Начинать с начала Unit4

<https://developer.android.com/courses/android-basics-compose/course>

ViewModel и состояние в Compose

Дополнительные видео

<https://www.youtube.com/watch?v=AfCzIEwt_i4&t=275s>

# <https://www.youtube.com/watch?v=p9VR8KbmzEE> - Architecture: The UI layer - MAD Skills

На предыдущих занятиях по коду вы узнали о жизненном цикле активити и связанных с ним проблемах жизненного цикла, связанных с изменениями конфигурации. При изменении конфигурации вы можете сохранить данные приложения различными способами, например, используя rememberSaveable или сохраняя состояние экземпляра. Однако эти варианты могут создать проблемы. В большинстве случаев вы можете использовать, rememberSaveableно это может означать сохранение логики внутри компонуемых объектов или рядом с ними. Когда приложения растут, вам следует переместить данные и логику из составных элементов. В этой лаборатории кода вы узнаете о надежном способе разработки приложения и сохранении данных приложения во время изменений конфигурации, используя преимущества библиотеки Android Jetpack ViewModelи рекомендации по архитектуре приложений Android.

[Библиотеки Android Jetpack](https://developer.android.com/jetpack) — это набор библиотек, упрощающих разработку отличных приложений для Android. Эти библиотеки помогают вам следовать передовым практикам, освобождают вас от написания шаблонного кода и упрощают сложные задачи, чтобы вы могли сосредоточиться на коде, который вас интересует, например на логике приложения.

Архитектура приложения — это набор правил проектирования приложения. Подобно проекту дома, ваша архитектура обеспечивает структуру вашего приложения. Хорошая архитектура приложения может сделать ваш код надежным, гибким, масштабируемым, тестируемым и поддерживаемым на долгие годы. Руководство [по архитектуре приложений](https://developer.android.com/topic/libraries/architecture) содержит рекомендации по архитектуре приложений и рекомендуемые лучшие практики.

В этой лабораторной работе вы узнаете, как использовать [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel" \t "_blank), один из компонентов архитектуры из библиотек Android Jetpack, который может хранить данные вашего приложения. Сохраненные данные не теряются, если платформа уничтожает и воссоздает действия во время изменения конфигурации или других событий. Однако данные теряются, если действие уничтожается из-за смерти процесса. Единственный ViewModelкэширует данные посредством быстрого восстановления активности.

## **Предварительные условия**

* Знание Kotlin, включая функции, лямбды и составные элементы без сохранения состояния.
* Базовые знания о том, как создавать макеты в Jetpack Compose.
* Базовые знания Material Design.

## **Что вы узнаете**

* Введение в [архитектуру приложения Android](https://developer.android.com/topic/architecture#recommended-app-arch)
* Как использовать [ViewModel](https://developer.android.com/reference/android/arch/lifecycle/ViewModel" \t "_blank)класс в вашем приложении
* Как сохранить данные пользовательского интерфейса при изменении конфигурации устройства с помощью [ViewModel](https://developer.android.com/reference/android/arch/lifecycle/ViewModel" \t "_blank)

## **Что ты построишь**

* Игровое приложение [Unscramble](https://github.com/google-developer-training/basic-android-kotlin-compose-training-unscramble" \t "_blank) , в котором пользователь может угадывать зашифрованные слова.

## **Что вам понадобится**

* Последняя версия Android Studio
* Подключение к Интернету для загрузки стартового кода.

## [**2. Обзор приложения**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#1)

## **Обзор игры**

Приложение Unscramble — это однопользовательская игра в шифрование слов. Приложение отображает зашифрованное слово, и игрок должен угадать слово, используя все показанные буквы. Игрок получает очки, если слово правильное. В противном случае игрок может попытаться угадать слово любое количество раз. В приложении также есть возможность пропустить текущее слово. В правом верхнем углу приложение отображает количество слов, то есть количество зашифрованных слов, сыгранных в текущей игре. В каждой игре 10 зашифрованных слов.

|  |  |  |
| --- | --- | --- |
|  |  |  |

## **Получить стартовый код**

Для начала загрузите стартовый код:

[file\_downloadСкачать zip](https://github.com/google-developer-training/basic-android-kotlin-compose-training-unscramble/archive/refs/heads/starter.zip)

Альтернативно вы можете клонировать репозиторий GitHub для кода:

$ git clone

https://github.com/google-developer-training/basic-android-kotlin-compose-training-unscramble.git

$ cd basic-android-kotlin-compose-training-unscramble

$ git checkout starter

**Примечание.** Стартовый код находится в starterветке скачанного репозитория.

Вы можете просмотреть стартовый код в [Unscramble](https://github.com/google-developer-training/basic-android-kotlin-compose-training-unscramble/tree/starter" \t "_blank)репозитории GitHub.

[3. Обзор стартового приложения](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#2)

Чтобы ознакомиться со стартовым кодом, выполните следующие действия:

1. Откройте проект со стартовым кодом в Android Studio.
2. Запустите приложение на устройстве Android или в эмуляторе.
3. Нажмите кнопки **«Отправить»** и **«Пропустить»** , чтобы протестировать приложение.

Вы заметите ошибки в приложении. Зашифрованное слово не отображается, но оно жестко закодировано как «зашифрованное», и при нажатии кнопок ничего не происходит.

В этой лаборатории кода вы реализуете функциональность игры, используя архитектуру приложения Android.

Прохождение стартового кода

Стартовый код содержит заранее разработанный макет игрового экрана. На этом пути вы реализуете игровую логику. Вы будете использовать компоненты архитектуры для реализации рекомендуемой архитектуры приложения и решения вышеупомянутых проблем. Вот краткое описание некоторых файлов, которые помогут вам начать.

**WordsData.kt**

Этот файл содержит список слов, используемых в игре, константы для максимального количества слов за игру и количество очков, которые игрок получает за каждое правильное слово.

package com.example.android.unscramble.data  
  
const val MAX\_NO\_OF\_WORDS = 10  
const val SCORE\_INCREASE = 20  
  
// Set with all the words for the Game  
val allWords: Set<String> =  
   setOf(  
       "animal",  
       "auto",  
       "anecdote",  
       "alphabet",  
       "all",  
       "awesome",  
       "arise",  
       "balloon",  
       "basket",  
       "bench",  
      // ...  
       "zoology",  
       "zone",  
       "zeal"  
)

**ВНИМАНИЕ** : не рекомендуется жестко кодировать строки в коде. Добавьте строки strings.xmlдля упрощения локализации. В этом примере приложения строки жестко запрограммированы для простоты и позволяют вам сосредоточиться на архитектуре приложения.

**MainActivity.kt**

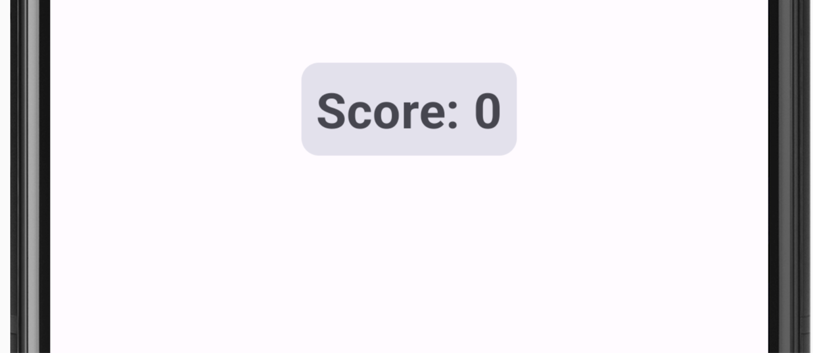
Этот файл содержит в основном код, сгенерированный шаблоном. Вы отображаете GameScreenкомпонуемое в setContent{}блоке.

**GameScreen.kt**

Все составные элементы пользовательского интерфейса определены в GameScreen.ktфайле. В следующих разделах представлено пошаговое описание некоторых компонуемых функций.

**Статус игры**

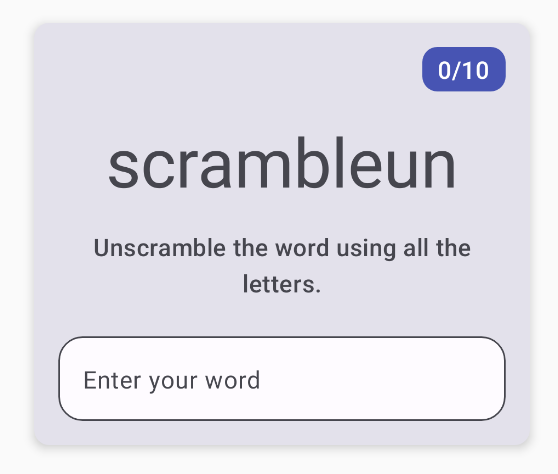
GameStatus— это составная функция, которая отображает счет игры в нижней части экрана. Компонуемая функция содержит текст, который можно компоновать в формате Card. На данный момент счет жестко запрограммирован в 0.



// No need to copy, this is included in the starter code.  
  
@Composable  
fun GameStatus(score: Int, modifier: Modifier = Modifier) {  
    Card(  
        modifier = modifier  
    ) {  
        Text(  
            text = stringResource(R.string.score, score),  
            style = typography.headlineMedium,  
            modifier = Modifier.padding(8.dp)  
        )  
    }  
}

**GameLayout**

GameLayout— это составная функция, которая отображает основные функции игры, включая зашифрованное слово, игровые инструкции и текстовое поле, принимающее догадки пользователя.



Обратите внимание, что GameLayoutприведенный ниже код содержит столбец внутри a Cardс тремя дочерними элементами: текст зашифрованного слова, текст инструкций и текстовое поле для слова пользователя OutlinedTextField. На данный момент зашифрованное слово жестко запрограммировано как scrambleun. Позже в лаборатории кода вы реализуете функцию отображения слова из файла WordsData.kt.

// No need to copy, this is included in the starter code.  
  
