ListOfWords.kt

Esse arquivo contém uma lista das palavras usadas no jogo, além das constantes para o número máximo de palavras por partida e o número de pontos que o jogador marca para cada palavra correta.

**AVISO**: não é uma prática recomendada fixar strings no código. Elas precisam ser armazenadas em um arquivo **strings.xml** para facilitar a localização. Para simplificar o processo e se concentrar nos Componentes da arquitetura, as strings são fixadas no código neste app.

GameFragment.kt

Esse é o único fragmento do app. É nele que a maior parte do jogo acontece:

* As variáveis são definidas para a palavra embaralhada atual (currentScrambledWord), contagem de palavras (currentWordCount) e a pontuação (score).
* A instância do objeto de vinculação com o nome binding e acesso às visualizações game\_fragment está definida.
* A função onCreateView() infla o XML do layout game\_fragment usando o objeto de vinculação.
* A função onViewCreated() configura os listeners de cliques no botão e atualiza a IU.
* O método onSubmitWord() é o listener de clique do botão **Submit**. Essa função exibe a próxima palavra embaralhada, limpa o campo de texto e aumenta a pontuação e a contagem da palavra sem validar a palavra do jogador.
* O método onSkipWord() é o listener de clique do botão **Skip**. Essa função atualiza a IU de forma semelhante ao método onSubmitWord(), mas não aumenta a pontuação.
* getNextScrambledWord() é uma função auxiliar que escolhe uma palavra aleatória da lista de palavras e embaralha as letras dela.
* As funções restartGame() e exitGame() são usadas para reiniciar e finalizar o jogo, respectivamente. Você usará essas funções mais tarde.
* O método resetTextField() limpa o conteúdo do campo de texto e redefine o status do erro.
* A função updateNextWordOnScreen() exibe a nova palavra embaralhada.

## [3. Saiba mais sobre a arquitetura de apps](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#2)

A arquitetura fornece as diretrizes para ajudar você a alocar responsabilidades no app entre as classes. Uma arquitetura de app bem projetada ajuda a escalonar e ampliar o app, adicionando outros recursos no futuro. Isso também facilita a colaboração em equipe.

Os [princípios de arquitetura](https://developer.android.com/jetpack/guide?hl=pt-br#common-principles) mais comuns são a separação de conceitos e o modelo de base da IU.

**Separação de conceitos**

A separação de conceitos é o princípio que determina que o app precisa ser dividido em classes, cada uma com responsabilidades separadas.

**Basear a IU em um modelo**

Outro princípio importante é que você precisa basear sua IU em um modelo, de preferência um que seja persistente. Modelos são componentes responsáveis por manipular os dados de um app. Eles são independentes dos objetos Views e dos componentes do app, portanto, não são afetados pelo ciclo de vida do app nem pelos conceitos associados.

As principais classes ou Componentes da arquitetura do Android são os controladores de IU (atividade/fragmento), o ViewModel, o LiveData e o Room. Esses componentes cuidam de parte da complexidade do ciclo de vida e ajudam a evitar problemas relacionados a ele. Você aprenderá sobre o LiveData e o Room nos próximos codelabs.

Este diagrama mostra uma parte básica da arquitetura:

### 

### **Controladores de IU (atividade / fragmento)**

Atividades e fragmentos são controladores de IU. Os controladores de IU comandam as ações da IU exibindo visualizações na tela, capturando eventos e todas as ações relacionadas à IU com que o usuário interage. Os dados no app ou qualquer lógica de tomada de decisão sobre esses dados não podem estar nas classes de controlador de IU.

O sistema Android pode destruir controladores de IU a qualquer momento com base em determinadas interações do usuário ou condições do sistema, como pouca memória. Como esses eventos não estão sob seu controle, não armazene dados ou estados de app em controladores de IU. Em vez disso, a lógica de tomada de decisão sobre os dados precisa ser adicionada ao ViewModel.

Por exemplo, no app **Unscramble**, a palavra embaralhada, a contagem de palavras e a pontuação são exibidas em um fragmento (controlador de IU). O código de tomada de decisão, como a descoberta da próxima palavra embaralhada, e os cálculos da pontuação e da contagem de palavras precisam estar no ViewModel.

### **ViewModel**

O [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel?hl=pt-br" \t "_blank) é um modelo dos dados do app exibidos nas visualizações. Modelos são componentes responsáveis por manipular os dados de um app. Eles permitem que seu app siga o princípio da arquitetura, fornecendo a base da IU pelo modelo.

O [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel?hl=pt-br" \t "_blank) armazena os dados relacionados ao app que não são destruídos quando a atividade ou o fragmento são destruídos e recriados pelo framework do Android. Os objetos ViewModel são mantidos automaticamente (não são destruídos como a atividade ou uma instância de fragmento) durante as mudanças de configuração. Assim, os dados mantidos ficam imediatamente disponíveis para a próxima atividade ou instância de fragmento.

Para implementar o ViewModel no seu app, estenda a classe ViewModel, que é da biblioteca de Componentes da arquitetura, e armazene os dados do app nessa classe.

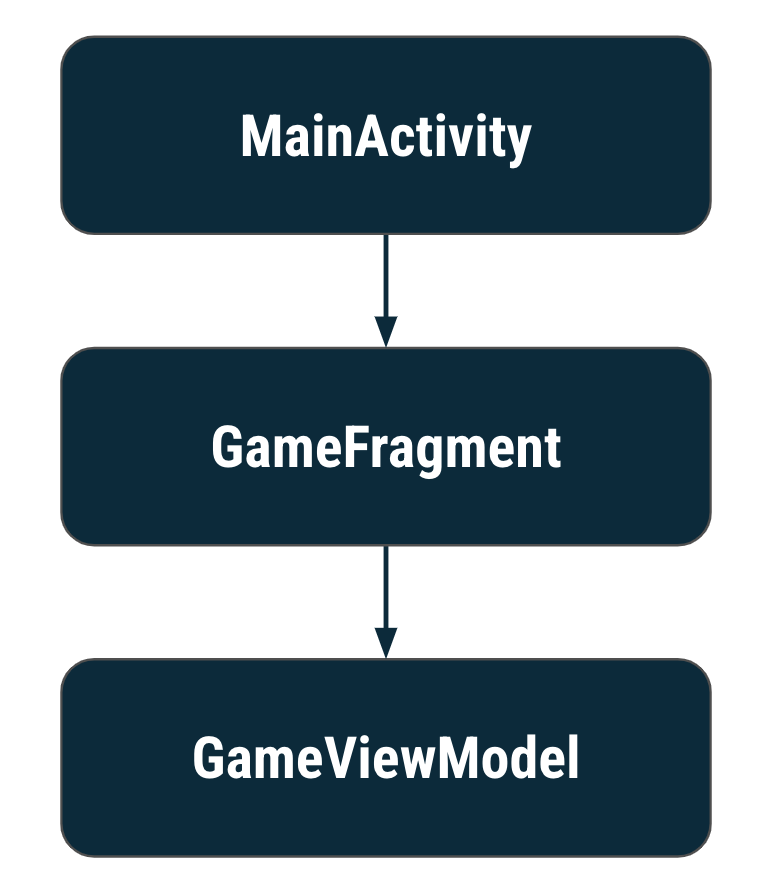
Para resumir:

|  |  |
| --- | --- |
| Responsabilidades de fragmento / atividade (controlador de IU) | Responsabilidades ViewModel |
| Atividades e fragmentos são responsáveis por exibir visualizações e dados na tela e responder aos eventos do usuário. | O ViewModel é responsável por armazenar e processar todos os dados necessários para a IU. Ele nunca pode acessar sua hierarquia de visualização (como o objeto de vinculação de visualizações) nem manter uma referência à atividade ou ao fragmento. |

## [4. Adicionar um ViewModel](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#3)

Nesta tarefa, você adicionará um ViewModel ao app para armazenar dados do app (a palavra embaralhada, a contagem de palavras e a pontuação).

Seu app será arquitetado da seguinte maneira. Sua MainActivity terá um GameFragment, e o GameFragment acessará informações sobre o jogo no GameViewModel.

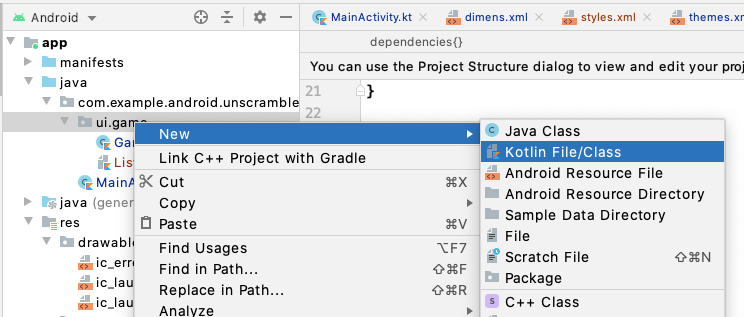


1. Na janela **Android** do Android Studio na pasta **Gradle Scripts**, abra o arquivo build.gradle(Module:Unscramble.app).
2. Para usar o ViewModel no seu app, verifique se você tem a dependência da biblioteca do ViewModel no bloco dependencies. Esta etapa já foi concluída.

// ViewModel  
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0'

Sempre use a [versão mais recente](https://developer.android.com/jetpack/androidx/releases/lifecycle?hl=pt-br) da biblioteca.

1. Crie um novo arquivo de classe do Kotlin com o nome GameViewModel. Na janela **Android**, clique com o botão direito do mouse na pasta **ui.game**. Selecione **New** > **Kotlin File/Class**.



1. Nomeie o arquivo como GameViewModel e selecione **Class** na lista.
2. Mude GameViewModel para ser uma subclasse do ViewModel. O ViewModel é uma classe abstrata, então será necessário estendê-lo para usá-lo no seu app. Veja a definição da classe GameViewModel abaixo.

class GameViewModel : ViewModel() {  
}

## Anexar o ViewModel ao fragmento

Para associar um ViewModel a um controlador de IU (atividade / fragmento), crie uma referência (objeto) ao ViewModel no controlador de IU.