@Composable  
fun GameLayout(modifier: Modifier = Modifier) {  
   val mediumPadding = dimensionResource(R.dimen.padding\_medium)  
   Card(  
       modifier = modifier,  
       elevation = CardDefaults.cardElevation(defaultElevation = 5.dp)  
   ) {  
       Column(  
           verticalArrangement = Arrangement.spacedBy(mediumPadding),  
           horizontalAlignment = Alignment.CenterHorizontally,  
           modifier = Modifier.padding(mediumPadding)  
       ) {  
           Text(  
               modifier = Modifier  
                   .clip(shapes.medium)  
                   .background(colorScheme.surfaceTint)  
                   .padding(horizontal = 10.dp, vertical = 4.dp)  
                   .align(alignment = Alignment.End),  
               text = stringResource(R.string.word\_count, 0),  
               style = typography.titleMedium,  
               color = colorScheme.onPrimary  
           )  
           Text(  
               text = "scrambleun",  
               style = typography.displayMedium  
           )  
           Text(  
               text = stringResource(R.string.instructions),  
               textAlign = TextAlign.Center,  
               style = typography.titleMedium  
           )  
           OutlinedTextField(  
               value = "",  
               singleLine = true,  
               shape = shapes.large,  
               modifier = Modifier.fillMaxWidth(),  
               colors = TextFieldDefaults.textFieldColors(containerColor = colorScheme.surface),  
               onValueChange = { },  
               label = { Text(stringResource(R.string.enter\_your\_word)) },  
               isError = false,  
               keyboardOptions = KeyboardOptions.Default.copy(  
                   imeAction = ImeAction.Done  
               ),  
               keyboardActions = KeyboardActions(  
                   onDone = { }  
               )  
           )  
       }  
   }  
}

Компонуемый OutlinedTextFieldобъект похож на TextFieldкомпонуемый из приложений в предыдущих лабораториях кода.

Текстовые поля бывают двух типов:

* Заполненные текстовые поля
* Контурные текстовые поля

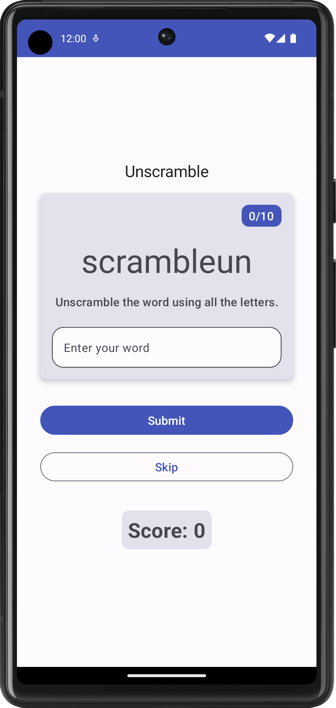


Обведенные текстовые поля имеют меньший визуальный акцент, чем заполненные текстовые поля. Когда они появляются в таких местах, как формы, где много текстовых полей расположены вместе, их уменьшенное выделение помогает упростить макет.

В стартовом коде OutlinedTextFieldне обновляется, когда пользователь вводит предположение. Вы обновите эту функцию в лаборатории кода.

**GameScreen**

Составной GameScreenэлемент содержит составные функции GameStatusи GameLayout, название игры, количество слов и составные элементы для кнопок **«Отправить»** и **«Пропустить»** .

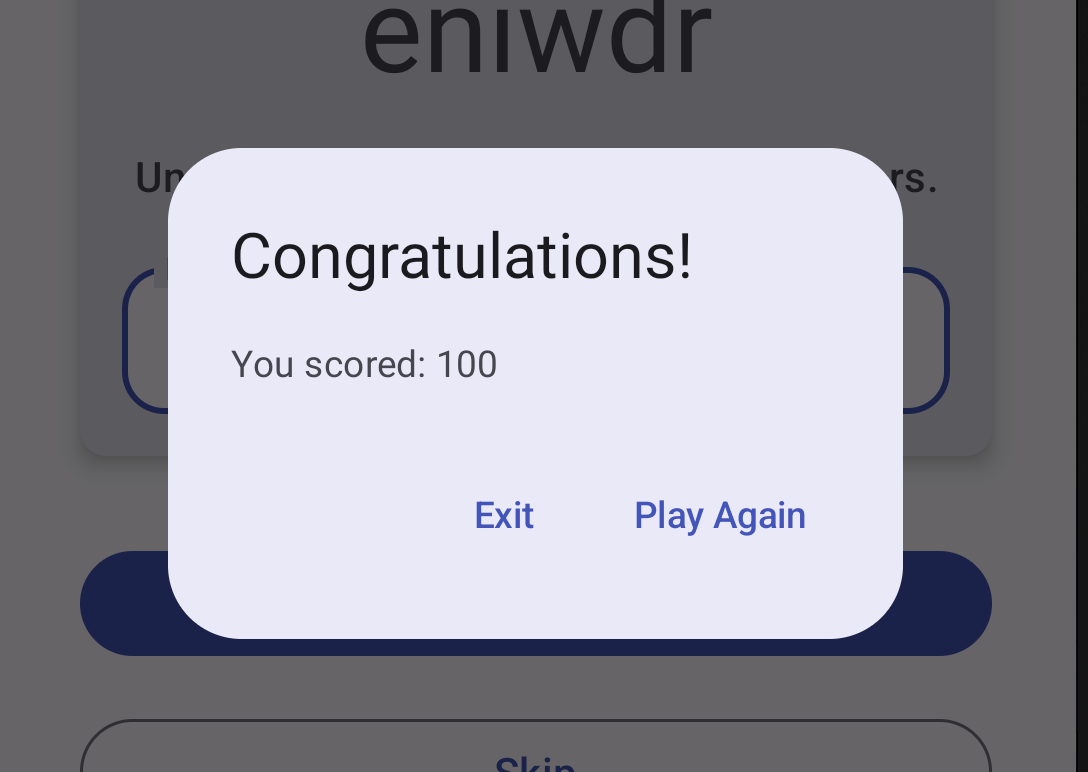


@Composable  
fun GameScreen() {  
    val mediumPadding = dimensionResource(R.dimen.padding\_medium)  
  
    Column(  
        modifier = Modifier  
            .verticalScroll(rememberScrollState())  
            .padding(mediumPadding),  
        verticalArrangement = Arrangement.Center,  
        horizontalAlignment = Alignment.CenterHorizontally  
    ) {  
  
        Text(  
            text = stringResource(R.string.app\_name),  
            style = typography.titleLarge,  
        )  
  
        GameLayout(  
            modifier = Modifier  
                .fillMaxWidth()  
                .wrapContentHeight()  
                .padding(mediumPadding)  
        )  
        Column(  
            modifier = Modifier  
                .fillMaxWidth()  
                .padding(mediumPadding),  
            verticalArrangement = Arrangement.spacedBy(mediumPadding),  
            horizontalAlignment = Alignment.CenterHorizontally  
        ) {  
  
            Button(  
                modifier = Modifier.fillMaxWidth(),  
                onClick = { }  
            ) {  
                Text(  
                    text = stringResource(R.string.submit),  
                    fontSize = 16.sp  
                )  
            }  
  
            OutlinedButton(  
                onClick = { },  
                modifier = Modifier.fillMaxWidth()  
            ) {  
                Text(  
                    text = stringResource(R.string.skip),  
                    fontSize = 16.sp  
                )  
            }  
        }  
  
        GameStatus(score = 0, modifier = Modifier.padding(20.dp))  
    }  
}

События нажатия кнопки не реализованы в стартовом коде. Вы реализуете эти события как часть лаборатории кода.

**FinalScoreDialog**

Компонуемый FinalScoreDialogэлемент отображает диалоговое окно, то есть небольшое окно, предлагающее пользователю варианты « **Играть снова»** или **«Выйти из** игры». Позже в этой лаборатории кода вы реализуете логику для отображения этого диалогового окна в конце игры.



// No need to copy, this is included in the starter code.  
  
@Composable  
private fun FinalScoreDialog(  
    score: Int,  
    onPlayAgain: () -> Unit,  
    modifier: Modifier = Modifier  
) {  
    val activity = (LocalContext.current as Activity)  
  
    AlertDialog(  
        onDismissRequest = {  
           // Dismiss the dialog when the user clicks outside the dialog or on the back  
           // button. If you want to disable that functionality, simply use an empty  
           // onDismissRequest.  
        },  
        title = { Text(text = stringResource(R.string.congratulations)) },  
        text = { Text(text = stringResource(R.string.you\_scored, score)) },  
        modifier = modifier,  
        dismissButton = {  
            TextButton(  
                onClick = {  
                    activity.finish()  
                }  
            ) {  
                Text(text = stringResource(R.string.exit))  
            }  
        },  
        confirmButton = {  
            TextButton(onClick = onPlayAgain) {  
                Text(text = stringResource(R.string.play\_again))  
            }  
        }  
    )  
}

## [**4. Узнайте об архитектуре приложения**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#3)

Архитектура приложения содержит рекомендации, которые помогут вам распределить обязанности приложения между классами. Хорошо продуманная архитектура приложения помогает масштабировать приложение и расширять его дополнительными функциями. Архитектура также может упростить совместную работу команды.

Наиболее распространенными [архитектурными принципами](https://developer.android.com/jetpack/guide#common-principles) являются **разделение задач** и **управление пользовательским интерфейсом на основе модели** .

**Разделение интересов**

Принцип разделения ответственности гласит, что приложение разделено на классы функций, каждый из которых имеет отдельные обязанности.

**Управляйте пользовательским интерфейсом на основе модели**

Принцип управления пользовательским интерфейсом на основе модели гласит, что вы должны управлять своим пользовательским интерфейсом на основе модели, предпочтительно постоянной модели. Модели — это компоненты, отвечающие за обработку данных приложения. Они независимы от элементов пользовательского интерфейса и компонентов приложения в вашем приложении, поэтому на них не влияет жизненный цикл приложения и связанные с ним проблемы.

## **Рекомендуемая архитектура приложения**

Учитывая общие архитектурные принципы, упомянутые в предыдущем разделе, каждое приложение должно иметь как минимум два уровня:

* **Уровень пользовательского интерфейса:** слой, который отображает данные приложения на экране, но не зависит от данных.
* **Уровень данных:** уровень, который хранит, извлекает и предоставляет данные приложения.