Nesta etapa, você criará uma instância de objeto do GameViewModel no controlador de IU correspondente, que é o GameFragment.

1. Na parte superior da classe GameFragment, adicione uma propriedade do tipo GameViewModel.
2. Inicialize o GameViewModel usando o delegado da propriedade by viewModels() do Kotlin. Você aprenderá mais sobre isso na próxima seção.

private val viewModel: GameViewModel by viewModels()

1. Se solicitado pelo Android Studio, import androidx.fragment.app.viewModels.

### **Delegado de propriedade do Kotlin**

Em Kotlin, cada propriedade mutável (var) tem funções getter e setter geradas automaticamente. As funções setter e getter são chamadas quando você atribui um valor ou lê o valor da propriedade.

Para uma propriedade somente leitura (val), o comportamento é um pouco diferente de uma propriedade mutável. Somente a função getter é gerada por padrão. Essa função getter é chamada quando você lê o valor de uma propriedade somente leitura.

A delegação de propriedades no Kotlin ajuda a passar a responsabilidade do getter-setter para uma classe diferente.

Essa classe, conhecida como classe delegada, fornece funções getter e setter da propriedade e processa as mudanças.

Uma propriedade delegada é definida usando a cláusula by e uma instância da classe delegada:

// Syntax for property delegation  
var <property-name> : <property-type> by <delegate-class>()

No seu app, se você inicializar o modelo de visualização usando o construtor padrão GameViewModel, como no exemplo abaixo:

private val viewModel = GameViewModel()

O app perderá o estado da referência viewModel quando uma mudança de configuração ocorrer no dispositivo. Por exemplo, se você girar o dispositivo, a atividade será destruída e criada novamente, e você terá uma nova instância do modelo de visualização com o estado inicial novamente.

Em vez disso, use a abordagem de delegação de propriedade e delegue a responsabilidade do objeto viewModel a uma classe separada com o nome viewModels. Isso significa que quando você acessar o objeto viewModel, ele será processado internamente pela classe delegada, viewModels. A classe delegada criará o objeto viewModel para você no primeiro acesso e manterá o valor dele durante as mudanças de configuração e retornará o valor quando solicitado.

[5. Transferir dados para o ViewModel](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#4)

Separar os dados de IU do seu app do controlador de IU (suas classes Activity / Fragment) permite seguir melhor o princípio de responsabilidade exclusiva discutido acima. Suas atividades e fragmentos são responsáveis por exibir visualizações e dados na tela, enquanto o ViewModel é responsável por armazenar e processar todos os dados necessários da IU.

Nesta tarefa, você moverá as variáveis de dados do GameFragment para a classe GameViewModel.

1. Mova as variáveis de dados score, currentWordCount, currentScrambledWord para a classe GameViewModel.

class GameViewModel : ViewModel() {  
  
    private var score = 0  
    private var currentWordCount = 0  
    private var currentScrambledWord = "test"  
...

1. Veja que há erros sobre referências não resolvidas. Isso ocorre porque as propriedades são particulares para o ViewModel e não podem ser acessadas pelo seu controlador de IU. Você corrigirá esses erros em seguida.

Para resolver esse problema, não é recomendado criar modificadores de visibilidade para tornar as propriedades public. Esses dados não podem ser editados por outras classes. Isso é arriscado porque uma classe externa pode mudar os dados de maneiras inesperadas que não seguem as regras de jogo especificadas no modelo de visualização. Por exemplo, uma classe externa pode mudar a score para um valor negativo.

No ViewModel, os dados precisam ser editáveis. Portanto, eles precisam ser private e var. Fora do ViewModel, os dados precisam ser legíveis, mas não editáveis, portanto, os dados precisam ser expostos como public e val. Para criar esse comportamento, o Kotlin oferece um recurso conhecido como [propriedade de apoio](https://kotlinlang.org/docs/reference/properties.html#backing-properties) (link em inglês).

Propriedade de apoio

Uma propriedade de apoio permite que você retorne algo de um getter diferente do objeto exato.

Você já sabe que, para cada propriedade, o framework do Kotlin gera getters e setters.

Para os métodos getter e setter, é possível substituir um desses métodos, ou ambos, e fornecer um comportamento personalizado próprio. Para implementar uma propriedade de apoio, você substituirá o método getter para retornar uma versão somente leitura dos dados. Exemplo de uma propriedade de apoio:

// Declare private mutable variable that can only be modified  
// within the class it is declared.  
private var \_count = 0  
  
// Declare another public immutable field and override its getter method.  
// Return the private property's value in the getter method.  
// When count is accessed, the get() function is called and  
// the value of \_count is returned.  
val count: Int  
   get() = \_count

Considere um exemplo, no app, para que os dados do app sejam particulares para o ViewModel:

Na classe ViewModel:

* A propriedade \_count é private e mutável. Portanto, ela só pode ser acessada e editada na classe ViewModel. A convenção é prefixar a propriedade private com um sublinhado.

Fora da classe ViewModel:

* O modificador de visibilidade padrão no Kotlin é public. Portanto, a classe count é pública e pode ser acessada por outras classes, como controladores de IU. Como apenas o método get() está sendo substituído, essa propriedade é imutável e somente leitura. Quando uma classe externa acessar essa propriedade, ela retornará o valor de \_count que não pode ser modificado. Isso protege os dados do app no ViewModel contra mudanças indesejadas e não seguras por classes externas, mas permite que os autores de chamadas externas acessem o valor com segurança.

Adicionar uma propriedade de apoio à currentScrambledWord

1. No GameViewModel, mude a declaração currentScrambledWord para adicionar uma propriedade de apoio. Agora a \_currentScrambledWord pode ser acessada e editada somente no GameViewModel. O controlador de IU GameFragment pode ler o valor dela usando a propriedade somente leitura, currentScrambledWord.

private var \_currentScrambledWord = "test"  
val currentScrambledWord: String  
   get() = \_currentScrambledWord

1. No GameFragment, atualize o método updateNextWordOnScreen() para usar a propriedade viewModel somente leitura, currentScrambledWord.

private fun updateNextWordOnScreen() {  
   binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord  
}

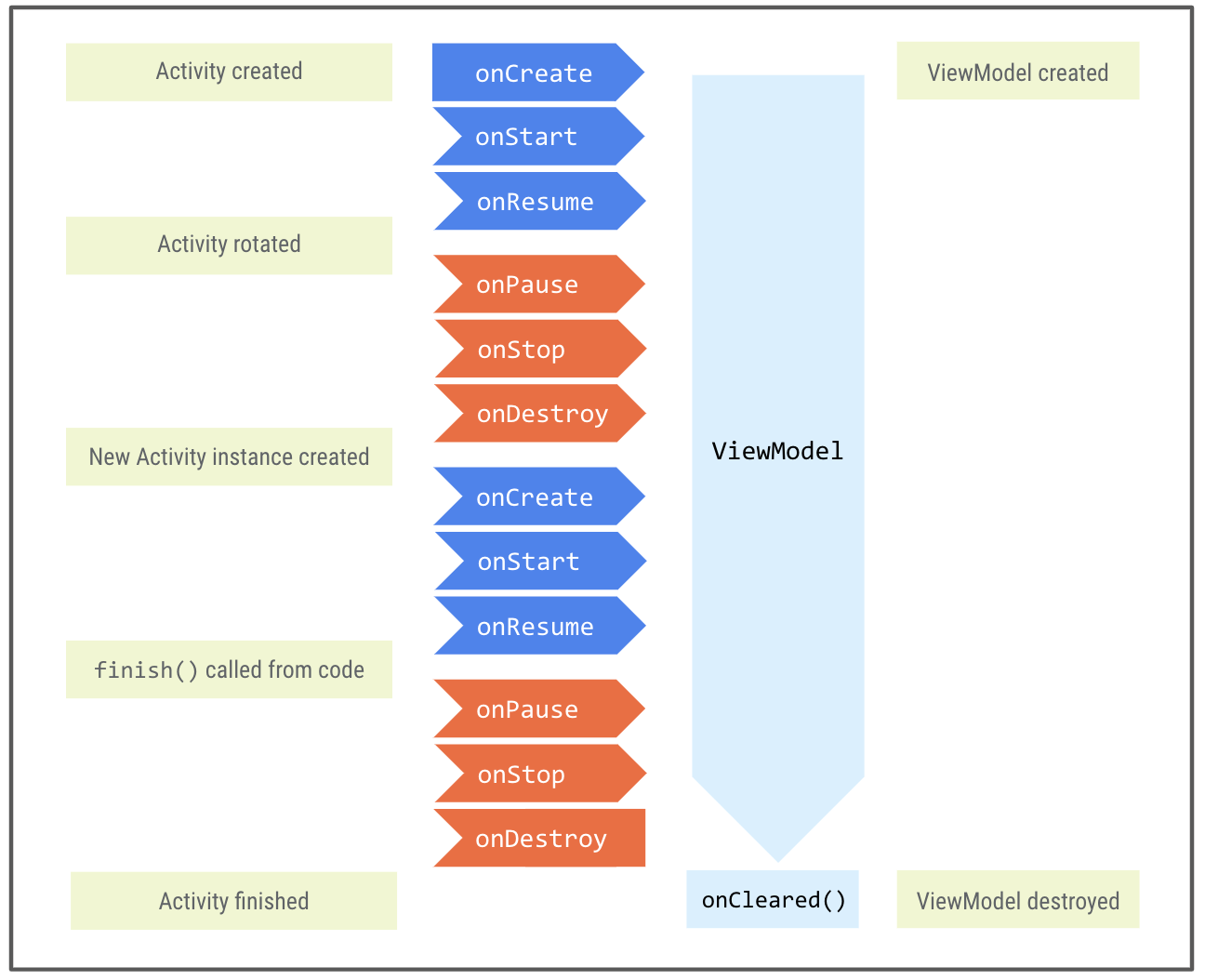
1. No GameFragment, exclua o código dos métodos onSubmitWord() e onSkipWord(). Você implementará esses métodos mais tarde. Agora, você poderá compilar o código sem erros.

**Aviso**: nunca exponha campos de dados mutáveis do **ViewModel**. Esses dados não podem ser modificados por outras classes. Os dados mutáveis dentro do **ViewModel** precisam ser sempre **private**.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[6. O ciclo de vida de um ViewModel](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#5)

O framework mantém o [ViewModel](https://developer.android.com/reference/androidx/lifecycle/ViewModel.html?hl=pt-br" \t "_blank) ativo, desde que o escopo da atividade ou o fragmento esteja ativo. Um ViewModel não será destruído se o proprietário for destruído durante uma mudança de configuração, como a rotação da tela. A nova instância do proprietário se reconectará à instância ViewModel existente, conforme ilustrado por este diagrama:



Entender o ciclo de vida do ViewModel

Adicione a geração de registros no GameViewModel e no GameFragment para entender melhor o ciclo de vida do ViewModel.

1. No arquivo GameViewModel.kt, adicione um bloco init com um log statement.

class GameViewModel : ViewModel() {  
   init {  
       Log.d("GameFragment", "GameViewModel created!")  
   }  
  
   ...  
}

O Kotlin fornece o bloco inicializador, também conhecido como o bloco init, como o local para o código de configuração inicial necessário durante a inicialização de uma instância de objeto. Os blocos do inicializador são prefixados pela palavra-chave init seguida por chaves {}. Esse bloco de código é executado quando a instância do objeto é criada e inicializada pela primeira vez.

1. Na classe GameViewModel, substitua o método [onCleared()](https://developer.android.com/reference/androidx/lifecycle/ViewModel?hl=pt-br" \l "onCleared()" \t "_blank). O ViewModel é destruído quando o fragmento associado é desanexado ou quando a atividade é concluída. Logo antes do ViewModel ser destruído, o callback onCleared() é chamado.
2. Adicione um log statement ao método onCleared() para monitorar o ciclo de vida do GameViewModel.

override fun onCleared() {  
    super.onCleared()  
    Log.d("GameFragment", "GameViewModel destroyed!")  
}

1. No GameFragment, no método onCreateView(), depois de receber uma referência para o objeto de vinculação, adicione um log statement para registrar a criação do fragmento. O callback onCreateView() será acionado quando o fragmento for criado pela primeira vez e sempre que ele for recriado para eventos como mudanças de configuração.

override fun onCreateView(  
   inflater: LayoutInflater, container: ViewGroup?,  
   savedInstanceState: Bundle?  
): View {  
   binding = GameFragmentBinding.inflate(inflater, container, false)  
   Log.d("GameFragment", "GameFragment created/re-created!")  
   return binding.root  
}

1. No GameFragment, substitua o método de callback onDetach(), que será chamado quando a atividade e o fragmento correspondentes forem destruídos.

override fun onDetach() {  
    super.onDetach()  
    Log.d("GameFragment", "GameFragment destroyed!")  
}

1. No Android Studio, execute o app, abra a janela **Logcat** e filtre por GameFragment. GameFragment e GameViewModel serão criados.

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: GameViewModel created!

1. Ative a configuração de giro automático do dispositivo ou emulador e mude a orientação da tela algumas vezes. O GameFragment é destruído e recriado todas as vezes que a tela gira, mas o GameViewModel é criado apenas uma vez e não é recriado nem destruído para cada chamada.

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: GameViewModel created!

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

1. Saia do jogo ou saia do app usando a seta para voltar. O GameViewModel será destruído e o callback onCleared() será chamado. O GameFragment será destruído.

com.example.android.unscramble D/GameFragment: GameViewModel destroyed!

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[7. Preencher o ViewModel](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#6)

Nesta tarefa, você preencherá ainda mais o GameViewModel com métodos auxiliares para acessar a próxima palavra, validar a palavra do jogador para aumentar a pontuação e verificar a contagem de palavras para finalizar o jogo.

Inicialização atrasada

Normalmente, ao declarar uma variável, você fornece um valor inicial antecipadamente. No entanto, se ainda não estiver pronto para atribuir um valor, você poderá inicializá-lo depois. Para inicializar uma propriedade no Kotlin posteriormente, use a palavra-chave lateinit, que significa inicialização atrasada. Se você tiver certeza que a propriedade será inicializada antes de usá-la, poderá declarar a propriedade usando lateinit. A memória não será alocada para a variável até que ela seja inicializada. Se você tentar acessar a variável antes de inicializá-la, o app falhará.

Acessar a próxima palavra

Crie o método getNextWord() na classe GameViewModel com esta funcionalidade:

* Acesse uma palavra aleatória da allWordsList e atribua-a à currentWord..
* Embaralhe as letras na currentWord e atribua o resultado à currentScrambledWord para criar uma palavra embaralhada.
* Lide com o caso em que a palavra embaralhada é igual à palavra não embaralhada.
* Não mostre a mesma palavra duas vezes durante a partida.

Implemente estas etapas na classe GameViewModel:

1. No GameViewModel,, adicione uma nova variável de classe do tipo MutableList<String> com o nome wordsList para armazenar uma lista de palavras que você usará no jogo para evitar repetições.
2. Adicione outra variável de classe com o nome currentWord para armazenar a palavra que o jogador está tentando decifrar. Use a palavra-chave lateinit, já que você inicializará esta propriedade mais tarde.

private var wordsList: MutableList<String> = mutableListOf()  
private lateinit var currentWord: String

1. Adicione um novo método private sem parâmetros com o nome getNextWord() que não retornará nada.
2. Acesse uma palavra aleatória da allWordsList e atribua-a à currentWord.

private fun getNextWord() {  
   currentWord = allWordsList.random()  
}

1. No método getNextWord(), converta a string currentWord em uma matriz de caracteres e atribua ela a uma nova val com o nome tempWord. Embaralhe os caracteres nesta matriz usando o método do Kotlin, [shuffle()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/shuffle.html" \t "_blank) (link em inglês) para criar a palavra embaralhada.

val tempWord = currentWord.toCharArray()  
tempWord.shuffle()

Uma Array é semelhante a uma List, mas tem um tamanho fixo quando é inicializada. Uma Array não pode aumentar nem diminuir de tamanho (é necessário copiar uma matriz para redimensioná-la), enquanto uma List tem as funções add() e remove(), para que possa aumentar e diminuir de tamanho.

1. Algumas vezes, a ordem aleatória de caracteres será igual a palavra original. Adicione esta repetição while na chamada para embaralhar a palavra para continuar a repetição até que a palavra embaralhada não seja igual à palavra original.

while (tempWord.toString().equals(currentWord, false)) {  
    tempWord.shuffle()  
}

1. Adicione um bloco if-else para verificar se uma palavra já foi usada. Se a wordsList contiver currentWord, chame getNextWord(). Caso contrário, atualize o valor da \_currentScrambledWord com a palavra recém embaralhada, aumente a contagem de palavras e adicione a nova palavra à wordsList.

if (wordsList.contains(currentWord)) {  
    getNextWord()  
} else {  
    \_currentScrambledWord = String(tempWord)  
    ++currentWordCount  
    wordsList.add(currentWord)  
}

1. Este é o método getNextWord() completo da sua referência.

/\*  
\* Updates currentWord and currentScrambledWord with the next word.  
\*/  
private fun getNextWord() {  
   currentWord = allWordsList.random()  
   val tempWord = currentWord.toCharArray()  
   tempWord.shuffle()  
  
   while (tempWord.toString().equals(currentWord, false)) {  
       tempWord.shuffle()  
   }  
   if (wordsList.contains(currentWord)) {  
       getNextWord()  
   } else {  
       \_currentScrambledWord = String(tempWord)  
       ++currentWordCount  
       wordsList.add(currentWord)  
   }  
}

Inicialização atrasada da currentScrambledWord

Agora, você criou o método getNextWord() para acessar a próxima palavra. Ele será chamado quando o GameViewModel for inicializado pela primeira vez. Use o bloco init para inicializar propriedades lateinit na classe, como a palavra atual. O resultado é que a primeira palavra exibida na tela será uma palavra embaralhada em vez de **test**.