Вы можете добавить еще один уровень, называемый уровнем домена, чтобы упростить и повторно использовать взаимодействие между уровнями пользовательского интерфейса и данных. Этот уровень является необязательным и выходит за рамки данного курса.

## 

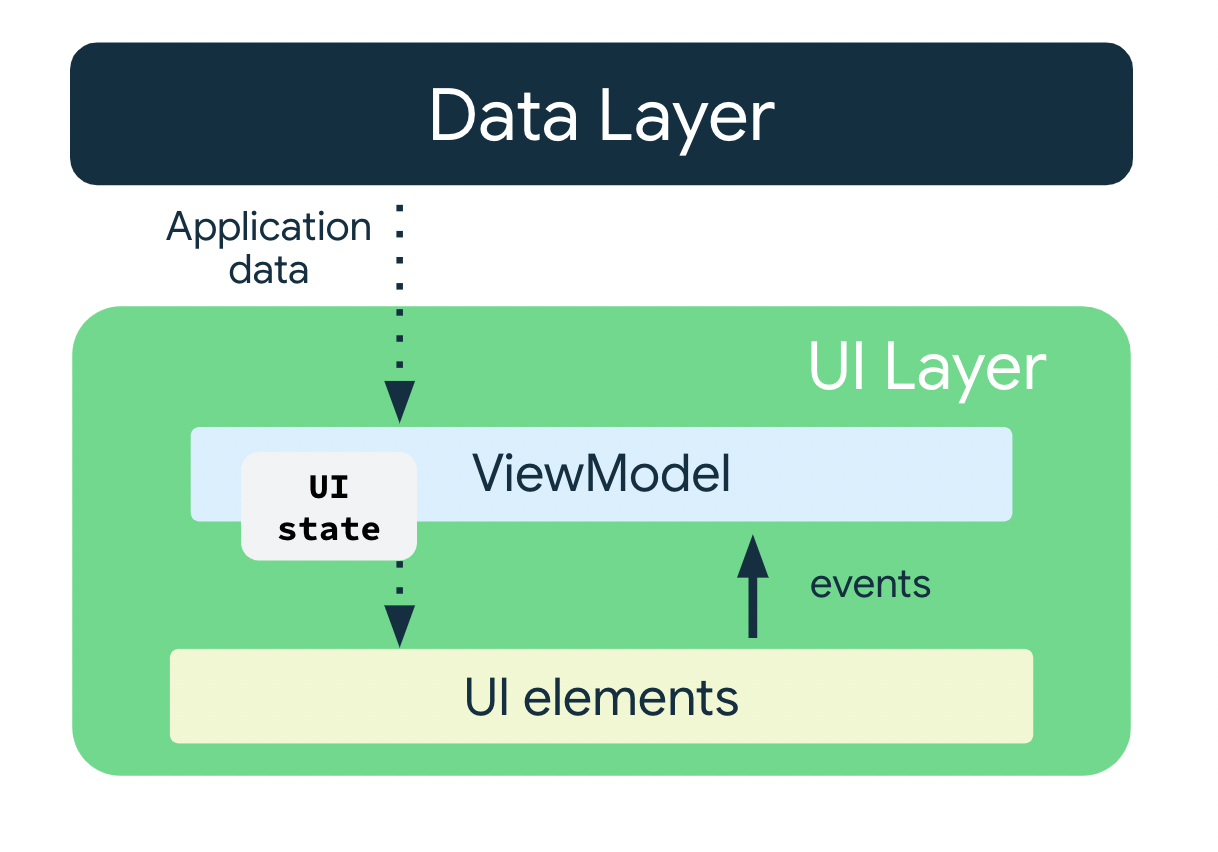
**Примечание** . Стрелки на диаграммах в этом руководстве обозначают зависимости между классами. Например, уровень предметной области зависит от классов уровня данных.

### **Уровень пользовательского интерфейса**

Роль уровня пользовательского интерфейса или уровня представления заключается в отображении данных приложения на экране. Всякий раз, когда данные изменяются в результате взаимодействия с пользователем, например нажатия кнопки, пользовательский интерфейс должен обновляться, чтобы отразить изменения.

Уровень пользовательского интерфейса состоит из следующих компонентов:

* **Элементы пользовательского интерфейса:** компоненты, которые отображают данные на экране. Вы создаете эти элементы с помощью [Jetpack Compose](https://developer.android.com/jetpack/compose" \t "_blank) .
* **Держатели состояний:** компоненты, которые хранят данные, предоставляют их пользовательскому интерфейсу и обрабатывают логику приложения. Пример держателя состояния — [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel" \t "_blank) .



### **Модель представления**

Компонент ViewModelхранит и предоставляет состояние, которое использует пользовательский интерфейс. Состояние пользовательского интерфейса — это данные приложения, преобразованные с помощью ViewModel. ViewModelпозволяет вашему приложению следовать архитектурному принципу управления пользовательским интерфейсом из модели.

ViewModelхранит данные, связанные с приложением, которые не уничтожаются, когда действие уничтожается и воссоздается платформой Android. В отличие от экземпляра активности, ViewModelобъекты не уничтожаются. Приложение автоматически сохраняет ViewModelобъекты во время изменений конфигурации, поэтому содержащиеся в них данные становятся доступными сразу после рекомпозиции.

Чтобы реализовать это ViewModelв своем приложении, расширьте ViewModelкласс, который берется из библиотеки компонентов архитектуры и хранит данные приложения внутри этого класса.

### **Состояние пользовательского интерфейса**

Пользовательский интерфейс — это то, что видит пользователь, а состояние пользовательского интерфейса — это то, что приложение говорит, что он должен видеть. Пользовательский интерфейс — это визуальное представление состояния пользовательского интерфейса. Любые изменения состояния пользовательского интерфейса немедленно отражаются в пользовательском интерфейсе.



Пользовательский интерфейс — это результат привязки элементов пользовательского интерфейса на экране к состоянию пользовательского интерфейса.

// Example of UI state definition, do not copy over  
  
data class NewsItemUiState(  
    val title: String,  
    val body: String,  
    val bookmarked: Boolean = false,  
    ...  
)

### **Неизменяемость**

Определение состояния пользовательского интерфейса в приведенном выше примере является неизменяемым. Неизменяемые объекты гарантируют, что несколько источников не изменят состояние приложения в любой момент времени. Эта защита позволяет пользовательскому интерфейсу сосредоточиться на единственной роли: чтении состояния и соответствующем обновлении элементов пользовательского интерфейса. Поэтому никогда не следует изменять состояние пользовательского интерфейса непосредственно в пользовательском интерфейсе, если только сам пользовательский интерфейс не является единственным источником его данных. Нарушение этого принципа приводит к появлению нескольких источников достоверной информации для одной и той же информации, что приводит к несогласованности данных и тонким ошибкам.

[**Назад**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state)

## [**5. Добавьте модель представления**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#4)

В этой задаче вы добавляете ViewModelв свое приложение элемент для хранения состояния пользовательского интерфейса игры (зашифрованное слово, количество слов и счет). Чтобы решить проблему в стартовом коде, которую вы заметили в предыдущем разделе, вам необходимо сохранить данные игры в формате ViewModel.

1. Откройте build.gradle.kts (Module :app), прокрутите до dependenciesблока и добавьте следующую зависимость для ViewModel. Эта зависимость используется для добавления модели представления с учетом жизненного цикла в ваше приложение для создания сообщений.

dependencies {  
// other dependencies  
  
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.6.1")  
//...  
}

1. В uiпакете создайте класс/файл Kotlin с именем GameViewModel. Расширьте его из ViewModelкласса.

import androidx.lifecycle.ViewModel  
  
class GameViewModel : ViewModel() {  
}

1. В uiпакет добавьте класс модели для пользовательского интерфейса состояния под названием GameUiState. Сделайте это классом данных и добавьте переменную для текущего зашифрованного слова.

data class GameUiState(  
   val currentScrambledWord: String = ""  
)

## **StateFlow**

[StateFlow](https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-state-flow/)— это наблюдаемый поток держателя данных, который генерирует текущие и новые обновления состояния. Его [value](https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-state-flow/value.html" \t "_blank)свойство отражает текущее значение состояния. Чтобы обновить состояние и отправить его в поток, присвойте новое значение свойству value класса [MutableStateFlow](https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-mutable-state-flow/index.html" \t "_blank).

В Android StateFlowхорошо работает с классами, которые должны поддерживать наблюдаемое неизменяемое состояние.

A StateFlowможет быть предоставлен из GameUiState, чтобы составные элементы могли прослушивать обновления состояния пользовательского интерфейса и обеспечивать сохранение состояния экрана при изменении конфигурации.

Добавьте в GameViewModelкласс следующее \_uiStateсвойство.

import kotlinx.coroutines.flow.MutableStateFlow  
  
// Game UI state  
private val \_uiState = MutableStateFlow(GameUiState())

## **Поддержка собственности**

Резервное свойство позволяет вам вернуть из метода получения что-то, кроме конкретного объекта.

Для varсвойства платформа Kotlin генерирует геттеры и сеттеры.

Для методов получения и установки вы можете переопределить один или оба этих метода и обеспечить свое собственное поведение. Чтобы реализовать резервное свойство, вы переопределяете метод получения, чтобы вернуть версию ваших данных, доступную только для чтения. В следующем примере показано резервное свойство:

//Example code, no need to copy over  
  
// 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

В качестве другого примера предположим, что вы хотите, чтобы данные приложения были конфиденциальными для ViewModel:

Внутри ViewModelкласса:

* Свойство \_countявляется privateизменяемым. Следовательно, он доступен и доступен для редактирования только внутри ViewModelкласса.