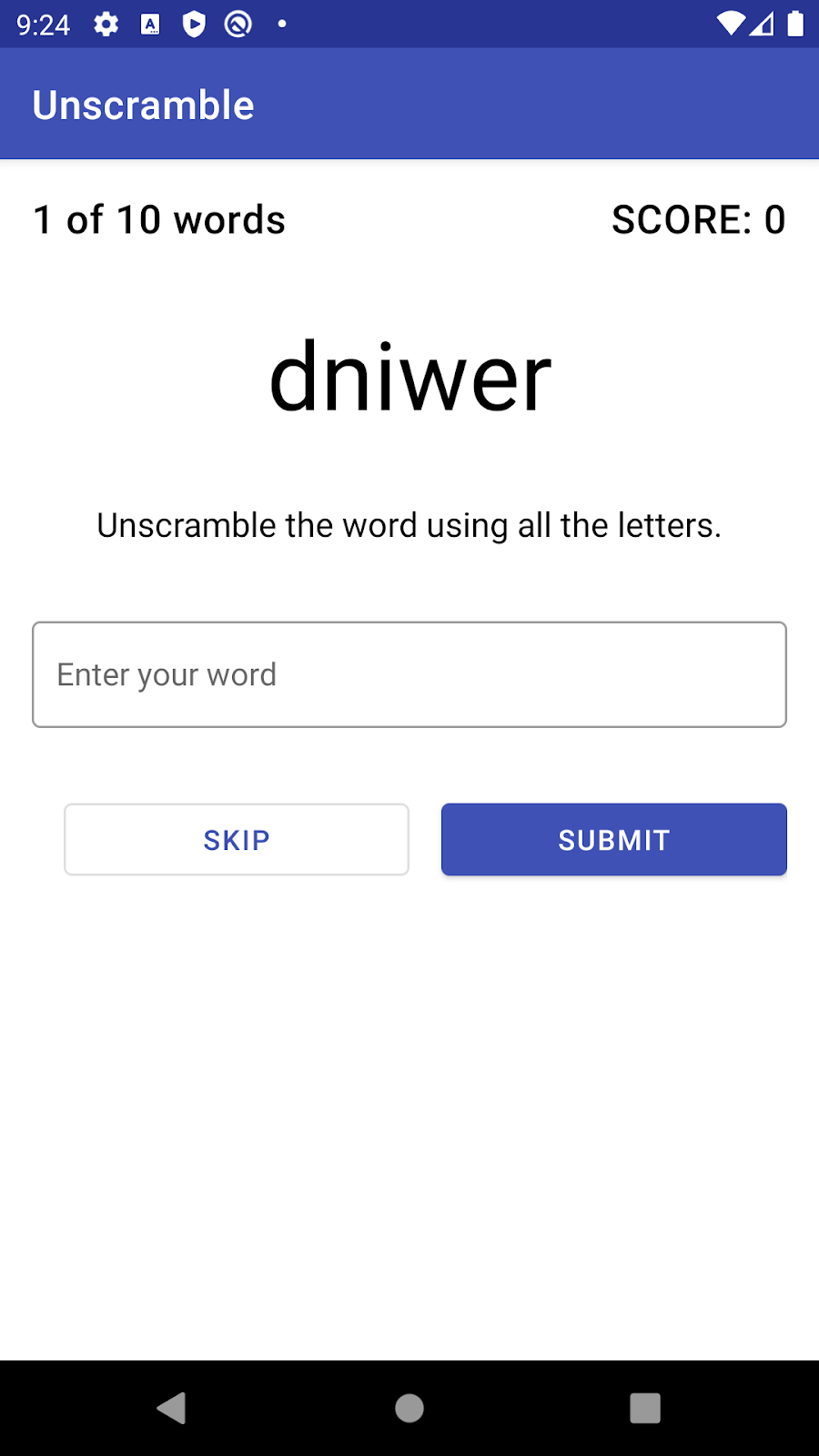
1. Execute o app. A primeira palavra sempre será "**test**".
2. Para exibir uma palavra embaralhada ao iniciar o app, você precisa chamar o método getNextWord(), que atualizará a currentScrambledWord. Chame o método getNextWord() no bloco init do GameViewModel.

init {  
    Log.d("GameFragment", "GameViewModel created!")  
    getNextWord()  
}

1. Adicione o modificador lateinit à propriedade \_currentScrambledWord. Adicione uma referência explícita do tipo de dados String, já que nenhum valor inicial é fornecido.

private lateinit var \_currentScrambledWord: String

1. Execute o app. Observe que uma nova palavra embaralhada será exibida na inicialização do app. Incrível!



Adicionar um método auxiliar

Em seguida, adicione um método auxiliar para processar e modificar os dados no ViewModel. Você usará esse método em tarefas futuras de codelab.

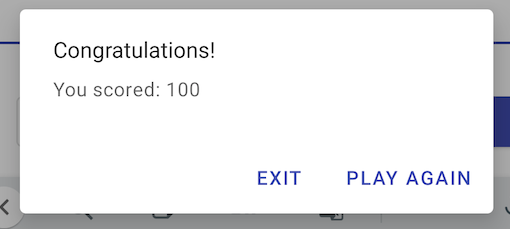
1. Na classe GameViewModel, adicione outro método com o nome nextWord().. Acesse a próxima palavra da lista e retorne true se a contagem de palavras for menor que o *MAX\_NO\_OF\_WORDS*.

/\*  
\* Returns true if the current word count is less than MAX\_NO\_OF\_WORDS.  
\* Updates the next word.  
\*/  
fun nextWord(): Boolean {  
    return if (currentWordCount < MAX\_NO\_OF\_WORDS) {  
        getNextWord()  
        true  
    } else false  
}

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

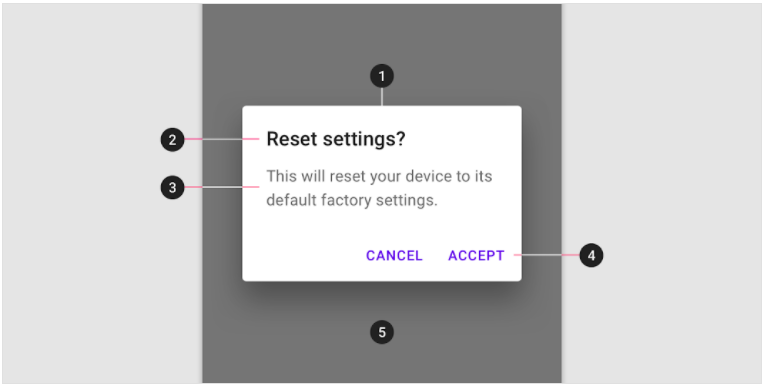
## [8. Caixas de diálogo](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#7)

No código inicial, a partida nunca acabava, mesmo após 10 palavras. Modifique o app para que, após o usuário passar por 10 palavras, a partida termine exibindo uma caixa de diálogo com a pontuação final. Também é possível oferecer ao usuário a opção de jogar novamente ou sair do jogo.



Esta é a primeira vez que você adicionará uma caixa de diálogo a um app. As caixas de diálogo são pequenas janelas (telas) que solicitam ao usuário uma decisão ou mais informações. Normalmente, uma caixa de diálogo não preenche toda a tela e exige que os usuários realizem uma ação antes de continuar. O Android oferece diferentes tipos de caixas de diálogo. Neste codelab, você aprenderá sobre caixas de diálogo de aviso.

### **Anatomia da caixa de diálogo de aviso**



1. Caixa de diálogo de aviso
2. Título (opcional)
3. Mensagem
4. Botões de texto

## Implementar a caixa de diálogo da pontuação final

Use a [MaterialAlertDialog](https://material.io/develop/android/components/dialogs" \t "_blank) (link em inglês) da biblioteca de componentes do Material Design para adicionar uma caixa de diálogo ao seu app que siga as diretrizes do Material Design. Como a caixa de diálogo está relacionada à IU, o GameFragment será responsável por criar e exibir a caixa de diálogo da pontuação final.

1. Primeiro, adicione uma propriedade de apoio à variável score. No GameViewModel, mude a declaração da variável score para esta.

private var \_score = 0  
val score: Int  
   get() = \_score

1. No GameFragment, adicione uma função particular com o nome showFinalScoreDialog(). Para criar uma MaterialAlertDialog, use a classe MaterialAlertDialogBuilder para criar partes da caixa de diálogo passo a passo. Chame o construtor [MaterialAlertDialogBuilder](https://developer.android.com/reference/com/google/android/material/dialog/MaterialAlertDialogBuilder?hl=pt-br" \t "_blank) transmitindo o conteúdo usando o método [requireContext()](https://developer.android.com/reference/androidx/fragment/app/Fragment?hl=pt-br" \l "requireContext()" \t "_blank) do fragmento. O método requireContext() retorna um Context não nulo.

/\*  
\* Creates and shows an AlertDialog with the final score.  
\*/  
private fun showFinalScoreDialog() {  
   MaterialAlertDialogBuilder(requireContext())  
}

O [Context](https://developer.android.com/reference/android/content/Context.html?hl=pt-br" \t "_blank), como o nome sugere, significa o contexto ou o estado atual do aplicativo, da atividade ou do fragmento. Ele contém as informações relacionadas à atividade, ao fragmento ou ao app. Geralmente, ele é usado para acessar recursos, bancos de dados e outros serviços do sistema. Nesta etapa, você transmitirá o contexto do fragmento para criar a caixa de diálogo de aviso.

Se solicitado pelo Android Studio, import com.google.android.material.dialog.MaterialAlertDialogBuilder.

1. Adicione o código para definir o título na caixa de diálogo de aviso e use um recurso de string do arquivo strings.xml.

MaterialAlertDialogBuilder(requireContext())  
   .setTitle(getString(R.string.congratulations))

1. Defina a mensagem para mostrar a pontuação final e use a versão somente leitura da variável correspondente viewModel.score que você adicionou anteriormente.

   .setMessage(getString(R.string.you\_scored, viewModel.score))

1. Faça com que não seja possível cancelar a caixa de diálogo de aviso quando a tecla "Voltar" for pressionada usando o método [setCancelable()](https://developer.android.com/reference/android/app/Dialog?hl=pt-br" \l "setCancelable(boolean)" \t "_blank) e transmitindo false.

    .setCancelable(false)

1. Adicione dois botões de texto **SAIR** e **JOGAR NOVAMENTE** usando os métodos setNegativeButton() e setPositiveButton(). Chame exitGame() e restartGame(), respectivamente, nos lambdas.

    .setNegativeButton(getString(R.string.exit)) { \_, \_ ->  
        exitGame()  
    }  
    .setPositiveButton(getString(R.string.play\_again)) { \_, \_ ->  
        restartGame()  
    }

Essa sintaxe pode ser novidade para você, mas é uma abreviação de setNegativeButton(getString(R.string.exit), { \_, \_ -> exitGame()}) em que o método setNegativeButton() recebe dois parâmetros: uma String e uma função, DialogInterface.OnClickListener(), que podem ser expressos como lambdas. Quando o último argumento transmitido é uma função, você pode colocar a expressão lambda fora dos parênteses. Isso é conhecido como [sintaxe de lambda final](https://kotlinlang.org/docs/reference/lambdas.html#passing-a-lambda-to-the-last-parameter) (link em inglês). Ambas as formas de escrever o código (com o lambda dentro ou fora dos parênteses) são aceitáveis. O mesmo se aplica à função setPositiveButton.

1. No final, adicione o método [show()](https://developer.android.com/reference/android/app/Dialog?hl=pt-br#show()), que cria e exibe a caixa de diálogo de aviso.

      .show()

1. Este é o método showFinalScoreDialog() completo para referência.

/\*  
\* Creates and shows an AlertDialog with the final score.  
\*/  
private fun showFinalScoreDialog() {  
   MaterialAlertDialogBuilder(requireContext())  
       .setTitle(getString(R.string.congratulations))  
       .setMessage(getString(R.string.you\_scored, viewModel.score))  
       .setCancelable(false)  
       .setNegativeButton(getString(R.string.exit)) { \_, \_ ->  
           exitGame()  
       }  
       .setPositiveButton(getString(R.string.play\_again)) { \_, \_ ->  
           restartGame()  
       }  
       .show()  
}

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

## [9. Implementar OnClickListener para botão](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#8)

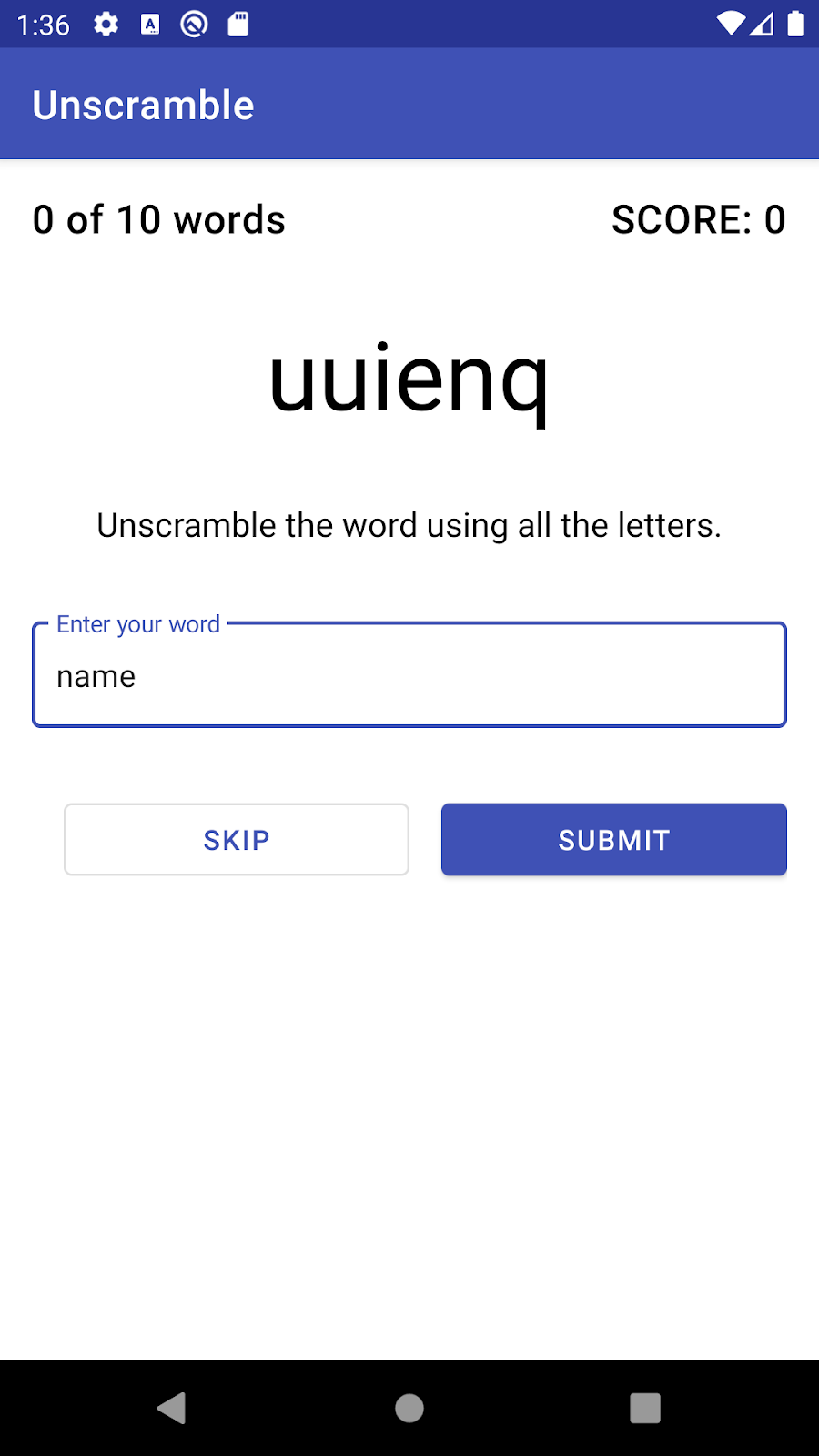
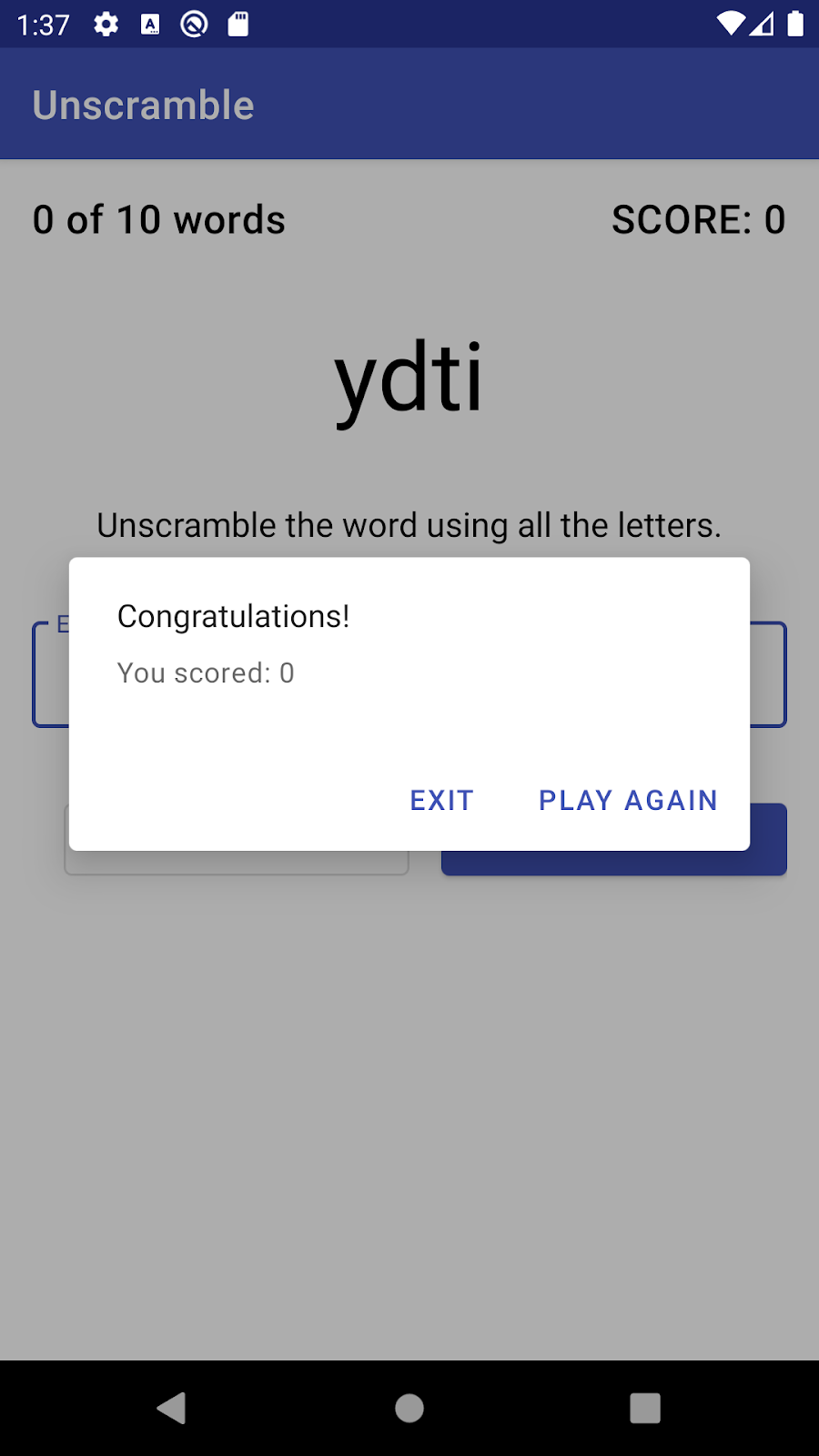
Nesta tarefa, você usa o ViewModel e a caixa de diálogo de aviso que adicionou para implementar a lógica do jogo para o listener de clique do botão **Submit**.

## Exibir as palavras embaralhadas

1. Caso ainda não tenha feito isso, no GameFragment, exclua o código do método onSubmitWord(), que é chamado quando o usuário toca no botão **Submit**.
2. Adicione uma verificação para o valor de retorno do método viewModel.nextWord(). Se o valor for true, outra palavra estará disponível, portanto, atualize a palavra embaralhada exibida na tela usando updateNextWordOnScreen(). Caso contrário, a partida terminou, portanto, exiba a caixa de diálogo de aviso com a pontuação final.

private fun onSubmitWord() {  
    if (viewModel.nextWord()) {  
        updateNextWordOnScreen()  
    } else {  
        showFinalScoreDialog()  
    }  
}

1. Execute o app. Jogue uma partida com algumas palavras. Lembre-se de que você ainda não implementou o botão **Skip**, por isso não é possível pular a palavra.
2. O campo de texto não é atualizado, por isso o jogador precisa excluir manualmente a palavra anterior. A pontuação final na caixa de diálogo de aviso será sempre zero. Você corrigirá esses bugs nas próximas etapas.