Вне ViewModelкласса:

* Модификатор видимости по умолчанию в Kotlin — public, поэтому countон общедоступен и доступен из других классов, таких как контроллеры пользовательского интерфейса. Тип valне может иметь сеттер. Он неизменяем и доступен только для чтения, поэтому вы можете только переопределить метод get(). Когда внешний класс обращается к этому свойству, он возвращает значение, \_countи его значение нельзя изменить. Это резервное свойство защищает данные приложения внутри ViewModelот нежелательных и небезопасных изменений со стороны внешних классов, но позволяет внешним вызывающим сторонам безопасно получать доступ к его значению.

1. В GameViewModel.ktфайле добавьте резервное свойство в uiStateименованный \_uiState. Назовите свойство uiStateи имеет тип StateFlow<GameUiState>.

Теперь \_uiStateдоступно и редактируется только внутри файла GameViewModel. Пользовательский интерфейс может прочитать свое значение, используя свойство только для чтения, uiState. Исправить ошибку инициализации можно на следующем шаге.

import kotlinx.coroutines.flow.StateFlow  
  
// Game UI state  
  
// Backing property to avoid state updates from other classes  
private val \_uiState = MutableStateFlow(GameUiState())  
val uiState: StateFlow<GameUiState>

1. Установлен uiStateв \_uiState.asStateFlow().

Это asStateFlow()делает этот изменяемый поток состояний потоком состояний, доступным только для чтения .

import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.flow.asStateFlow  
  
// Game UI state  
private val \_uiState = MutableStateFlow(GameUiState())  
val uiState: StateFlow<GameUiState> = \_uiState.asStateFlow()

## **Отображать случайное зашифрованное слово**

В этой задаче вы добавляете вспомогательные методы для выбора случайного слова WordsData.ktи его шифрования.

1. В файле GameViewModelдобавьте свойство currentWordтипа Stringдля сохранения текущего зашифрованного слова.

private lateinit var currentWord: String

1. Добавьте вспомогательный метод, чтобы выбрать случайное слово из списка и перетасовать его. Назовите его pickRandomWordAndShuffle()без входных параметров и заставьте его возвращать расширение String.

import com.example.unscramble.data.allWords  
  
private fun pickRandomWordAndShuffle(): String {  
   // Continue picking up a new random word until you get one that hasn't been used before  
   currentWord = allWords.random()  
   if (usedWords.contains(currentWord)) {  
       return pickRandomWordAndShuffle()  
   } else {  
       usedWords.add(currentWord)  
       return shuffleCurrentWord(currentWord)  
   }  
}

Студия Android отмечает ошибку для неопределенной переменной и функции.

1. В файле GameViewModelдобавьте следующее свойство после currentWordсвойства, которое будет служить изменяемым набором для хранения использованных слов в игре.

// Set of words used in the game  
private var usedWords: MutableSet<String> = mutableSetOf()

1. Добавьте еще один вспомогательный метод для перетасовки текущего слова, shuffleCurrentWord()который принимает a Stringи возвращает перетасованное значение String.

private fun shuffleCurrentWord(word: String): String {  
   val tempWord = word.toCharArray()  
   // Scramble the word  
   tempWord.shuffle()  
   while (String(tempWord).equals(word)) {  
       tempWord.shuffle()  
   }  
   return String(tempWord)  
}

1. Добавьте вспомогательную функцию для инициализации игры под названием resetGame(). Вы используете эту функцию позже, чтобы запустить и перезапустить игру. В этой функции очистите все слова в usedWordsнаборе, инициализируйте файл \_uiState. Подберите новое слово для currentScrambledWordиспользования pickRandomWordAndShuffle().

fun resetGame() {  
   usedWords.clear()  
   \_uiState.value = GameUiState(currentScrambledWord = pickRandomWordAndShuffle())  
}

1. Добавьте initблок в GameViewModelи вызовите resetGame()из него.

init {  
   resetGame()  
}

Когда вы создадите свое приложение сейчас, вы по-прежнему не увидите никаких изменений в пользовательском интерфейсе. Вы не передаете данные из ViewModelсоставных элементов в файле GameScreen.

## [**6. Разработка пользовательского интерфейса Compose**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#5)

В Compose единственный способ обновить пользовательский интерфейс — изменить состояние приложения. Что вы можете контролировать, так это состояние вашего пользовательского интерфейса. Каждый раз, когда состояние пользовательского интерфейса изменяется, Compose воссоздает изменившиеся части дерева пользовательского интерфейса. Composables может принимать состояние и предоставлять события. Например, TextField/ OutlinedTextFieldпринимает значение и предоставляет обратный вызов onValueChange, который запрашивает обработчик обратного вызова изменить значение.

//Example code no need to copy over  
  
var name by remember { mutableStateOf("") }  
OutlinedTextField(  
    value = name,  
    onValueChange = { name = it },  
    label = { Text("Name") }  
)

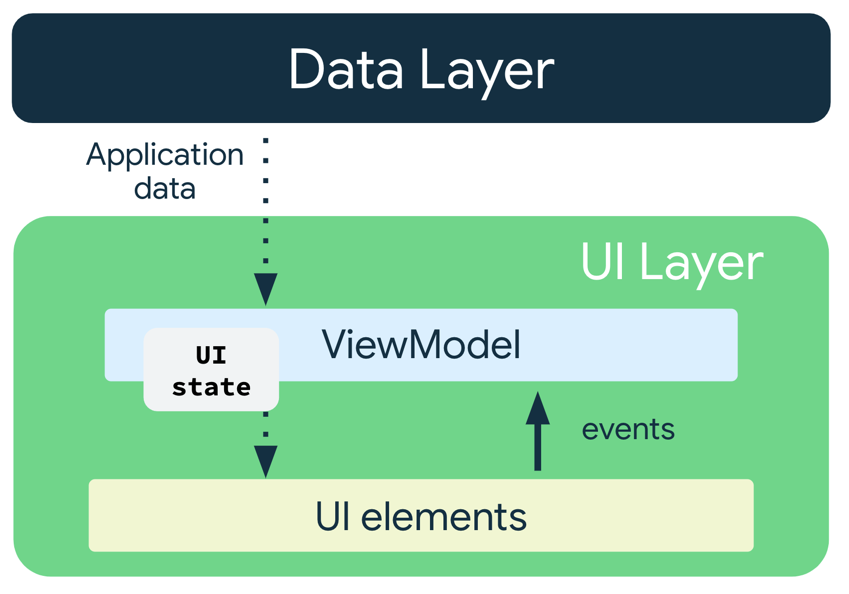
Поскольку составные объекты принимают состояние и предоставляют события, шаблон однонаправленного потока данных хорошо подходит для Jetpack Compose. В этом разделе основное внимание уделяется тому, как реализовать шаблон однонаправленного потока данных в Compose, как реализовать события и держатели состояний, а также как работать с ViewModels в Compose.

## **Однонаправленный поток данных**

Однонаправленный поток данных (UDF) — это шаблон проектирования, в котором состояние течет вниз, а события — вверх. Следуя однонаправленному потоку данных, вы можете отделить составные элементы, которые отображают состояние в пользовательском интерфейсе, от частей вашего приложения, которые сохраняют и изменяют состояние.

Цикл обновления пользовательского интерфейса для приложения, использующего однонаправленный поток данных, выглядит следующим образом:

* **Событие:** часть пользовательского интерфейса генерирует событие и передает его вверх — например, нажатие кнопки передается в ViewModel для обработки — или событие, которое передается из других слоев вашего приложения, например, указание на то, что срок сеанса пользователя истек.
* **Состояние обновления:** обработчик событий может изменить состояние.
* **Состояние отображения:** держатель состояния передает состояние, и пользовательский интерфейс отображает его.



Использование шаблона UDF для архитектуры приложения имеет следующие последствия:

* Сохраняет ViewModelи предоставляет состояние, которое использует пользовательский интерфейс.
* Состояние пользовательского интерфейса — это данные приложения, преобразованные с помощью ViewModel.
* Пользовательский интерфейс уведомляет ViewModelо пользовательских событиях.
* Обрабатывает ViewModelдействия пользователя и обновляет состояние.
* Обновленное состояние передается обратно в пользовательский интерфейс для рендеринга.
* Этот процесс повторяется для любого события, вызывающего изменение состояния.

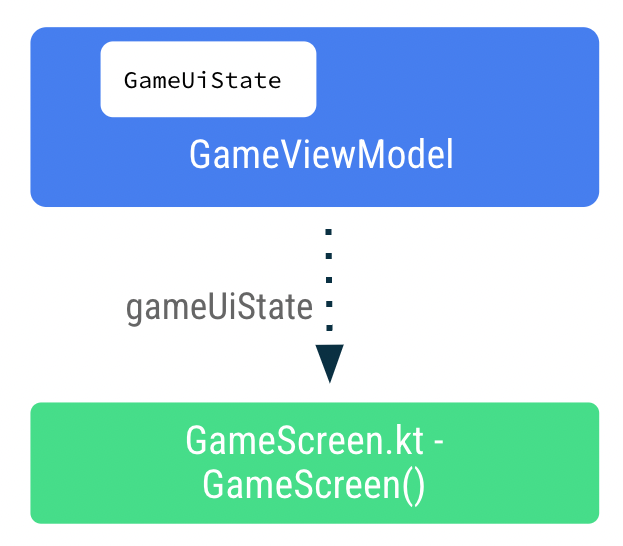
## **Передайте данные**

Передайте экземпляр ViewModel в пользовательский интерфейс, то есть из файла GameViewModelв GameScreen()файл GameScreen.kt. Воспользуйтесь GameScreen()экземпляром ViewModel для доступа к uiStateиспользованию collectAsState().

Функция collectAsState()собирает значения из этого [StateFlow](https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-state-flow/index.html" \t "_blank)и представляет свое последнее значение через [State](https://developer.android.com/reference/kotlin/androidx/compose/runtime/State). Используется [StateFlow.value](https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-state-flow/value.html" \t "_blank)в качестве начального значения. Каждый раз, когда в , будет отправлено новое значение [StateFlow](https://kotlin.github.io/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-state-flow/index.html" \t "_blank), будут возвращены [State](https://developer.android.com/reference/kotlin/androidx/compose/runtime/State" \t "_blank)обновления, что приведет к рекомпозиции каждого [State.value](https://developer.android.com/reference/kotlin/androidx/compose/runtime/State" \l "value()" \t "_blank)использования.