## Adicionar um método auxiliar para validar a palavra do jogador

1. No GameViewModel, adicione um novo método particular com o nome increaseScore() sem parâmetros e nenhum valor de retorno. Aumente a variável score usando *SCORE\_INCREASE*.

private fun increaseScore() {  
   \_score += SCORE\_INCREASE  
}

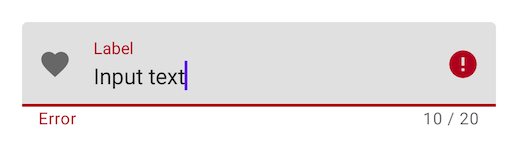
1. No método GameViewModel, adicione um método auxiliar com o nome isUserWordCorrect() que retorna um Boolean e recebe uma String, a palavra do jogador, como um parâmetro.
2. No método isUserWordCorrect(), valide a palavra do jogador e aumente a pontuação se o palpite estiver correto. Isso atualizará a pontuação final na caixa de diálogo de aviso.

fun isUserWordCorrect(playerWord: String): Boolean {  
   if (playerWord.equals(currentWord, true)) {  
       increaseScore()  
       return true  
   }  
   return false  
}

Atualizar o campo de texto

### **Mostrar erros no campo de texto**

Para campos de texto do Material Design, o [TextInputLayout](https://developer.android.com/reference/com/google/android/material/textfield/TextInputLayout?hl=pt-br" \t "_blank) oferece uma funcionalidade integrada para exibir mensagens de erro. Por exemplo, no campo de texto a seguir, a cor da etiqueta muda, um ícone de erro é exibido, uma mensagem de erro é exibida e assim por diante.



Para mostrar um erro no campo de texto, defina a mensagem de erro dinamicamente no código ou estaticamente no arquivo de layout. Veja abaixo um exemplo para definir e redefinir o erro no código:

// Set error text  
passwordLayout.error = getString(R.string.error)  
  
// Clear error text  
passwordLayout.error = null

No código inicial, você verá que o método auxiliar setErrorTextField(error: Boolean) já está definido para ajudar a definir e redefinir o erro no campo de texto. Chame esse método usando true ou false como o parâmetro de entrada se você quiser ou não que um erro seja exibido no campo de texto.

Snippet de código no código inicial

private fun setErrorTextField(error: Boolean) {  
   if (error) {  
       binding.textField.isErrorEnabled = true  
       binding.textField.error = getString(R.string.try\_again)  
   } else {  
       binding.textField.isErrorEnabled = false  
       binding.textInputEditText.text = null  
   }  
}

Nesta tarefa, você implementará o método onSubmitWord(). Quando uma palavra for enviada, valide o palpite do usuário verificando a palavra original. Se a palavra estiver correta, vá para a próxima palavra (ou mostre a caixa de diálogo se a partida tiver terminado). Se a palavra estiver incorreta, mostre um erro no campo de texto e continue na palavra atual.

1. No GameFragment, no início do método onSubmitWord(), crie uma val com o nome playerWord. Armazene a palavra do jogador extraindo-a do campo de texto da variável binding.

private fun onSubmitWord() {  
    val playerWord = binding.textInputEditText.text.toString()  
    ...  
}

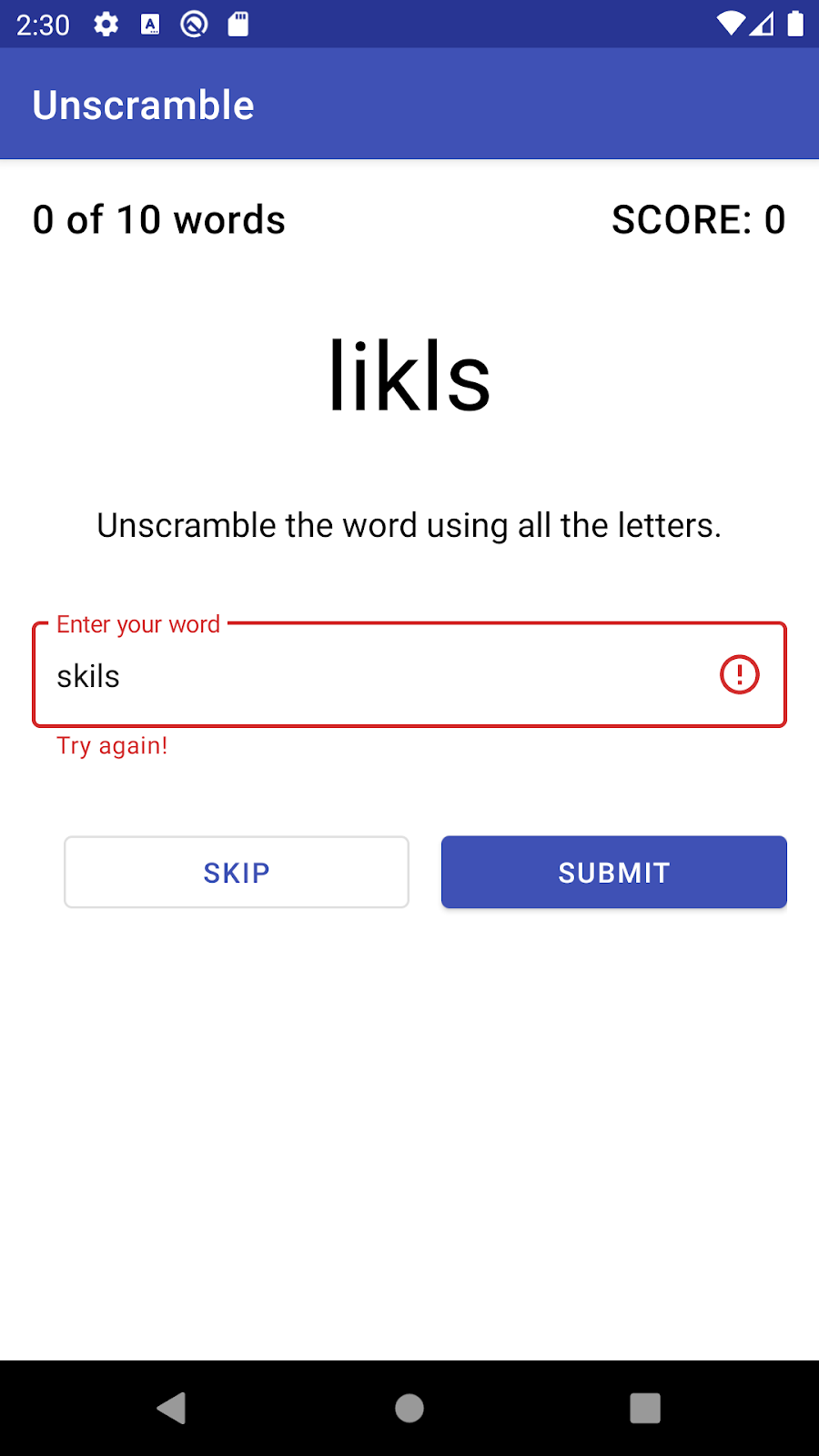
1. No método onSubmitWord(), abaixo da declaração da playerWord, valide a palavra do jogador. Adicione uma instrução if para verificar a palavra do jogador usando o método isUserWordCorrect() e transmitindo a playerWord.
2. No bloco if, redefina o campo de texto e chame setErrorTextField transmitindo false.
3. Mova o código existente para o bloco if.

private fun onSubmitWord() {  
    val playerWord = binding.textInputEditText.text.toString()  
  
    if (viewModel.isUserWordCorrect(playerWord)) {  
        setErrorTextField(false)  
        if (viewModel.nextWord()) {  
            updateNextWordOnScreen()  
        } else {  
            showFinalScoreDialog()  
        }  
    }  
}

1. Se a palavra do usuário estiver incorreta, mostre uma mensagem de erro no campo de texto. Adicione um bloco else ao bloco if acima e chame o método setErrorTextField() transmitindo true. O método onSubmitWord() concluído ficará assim:

private fun onSubmitWord() {  
    val playerWord = binding.textInputEditText.text.toString()  
  
    if (viewModel.isUserWordCorrect(playerWord)) {  
        setErrorTextField(false)  
        if (viewModel.nextWord()) {  
            updateNextWordOnScreen()  
        } else {  
            showFinalScoreDialog()  
        }  
    } else {  
        setErrorTextField(true)  
    }  
}

1. Execute o app. Jogue uma partida com algumas palavras. Se a palavra do jogador estiver correta, ela será apagada ao clicar no botão **Submit**, caso contrário, uma mensagem "Try again!" será exibida para que o jogador possa tentar novamente. O botão **Skip** ainda não funciona. Você adicionará essa implementação na próxima tarefa.



[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[10. Implementar o botão](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#9)

Nesta tarefa, você implementará o método onSkipWord() que processa o clique do botão **Skip**.

1. Semelhante ao método onSubmitWord(), adicione uma condição no método nextWord(). Se o valor retornado for true, exiba a palavra na tela e redefina o campo de texto. Se o valor retornado for false e não houver mais palavras nesta partida, mostre a caixa de diálogo de aviso com a pontuação final.

/\*  
\* Skips the current word without changing the score.  
\*/  
private fun onSkipWord() {  
    if (viewModel.nextWord()) {  
        setErrorTextField(false)  
        updateNextWordOnScreen()  
    } else {  
        showFinalScoreDialog()  
    }  
}

1. Execute o app. Jogue uma partida. Observe que os botões **Skip** e **Submit** estão funcionando como esperado. Excelente!

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[11. Verificar se o ViewModel preserva dados](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#10)

Nesta tarefa, adicione a geração de registros ao GameFragment para observar que os dados do app são preservados no ViewModel durante as mudanças na configuração. Para acessar a currentWordCount no GameFragment, é necessário expor uma versão somente leitura usando uma propriedade de apoio.

1. No GameViewModel, clique com o botão direito do mouse na variável currentWordCount, selecione **Refactor** > **Rename...** . Adicione um sublinhado ao novo nome: \_currentWordCount.
2. Adicione um campo de apoio.

private var \_currentWordCount = 0  
val currentWordCount: Int  
   get() = \_currentWordCount

1. No GameFragment no método onCreateView(), acima da instrução de retorno, adicione outro registro para exibir os dados do app, a palavra, a pontuação e a contagem de palavras.

Log.d("GameFragment", "Word: ${viewModel.currentScrambledWord} " +  
       "Score: ${viewModel.score} WordCount: ${viewModel.currentWordCount}")

1. No Android Studio, abra o **Logcat** e filtre por GameFragment. Execute o app e jogue uma partida com algumas palavras. Mude a orientação do dispositivo. O fragmento (controlador de IU) será destruído e recriado. Veja os registros. Agora, você poderá ver a pontuação e a contagem de palavras aumentando.