1. В GameScreenфункции передайте второй аргумент типа GameViewModelсо значением по умолчанию viewModel().

import androidx.lifecycle.viewmodel.compose.viewModel  
  
@Composable  
fun GameScreen(  
   gameViewModel: GameViewModel = viewModel()  
) {  
   // ...  
}



1. В GameScreen()функцию добавьте новую переменную с именем gameUiState. Используйте byделегата и collectAsState()вызовите uiState.

Этот подход гарантирует, что всякий раз, когда происходит изменение значения uiState, происходит перекомпозиция компонуемых объектов, использующих это gameUiStateзначение.

import androidx.compose.runtime.collectAsState  
import androidx.compose.runtime.getValue  
  
@Composable  
fun GameScreen(  
   // ...  
) {  
   val gameUiState by gameViewModel.uiState.collectAsState()  
   // ...  
}

1. Передайте gameUiState.currentScrambledWordкомпонуемому GameLayout(). Вы добавите аргумент на более позднем этапе, поэтому пока игнорируйте ошибку.

GameLayout(  
   currentScrambledWord = gameUiState.currentScrambledWord,  
   modifier = Modifier  
       .fillMaxWidth()  
       .wrapContentHeight()  
       .padding(mediumPadding)  
)

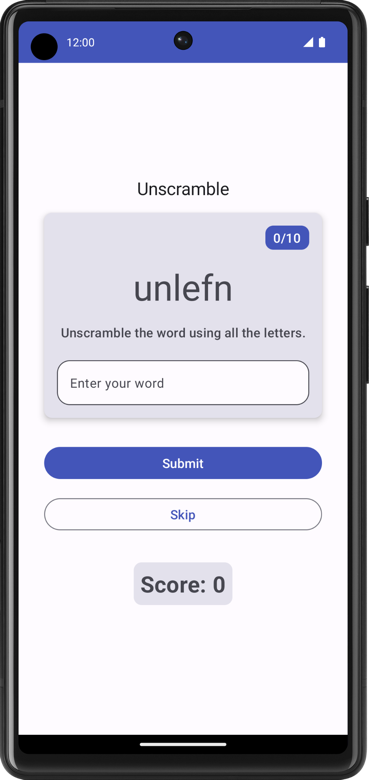
1. Добавьте currentScrambledWordеще один параметр в GameLayout()составную функцию.

@Composable  
fun GameLayout(  
   currentScrambledWord: String,  
   modifier: Modifier = Modifier  
) {  
}

1. Обновите GameLayout()составную функцию для отображения currentScrambledWord. Установите textпараметр первого текстового поля в столбце на currentScrambledWord.

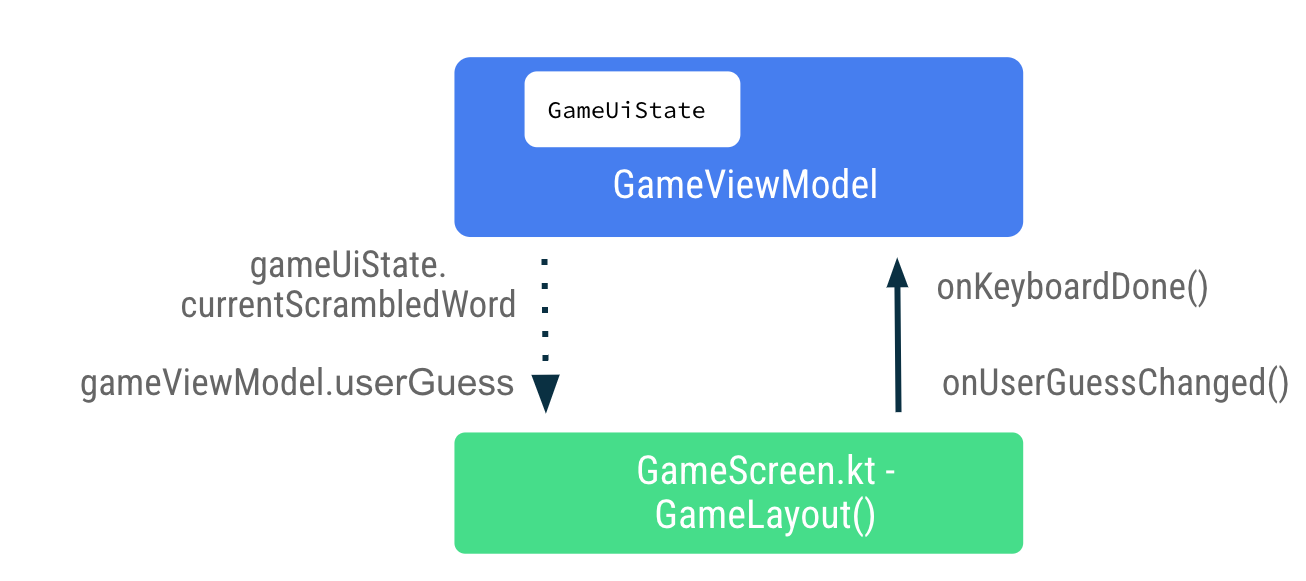
@Composable  
fun GameLayout(  
   // ...  
) {  
   Column(  
       verticalArrangement = Arrangement.spacedBy(24.dp)  
   ) {  
       Text(  
           text = currentScrambledWord,  
           fontSize = 45.sp,  
           modifier = modifier.align(Alignment.CenterHorizontally)  
       )  
    //...   
    }  
}

1. Запустите и создайте приложение. Вы должны увидеть зашифрованное слово.



## **Отображение загаданного слова**

В GameLayout()составном компоненте обновление предполагаемого слова пользователя является одним из обратных вызовов событий, которые передаются из GameScreenфайла ViewModel. Данные gameViewModel.userGuessбудут перетекать из файла ViewModelв файл GameScreen.



1. В GameScreen.ktфайле, в GameLayout()составном элементе, установлены onValueChangeи onUserGuessChangedдействия onKeyboardDone()клавиатуры onDone. Вы исправите ошибки на следующем шаге.

OutlinedTextField(  
   value = "",  
   singleLine = true,  
   modifier = Modifier.fillMaxWidth(),  
   onValueChange = onUserGuessChanged,  
   label = { Text(stringResource(R.string.enter\_your\_word)) },  
   isError = false,  
   keyboardOptions = KeyboardOptions.Default.copy(  
       imeAction = ImeAction.Done  
   ),  
   keyboardActions = KeyboardActions(  
       onDone = { onKeyboardDone() }  
   ),

1. В GameLayout()составную функцию добавьте еще два аргумента: onUserGuessChangedлямбда принимает Stringаргумент и ничего не возвращает, а лямбда onKeyboardDoneничего не принимает и ничего не возвращает.

@Composable  
fun GameLayout(  
   onUserGuessChanged: (String) -> Unit,  
   onKeyboardDone: () -> Unit,  
   currentScrambledWord: String,  
   modifier: Modifier = Modifier,  
   ) {  
}

1. В GameLayout()вызове функции добавьте лямбда-аргументы для onUserGuessChangedи onKeyboardDone.

GameLayout(  
   onUserGuessChanged = { gameViewModel.updateUserGuess(it) },  
   onKeyboardDone = { },  
   currentScrambledWord = gameUiState.currentScrambledWord,  
)

Вскоре вы определите updateUserGuessметод GameViewModel.

1. В GameViewModel.ktфайле добавьте метод, updateUserGuess()который принимает Stringаргумент — предполагаемое слово пользователя. Внутри функции обновите userGuessпереданный файл guessedWord.

  fun updateUserGuess(guessedWord: String){  
     userGuess = guessedWord  
  }

Далее вы добавляете userGuessViewModel.

1. В GameViewModel.ktфайл добавьте свойство var с именем userGuess. Используйте mutableStateOf(), чтобы Compose отслеживал это значение и устанавливал начальное значение "".

import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.setValue  
  
  
var userGuess by mutableStateOf("")  
   private set

1. В GameScreen.ktфайле внутри GameLayout()добавьте еще один Stringпараметр для userGuess. Установите параметр valueна OutlinedTextField.userGuess

fun GameLayout(  
   currentScrambledWord: String,  
   userGuess: String,  
   onUserGuessChanged: (String) -> Unit,  
   onKeyboardDone: () -> Unit,  
   modifier: Modifier = Modifier  
) {  
   Column(  
       verticalArrangement = Arrangement.spacedBy(24.dp)  
   ) {  
       //...  
       OutlinedTextField(  
           value = userGuess,  
           //..  
       )  
   }  
}

1. В GameScreenфункции обновите GameLayout()вызов функции, включив в него userGuessпараметр.

GameLayout(  
   currentScrambledWord = gameUiState.currentScrambledWord,  
   userGuess = gameViewModel.userGuess,  
   onUserGuessChanged = { gameViewModel.updateUserGuess(it) },  
   onKeyboardDone = { },  
   //...  
)

1. Создайте и запустите свое приложение.
2. Попробуйте угадать и ввести слово. Текстовое поле может отображать предположение пользователя.



## [**7. Проверьте угаданное слово и обновите оценку.**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#6)

В этой задаче вы реализуете метод проверки слова, угаданного пользователем, а затем либо обновляете счет игры, либо отображаете ошибку. Позже вы обновите пользовательский интерфейс состояния игры, добавив новый счет и новое слово.

1. В GameViewModelдобавьте еще один метод под названием checkUserGuess().
2. В checkUserGuess()функцию добавьте if elseблок, чтобы проверить, совпадает ли предположение пользователя с предположением currentWord. Сброс userGuessдо пустой строки.

fun checkUserGuess() {  
     
   if (userGuess.equals(currentWord, ignoreCase = true)) {  
   } else {  
   }  
   // Reset user guess  
   updateUserGuess("")  
}

1. Если предположение пользователя неверно, установите isGuessedWordWrongзначение true. обновляет, используя указанное значение.[MutableStateFlow<T>.](https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-mutable-state-flow/index.html) [update()](https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/update.html" \t "_blank)[MutableStateFlow.value](https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines.flow/-mutable-state-flow/value.html)

import kotlinx.coroutines.flow.update  
  
   if (userGuess.equals(currentWord, ignoreCase = true)) {  
   } else {  
       // User's guess is wrong, show an error  
       \_uiState.update { currentState ->  
           currentState.copy(isGuessedWordWrong = true)  
       }  
   }

**Примечание о методе copy():** используйте эту copy()функцию для копирования объекта, что позволяет вам изменять некоторые его свойства, сохраняя при этом остальные неизменными.

Пример:

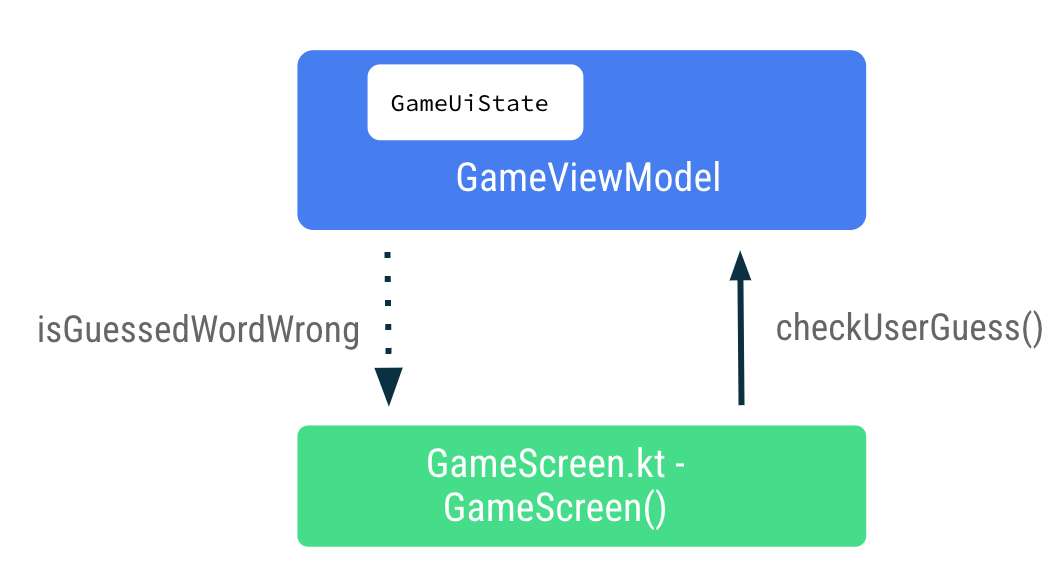
val jack = User(name = "Jack", age = 1)

val olderJack = jack.copy(age = 2)

1. В GameUiStateклассе добавьте Booleanвызываемый объект isGuessedWordWrongи инициализируйте его как false.

data class GameUiState(  
   val currentScrambledWord: String = "",  
   val isGuessedWordWrong: Boolean = false,  
)

Затем вы передаете обратный вызов события checkUserGuess()от GameScreenдо ViewModel, когда пользователь нажимает кнопку **«Отправить»** или клавишу «Готово» на клавиатуре. Передайте данные gameUiState.isGuessedWordWrongвниз от до, ViewModelчтобы GameScreenустановить ошибку в текстовом поле.



1. В GameScreen.ktфайле в конце GameScreen()составной функции вызовите gameViewModel.checkUserGuess()лямбда onClick-выражение кнопки **«Отправить»** .

Button(  
   modifier = modifier  
       .fillMaxWidth()  
       .weight(1f)  
       .padding(start = 8.dp),  
   onClick = { gameViewModel.checkUserGuess() }  
) {  
   Text(stringResource(R.string.submit))  
}

1. В GameScreen()составной функции обновите GameLayout()вызов функции, чтобы передать gameViewModel.checkUserGuess()лямбда onKeyboardDone-выражение.

GameLayout(  
   currentScrambledWord = gameUiState.currentScrambledWord,  
   userGuess = gameViewModel.userGuess,  
   onUserGuessChanged = { gameViewModel.updateUserGuess(it) },  
   onKeyboardDone = { gameViewModel.checkUserGuess() }  
)

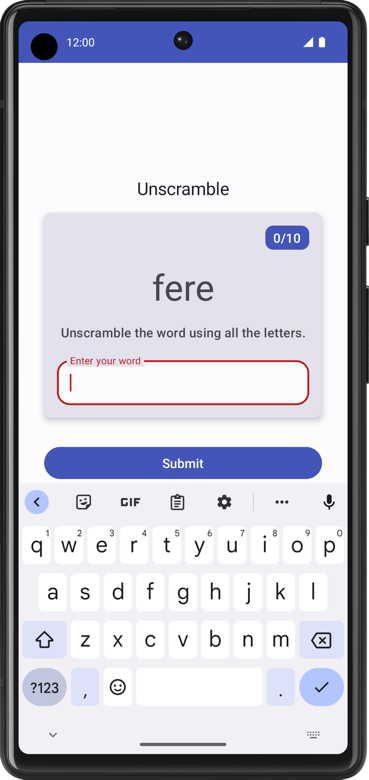
1. В GameLayout()составную функцию добавьте параметр функции для Boolean, isGuessWrong. Установите isErrorпараметр to, OutlinedTextFieldчтобы isGuessWrongотображать ошибку в текстовом поле, если предположение пользователя неверно.

fun GameLayout(  
   currentScrambledWord: String,  
   isGuessWrong: Boolean,  
   userGuess: String,  
   onUserGuessChanged: (String) -> Unit,  
   onKeyboardDone: () -> Unit,  
   modifier: Modifier = Modifier  
) {  
   Column(  
       // ,...  
       OutlinedTextField(  
           // ...  
           isError = isGuessWrong,  
           keyboardOptions = KeyboardOptions.Default.copy(  
               imeAction = ImeAction.Done  
           ),  
           keyboardActions = KeyboardActions(  
               onDone = { onKeyboardDone() }  
           ),  
       )  
}  
}

1. В GameScreen()составной функции обновите GameLayout()вызов функции, чтобы передать его isGuessWrong.

GameLayout(  
   currentScrambledWord = gameUiState.currentScrambledWord,  
   userGuess = gameViewModel.userGuess,  
   onUserGuessChanged = { gameViewModel.updateUserGuess(it) },  
   onKeyboardDone = { gameViewModel.checkUserGuess() },  
   isGuessWrong = gameUiState.isGuessedWordWrong,  
   // ...  
)

1. Создайте и запустите свое приложение.
2. Введите неверное предположение и нажмите **«Отправить»** . Обратите внимание, что текстовое поле становится красным, что указывает на ошибку.



Обратите внимание, что метка текстового поля по-прежнему гласит: «Введите слово». Чтобы сделать его удобным для пользователя, вам нужно добавить текст ошибки, указывающий, что слово неверно.

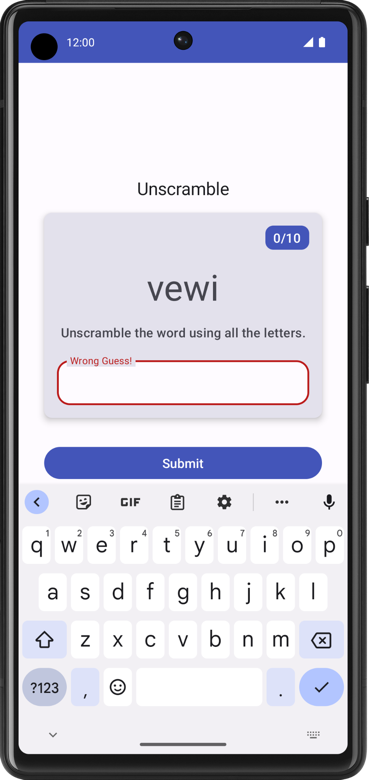
1. В GameScreen.ktфайле в GameLayout()компонуемом обновите параметр метки текстового поля в зависимости от isGuessWrongследующего:

OutlinedTextField(  
   // ...  
   label = {  
       if (isGuessWrong) {  
           Text(stringResource(R.string.wrong\_guess))  
       } else {  
           Text(stringResource(R.string.enter\_your\_word))  
       }  
   },  
   // ...  
)

1. В strings.xmlфайле добавьте строку к метке ошибки.

<string name="wrong\_guess">Wrong Guess!</string>

1. Создайте и снова запустите свое приложение.
2. Введите неверное предположение и нажмите **«Отправить»** . Обратите внимание на метку ошибки.



## [**8. Обновите счет и количество слов.**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#7)

В этой задаче вы обновляете счет и количество слов по мере того, как пользователь играет в игру. Оценка должна быть частью \_ uiState.

1. В GameUiStateдобавьте scoreпеременную и инициализируйте ее нулем.

data class GameUiState(  
   val currentScrambledWord: String = "",  
   val isGuessedWordWrong: Boolean = false,  
   val score: Int = 0  
)

1. Чтобы обновить значение оценки, GameViewModelв checkUserGuess()функции внутри ifусловия, когда предположение пользователя верно, увеличьте scoreзначение.

import com.example.unscramble.data.SCORE\_INCREASE  
  
fun checkUserGuess() {  
   if (userGuess.equals(currentWord, ignoreCase = true)) {  
       // User's guess is correct, increase the score  
       val updatedScore = \_uiState.value.score.plus(SCORE\_INCREASE)  
   } else {  
       //...  
   }  
}

1. В GameViewModelдобавьте еще один метод, вызываемый updateGameStateдля обновления оценки, увеличения текущего количества слов и выбора нового слова из WordsData.ktфайла. Добавьте Intимя updatedScoreв качестве параметра. Обновите переменные пользовательского интерфейса состояния игры следующим образом:

private fun updateGameState(updatedScore: Int) {  
   \_uiState.update { currentState ->  
       currentState.copy(  
           isGuessedWordWrong = false,  
           currentScrambledWord = pickRandomWordAndShuffle(),  
           score = updatedScore  
       )  
   }  
}

1. В checkUserGuess()функции, если предположение пользователя верно, выполните вызов updateGameStateс обновленным счетом, чтобы подготовить игру к следующему раунду.

fun checkUserGuess() {  
   if (userGuess.equals(currentWord, ignoreCase = true)) {  
       // User's guess is correct, increase the score  
       // and call updateGameState() to prepare the game for next round  
       val updatedScore = \_uiState.value.score.plus(SCORE\_INCREASE)  
       updateGameState(updatedScore)  
   } else {  
       //...  
   }  
}

Завершенное checkUserGuess()должно выглядеть следующим образом:

fun checkUserGuess() {  
   if (userGuess.equals(currentWord, ignoreCase = true)) {  
       // User's guess is correct, increase the score  
       // and call updateGameState() to prepare the game for next round  
       val updatedScore = \_uiState.value.score.plus(SCORE\_INCREASE)  
       updateGameState(updatedScore)  
   } else {  
       // User's guess is wrong, show an error  
       \_uiState.update { currentState ->  
           currentState.copy(isGuessedWordWrong = true)  
       }  
   }  
   // Reset user guess  
   updateUserGuess("")  
}

Далее, как и при обновлении партитуры, вам необходимо обновить количество слов.