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: GameViewModel created!

com.example.android.unscramble D/GameFragment: Word: oimfnru Score: 0 WordCount: 1

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: Word: ofx Score: 80 WordCount: 5

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: Word: ofx Score: 80 WordCount: 5

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: Word: nvoiil Score: 160 WordCount: 9

com.example.android.unscramble D/GameFragment: GameFragment destroyed!

com.example.android.unscramble D/GameFragment: GameFragment created/re-created!

com.example.android.unscramble D/GameFragment: Word: nvoiil Score: 160 WordCount: 9

Os dados do app são preservados no ViewModel durante as mudanças de orientação. Você atualizará o valor da pontuação e a contagem de palavras na IU usando o LiveData e a vinculação de dados nos próximos codelabs.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[12. Atualizar lógica de reinicialização do jogo](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#11)

1. Execute o app novamente, jogue até o final. Na caixa de diálogo de aviso **Congratulations!**, clique em **PLAY AGAIN** para jogar novamente. O app não deixará você jogar novamente porque o número de palavras atingiu o valor *MAX\_NO\_OF\_WORDS*. Você precisará redefinir a contagem de palavras para 0 para jogar de novo desde o começo.
2. Para redefinir os dados do app, no GameViewModel adicione um método com o nome reinitializeData(). Defina a pontuação e a contagem de palavras como 0. Limpe a lista de palavras e chame o método getNextWord().

/\*  
\* Re-initializes the game data to restart the game.  
\*/  
fun reinitializeData() {  
   \_score = 0  
   \_currentWordCount = 0  
   wordsList.clear()  
   getNextWord()  
}

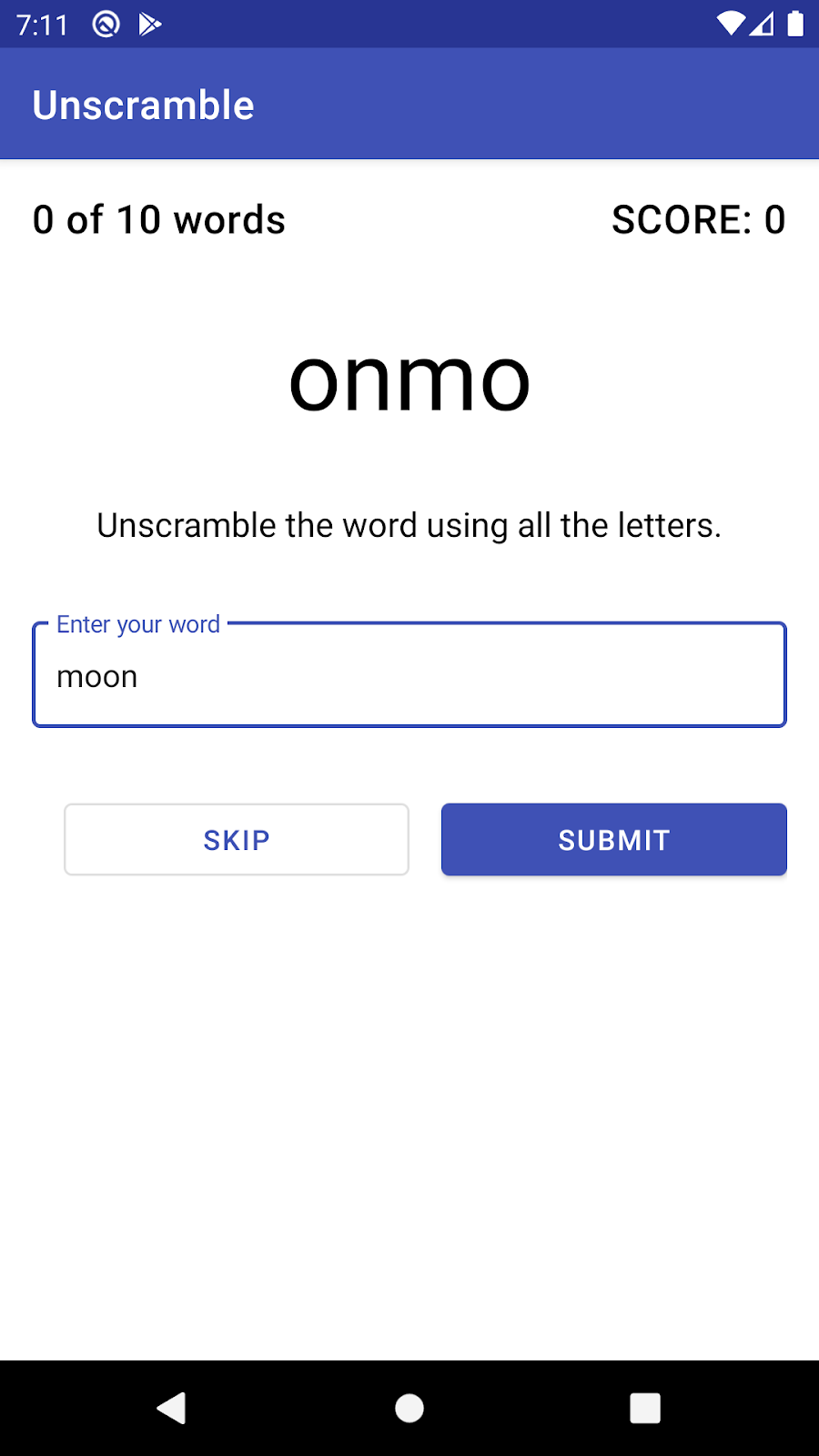
1. No GameFragment na parte superior do método restartGame(), chame o método recém-criado, reinitializeData().

private fun restartGame() {  
   viewModel.reinitializeData()  
   setErrorTextField(false)  
   updateNextWordOnScreen()  
}

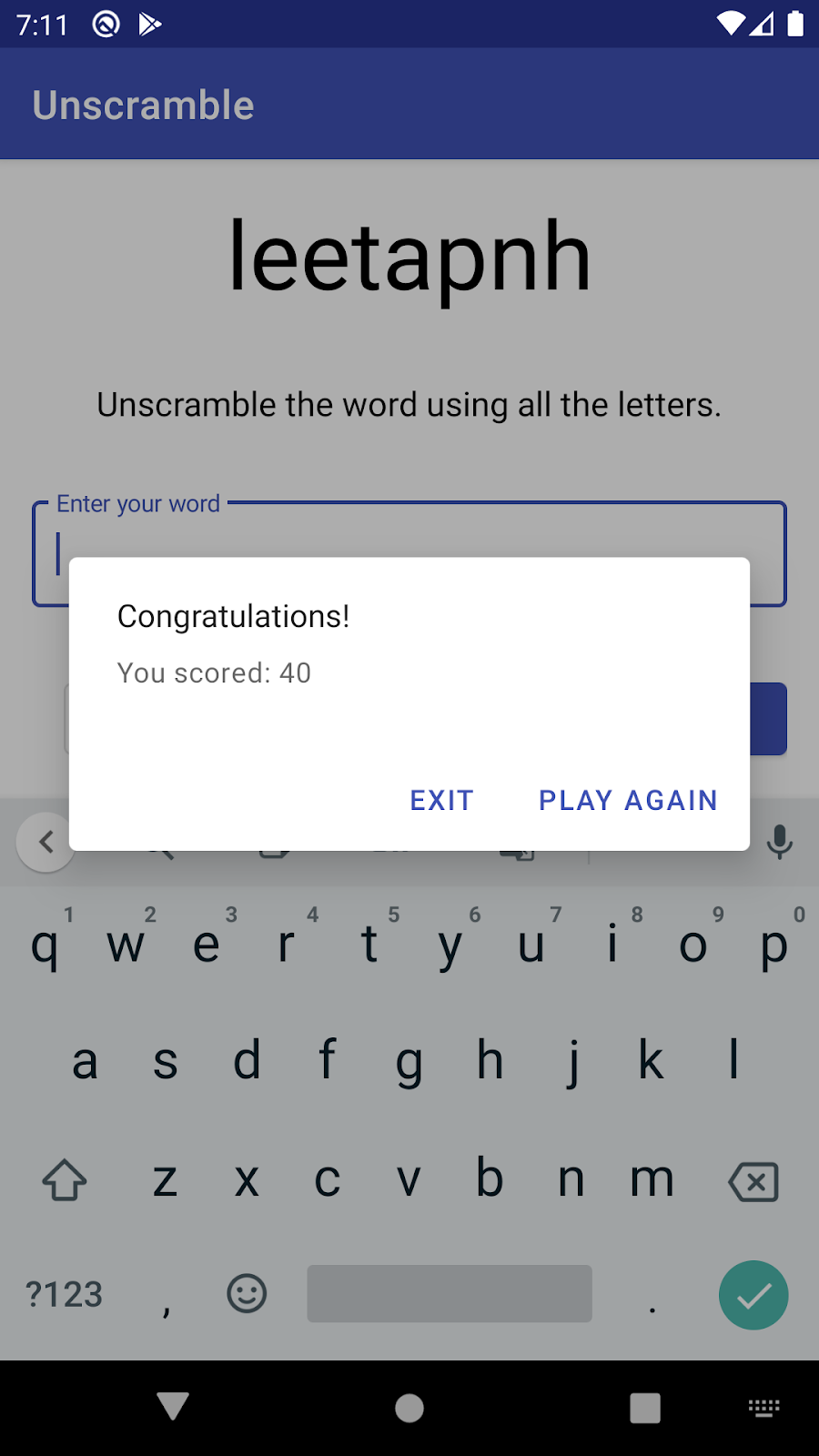
1. Execute o app novamente. Jogue uma partida. Na caixa de diálogo de parabéns, clique em **Play again**. Agora você conseguirá jogar novamente.

O app final funcionará assim. O jogo mostrará dez palavras aleatórias embaralhadas para o jogador decifrar. Você pode pular a palavra tocando no botão **Skip** ou tentar adivinhá-la tocando no botão **Submit**. Caso seu palpite esteja correto, a pontuação aumentará. Um palpite incorreto mostrará um estado de erro no campo de texto. A cada nova palavra, a contagem de palavras também aumentará.