1. Добавьте еще одну переменную для счетчика в файле GameUiState. Вызовите его currentWordCountи инициализируйте его как 1.

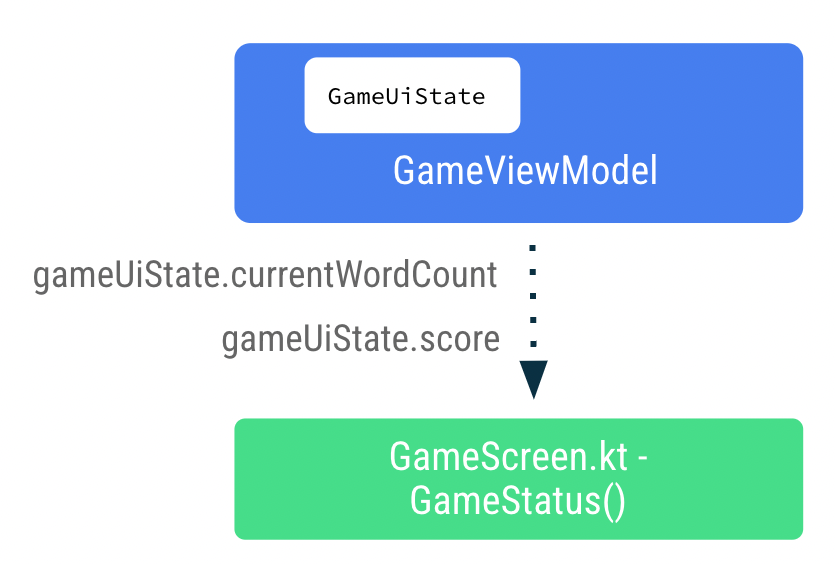
data class GameUiState(  
   val currentScrambledWord: String = "",  
   val currentWordCount: Int = 1,  
   val score: Int = 0,  
   val isGuessedWordWrong: Boolean = false,  
)

1. В GameViewModel.ktфайле в updateGameState()функции увеличьте количество слов, как показано ниже. Функция updateGameState()вызывается для подготовки игры к следующему раунду.

private fun updateGameState(updatedScore: Int) {  
   \_uiState.update { currentState ->  
       currentState.copy(  
           //...  
           currentWordCount = currentState.currentWordCount.inc(),  
           )  
   }  
}

## **Проходной балл и количество слов**

Выполните следующие шаги, чтобы передать данные о счете и количестве слов из ViewModelфайла GameScreen.



1. В GameScreen.ktфайле в GameLayout()составной функции добавьте количество слов в качестве аргумента и передайте wordCountаргументы формата текстовому элементу.

fun GameLayout(  
   onUserGuessChanged: (String) -> Unit,  
   onKeyboardDone: () -> Unit,  
   wordCount: Int,  
   //...  
) {  
   //...  
  
   Card(  
       //...  
   ) {  
       Column(  
           // ...  
       ) {  
           Text(  
               //..  
               text = stringResource(R.string.word\_count, wordCount),  
               style = typography.titleMedium,  
               color = colorScheme.onPrimary  
           )  
  
  
// ...  
  
}

1. Обновите GameLayout()вызов функции, включив в него количество слов.

GameLayout(  
   userGuess = gameViewModel.userGuess,  
   wordCount = gameUiState.currentWordCount,  
   //...  
)

1. В GameScreen()составной функции обновите GameStatus()вызов функции, включив в него score параметры. Передайте счет из gameUiState.

GameStatus(score = gameUiState.score, modifier = Modifier.padding(20.dp))

1. Создайте и запустите приложение.
2. Введите предполагаемое слово и нажмите **«Отправить»** . Обратите внимание на обновление оценки и количества слов.
3. Нажмите **«Пропустить»** и обратите внимание, что ничего не происходит.

Чтобы реализовать функцию пропуска, вам необходимо передать обратный вызов события пропуска в метод GameViewModel.

1. В GameScreen.ktфайле в GameScreen()составной функции вызовите gameViewModel.skipWord()лямбда onClick-выражение.

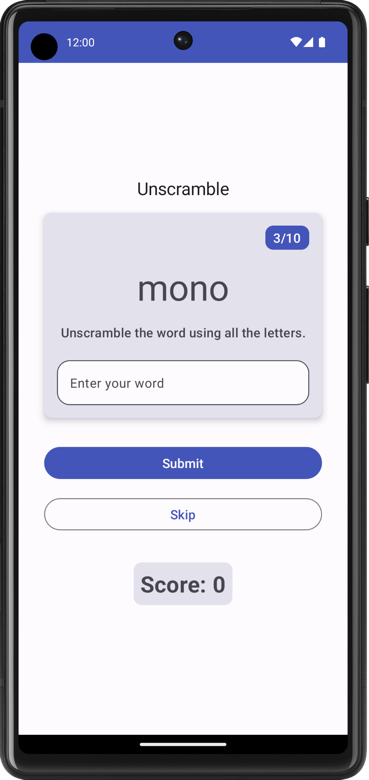
Android Studio отображает ошибку, поскольку вы еще не реализовали эту функцию. Эту ошибку можно исправить на следующем шаге, добавив метод skipWord() . Когда пользователь пропускает слово, вам необходимо обновить игровые переменные и подготовить игру к следующему раунду.

OutlinedButton(  
   onClick = { gameViewModel.skipWord() },  
   modifier = Modifier.fillMaxWidth()  
) {  
   //...  
}

1. В GameViewModelдобавьте метод skipWord().
2. Внутри skipWord()функции выполните вызов updateGameState(), передав оценку и сбросив предположение пользователя.

fun skipWord() {  
   updateGameState(\_uiState.value.score)  
   // Reset user guess  
   updateUserGuess("")  
}

1. Запустите приложение и играйте в игру. Теперь вы сможете пропускать слова.



Вы все еще можете играть в игру, превышающую 10 слов. В следующем задании вам предстоит справиться с последним раундом игры.

## [**9. Проведите последний раунд игры.**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#8)

В текущей реализации пользователи могут пропускать или воспроизводить более 10 слов. В этом задании вы добавляете логику для завершения игры.



Чтобы реализовать логику завершения игры, сначала нужно проверить, набрал ли пользователь максимальное количество слов.

1. Добавьте GameViewModelблок if-elseи переместите существующее тело функции внутри elseблока.
2. Добавьте ifусловие, проверяющее, что usedWordsразмер равен MAX\_NO\_OF\_WORDS.

import com.example.android.unscramble.data.MAX\_NO\_OF\_WORDS  
  
  
private fun updateGameState(updatedScore: Int) {  
   if (usedWords.size == MAX\_NO\_OF\_WORDS){  
       //Last round in the game  
   } else{  
       // Normal round in the game  
       \_uiState.update { currentState ->  
           currentState.copy(  
               isGuessedWordWrong = false,  
               currentScrambledWord = pickRandomWordAndShuffle(),  
               currentWordCount = currentState.currentWordCount.inc(),  
               score = updatedScore  
           )  
       }  
   }  
}

1. Внутри ifблока добавьте Booleanфлаг isGameOverи установите его, чтобы trueуказать конец игры.
2. Обновите scoreи выполните сброс isGuessedWordWrongвнутри ifблока. Следующий код показывает, как должна выглядеть ваша функция:

private fun updateGameState(updatedScore: Int) {  
   if (usedWords.size == MAX\_NO\_OF\_WORDS){  
       //Last round in the game, update isGameOver to true, don't pick a new word  
       \_uiState.update { currentState ->  
           currentState.copy(  
               isGuessedWordWrong = false,  
               score = updatedScore,  
               isGameOver = true  
           )  
       }  
   } else{  
       // Normal round in the game  
       \_uiState.update { currentState ->  
           currentState.copy(  
               isGuessedWordWrong = false,  
               currentScrambledWord = pickRandomWordAndShuffle(),  
               currentWordCount = currentState.currentWordCount.inc(),  
               score = updatedScore  
           )  
       }  
   }  
}

1. В GameUiStateдобавьте Booleanпеременную isGameOverи установите для нее значение false.

data class GameUiState(  
   val currentScrambledWord: String = "",  
   val currentWordCount: Int = 1,  
   val score: Int = 0,  
   val isGuessedWordWrong: Boolean = false,  
   val isGameOver: Boolean = false  
)

1. Запустите приложение и играйте в игру. Вы не можете играть более 10 слов.