A pontuação e a contagem de palavras exibidas na tela ainda não serão atualizadas. No entanto, as informações ainda estão sendo armazenadas no modelo de visualização e preservadas durante as mudanças de configuração, como a rotação do dispositivo. Você atualizará a pontuação e a contagem de palavras na tela nos próximos codelabs.

Ao final de 10 palavras, a partida acabará, e uma caixa de diálogo de aviso será exibida com a pontuação final, além de uma opção para sair do jogo ou jogar novamente.



Parabéns! Você criou seu primeiro ViewModel e salvou os dados.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

## [13. Código da solução](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#12)

**OBSERVAÇÃO**: sempre inclua o nome do pacote do seu app em todos os arquivos de origem do Kotlin.

GameFragment.kt

import android.os.Bundle  
import android.util.Log  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.fragment.app.Fragment  
import androidx.fragment.app.viewModels  
import com.example.android.unscramble.R  
import com.example.android.unscramble.databinding.GameFragmentBinding  
import com.google.android.material.dialog.MaterialAlertDialogBuilder  
  
/\*\*  
 \* Fragment where the game is played, contains the game logic.  
 \*/  
class GameFragment : Fragment() {  
  
    private val viewModel: GameViewModel by viewModels()  
  
    // Binding object instance with access to the views in the game\_fragment.xml layout  
    private lateinit var binding: GameFragmentBinding  
  
    // Create a ViewModel the first time the fragment is created.  
    // If the fragment is re-created, it receives the same GameViewModel instance created by the  
    // first fragment  
  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View {  
        // Inflate the layout XML file and return a binding object instance  
        binding = GameFragmentBinding.inflate(inflater, container, false)  
        Log.d("GameFragment", "GameFragment created/re-created!")  
        Log.d("GameFragment", "Word: ${viewModel.currentScrambledWord} " +  
                "Score: ${viewModel.score} WordCount: ${viewModel.currentWordCount}")  
        return binding.root  
    }  
  
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
        super.onViewCreated(view, savedInstanceState)  
  
        // Setup a click listener for the Submit and Skip buttons.  
        binding.submit.setOnClickListener { onSubmitWord() }  
        binding.skip.setOnClickListener { onSkipWord() }  
        // Update the UI  
        updateNextWordOnScreen()  
        binding.score.text = getString(R.string.score, 0)  
        binding.wordCount.text = getString(  
            R.string.word\_count, 0, MAX\_NO\_OF\_WORDS)  
    }  
  
    /\*  
    \* Checks the user's word, and updates the score accordingly.  
    \* Displays the next scrambled word.  
    \* After the last word, the user is shown a Dialog with the final score.  
    \*/  
    private fun onSubmitWord() {  
        val playerWord = binding.textInputEditText.text.toString()  
  
        if (viewModel.isUserWordCorrect(playerWord)) {  
            setErrorTextField(false)  
            if (viewModel.nextWord()) {  
                updateNextWordOnScreen()  
            } else {  
                showFinalScoreDialog()  
            }  
        } else {  
            setErrorTextField(true)  
        }  
    }  
  
    /\*  
    \* Skips the current word without changing the score.  
    \*/  
    private fun onSkipWord() {  
        if (viewModel.nextWord()) {  
            setErrorTextField(false)  
            updateNextWordOnScreen()  
        } else {  
            showFinalScoreDialog()  
        }  
    }  
  
    /\*  
     \* Gets a random word for the list of words and shuffles the letters in it.  
     \*/  
    private fun getNextScrambledWord(): String {  
        val tempWord = allWordsList.random().toCharArray()  
        tempWord.shuffle()  
        return String(tempWord)  
    }  
  
    /\*  
    \* Creates and shows an AlertDialog with the final score.  
    \*/  
    private fun showFinalScoreDialog() {  
        MaterialAlertDialogBuilder(requireContext())  
            .setTitle(getString(R.string.congratulations))  
            .setMessage(getString(R.string.you\_scored, viewModel.score))  
            .setCancelable(false)  
            .setNegativeButton(getString(R.string.exit)) { \_, \_ ->  
                exitGame()  
            }  
            .setPositiveButton(getString(R.string.play\_again)) { \_, \_ ->  
                restartGame()  
            }  
            .show()  
    }  
  
    /\*  
     \* Re-initializes the data in the ViewModel and updates the views with the new data, to  
     \* restart the game.  
     \*/  
    private fun restartGame() {  
        viewModel.reinitializeData()  
        setErrorTextField(false)  
        updateNextWordOnScreen()  
    }  
  
    /\*  
     \* Exits the game.  
     \*/  
    private fun exitGame() {  
        activity?.finish()  
    }  
  
    override fun onDetach() {  
        super.onDetach()  
        Log.d("GameFragment", "GameFragment destroyed!")  
    }  
  
    /\*  
    \* Sets and resets the text field error status.  
    \*/  
    private fun setErrorTextField(error: Boolean) {  
        if (error) {  
            binding.textField.isErrorEnabled = true  
            binding.textField.error = getString(R.string.try\_again)  
        } else {  
            binding.textField.isErrorEnabled = false  
            binding.textInputEditText.text = null  
        }  
    }  
  
    /\*  
     \* Displays the next scrambled word on screen.  
     \*/  
    private fun updateNextWordOnScreen() {  
        binding.textViewUnscrambledWord.text = viewModel.currentScrambledWord  
    }  
}

GameViewModel.kt

import android.util.Log  
import androidx.lifecycle.ViewModel  
  
/\*\*  
 \* ViewModel containing the app data and methods to process the data  
 \*/  
class GameViewModel : ViewModel(){  
    private var \_score = 0  
    val score: Int  
        get() = \_score  
  
    private var \_currentWordCount = 0  
    val currentWordCount: Int  
        get() = \_currentWordCount  
  
    private lateinit var \_currentScrambledWord: String  
    val currentScrambledWord: String  
        get() = \_currentScrambledWord  
  
    // List of words used in the game  
    private var wordsList: MutableList<String> = mutableListOf()  
    private lateinit var currentWord: String  
  
    init {  
        Log.d("GameFragment", "GameViewModel created!")  
        getNextWord()  
    }  
  
    override fun onCleared() {  
        super.onCleared()  
        Log.d("GameFragment", "GameViewModel destroyed!")  
    }  
  
    /\*  
    \* Updates currentWord and currentScrambledWord with the next word.  
    \*/  
    private fun getNextWord() {  
        currentWord = allWordsList.random()  
        val tempWord = currentWord.toCharArray()  
        tempWord.shuffle()  
  
        while (tempWord.toString().equals(currentWord, false)) {  
            tempWord.shuffle()  
        }  
        if (wordsList.contains(currentWord)) {  
            getNextWord()  
        } else {  
            \_currentScrambledWord = String(tempWord)  
            ++\_currentWordCount  
            wordsList.add(currentWord)  
        }  
    }  
  
    /\*  
    \* Re-initializes the game data to restart the game.  
    \*/  
    fun reinitializeData() {  
       \_score = 0  
       \_currentWordCount = 0  
       wordsList.clear()  
       getNextWord()  
    }  
  
    /\*  
    \* Increases the game score if the player's word is correct.  
    \*/  
    private fun increaseScore() {  
        \_score += SCORE\_INCREASE  
    }  
  
    /\*  
    \* Returns true if the player word is correct.  
    \* Increases the score accordingly.  
    \*/  
    fun isUserWordCorrect(playerWord: String): Boolean {  
        if (playerWord.equals(currentWord, true)) {  
            increaseScore()  
            return true  
        }  
        return false  
    }  
  
    /\*  
    \* Returns true if the current word count is less than MAX\_NO\_OF\_WORDS  
    \*/  
    fun nextWord(): Boolean {  
        return if (\_currentWordCount < MAX\_NO\_OF\_WORDS) {  
            getNextWord()  
            true  
        } else false  
    }  
}

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[14. Resumo](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#13)

* As diretrizes de arquitetura de apps Android recomendam separar classes que tenham responsabilidades diferentes usar um modelo para a IU.
* Um controlador de IU é uma classe com base na IU, como a Activity ou o Fragment. Os controladores de IU precisam conter somente a lógica que processa as interações entre a IU e o sistema operacional. Eles não podem ser a fonte de dados a ser exibida na IU. Armazene esses dados e qualquer lógica relacionada em um ViewModel.
* A classe ViewModel armazena e gerencia dados relacionados à IU. A classe ViewModel permite que os dados sobrevivam às mudanças de configuração, como a rotação da tela.
* O ViewModel é um dos componentes recomendados pelos Componentes da arquitetura do Android.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel)

[15. Saiba mais](https://developer.android.com/codelabs/basic-android-kotlin-training-viewmodel?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-3%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-viewmodel#14)

* [Visão geral do ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel?authuser=1&hl=pt-br)
* [Guia para a arquitetura do app](https://developer.android.com/jetpack/docs/guide?authuser=1&hl=pt-br)
* [Os componentes do Material Design para Android em prática: caixas de diálogo](https://medium.com/over-engineering/hands-on-with-material-components-for-android-dialogs-75c6d726f83a) (links em inglês)
* [Anatomia da caixa de diálogo de alerta](https://material.io/components/dialogs#anatomy) (link em inglês)
* [MaterialAlertDialogBuilder](https://developer.android.com/reference/com/google/android/material/dialog/MaterialAlertDialogBuilder?hl=pt-br)
* [Propriedades de apoio](https://kotlinlang.org/docs/reference/properties.html#backing-properties) (link em inglês)
* [Componentes da arquitetura do Android](https://developer.android.com/topic/libraries/architecture?hl=pt-br)
* [Caixas de diálogo](https://material.io/develop/android/components/dialogs) do Material Design para Android (link em inglês)
* [Propriedades e campos: getters, setters, const e lateinit](https://kotlinlang.org/docs/reference/properties.html) (link em inglês)