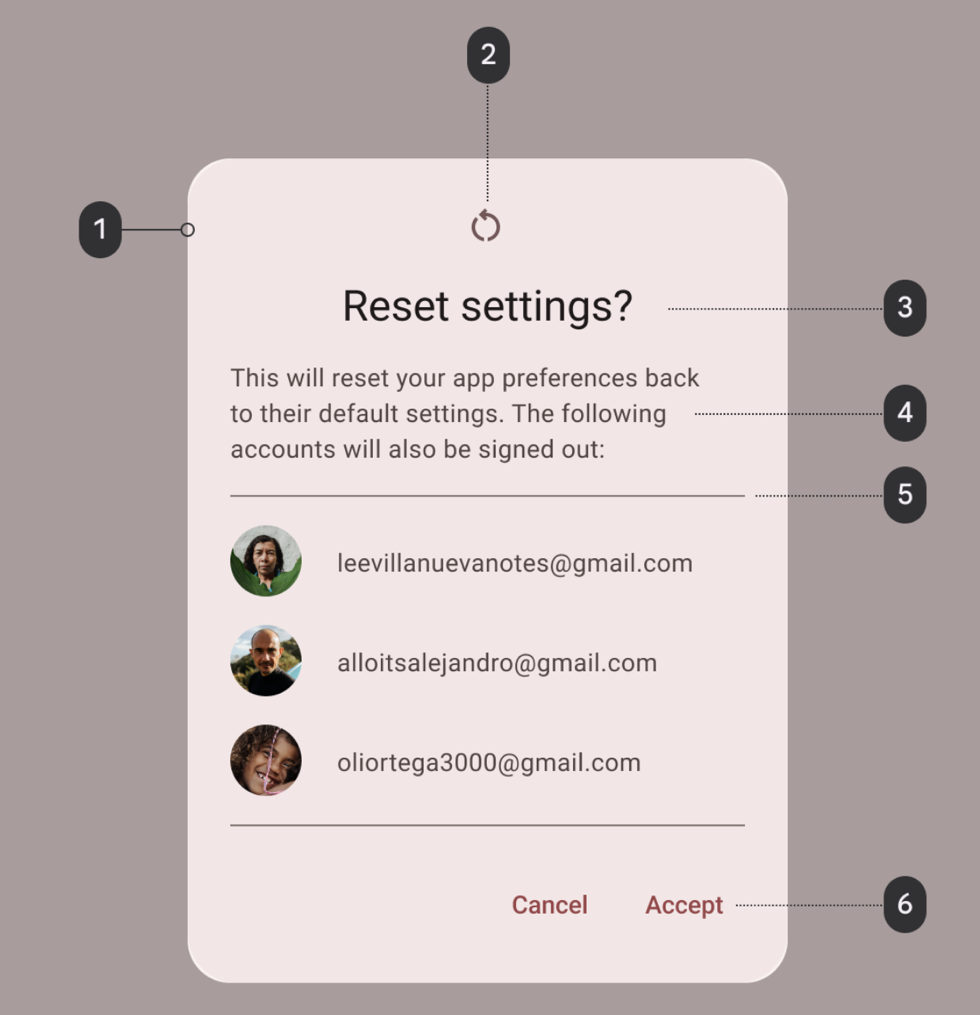
Когда игра закончится, было бы неплохо сообщить пользователю об этом и спросить, хочет ли он сыграть еще раз. Вы реализуете эту функцию в своей следующей задаче.

## **Отображение диалога окончания игры**

В этой задаче вы передаете isGameOverданные GameScreenиз ViewModel и используете их для отображения диалогового окна предупреждения с параметрами завершения или перезапуска игры.

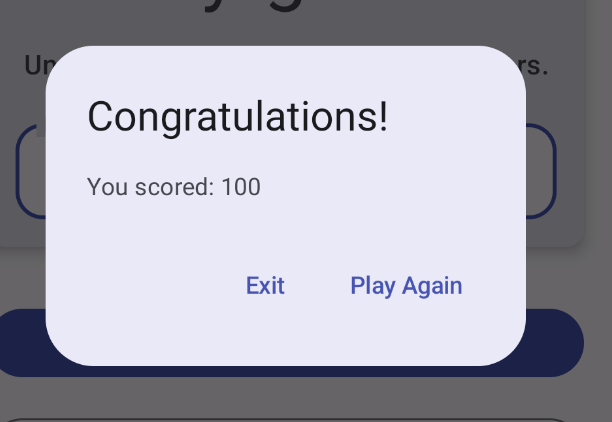
Диалог — это небольшое окно, предлагающее пользователю принять решение или ввести дополнительную информацию. Обычно диалоговое окно не занимает весь экран и требует от пользователей выполнения действия, прежде чем они смогут продолжить. Android предоставляет различные типы диалогов. В этой лаборатории вы узнаете о диалогах предупреждений.

### **Анатомия диалога оповещений**



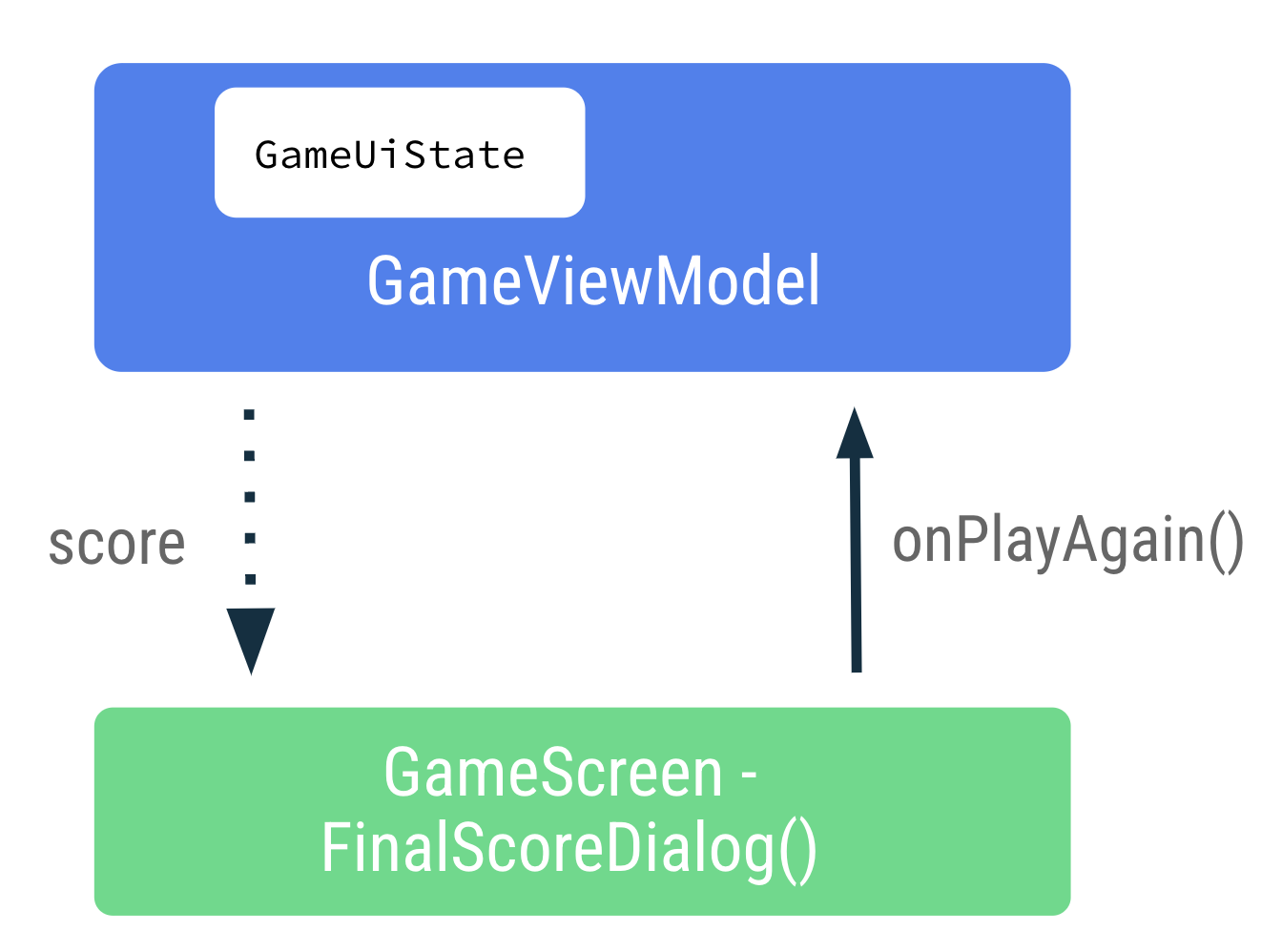
1. Контейнер
2. Значок (необязательно)
3. Заголовок (необязательно)
4. Сопроводительный текст
5. Разделитель (опционально)
6. Действия

Файл GameScreen.ktв стартовом коде уже предоставляет функцию, которая отображает диалоговое окно с опциями выхода или перезапуска игры.



@Composable  
private fun FinalScoreDialog(  
   onPlayAgain: () -> Unit,  
   modifier: Modifier = Modifier  
) {  
   val activity = (LocalContext.current as Activity)  
  
   AlertDialog(  
       onDismissRequest = {  
           // Dismiss the dialog when the user clicks outside the dialog or on the back  
           // button. If you want to disable that functionality, simply use an empty  
           // onDismissRequest.  
       },  
       title = { Text(stringResource(R.string.congratulations)) },  
       text = { Text(stringResource(R.string.you\_scored, 0)) },  
       modifier = modifier,  
       dismissButton = {  
           TextButton(  
               onClick = {  
                   activity.finish()  
               }  
           ) {  
               Text(text = stringResource(R.string.exit))  
           }  
       },  
       confirmButton = {  
           TextButton(  
               onClick = {  
                   onPlayAgain()  
               }  
           ) {  
               Text(text = stringResource(R.string.play\_again))  
           }  
       }  
   )  
}

В этой функции titleи textпараметрах отображается заголовок и вспомогательный текст в диалоговом окне оповещения. И dismissButton— confirmButtonэто текстовые кнопки. В dismissButtonпараметре вы отображаете текст « **Выход»** и завершаете приложение, завершив действие. В confirmButtonпараметре вы перезапускаете игру и отображаете текст **Play Again** .



1. В GameScreen.ktфайле в FinalScoreDialog()функции обратите внимание на параметр счета, чтобы отобразить счет игры в диалоговом окне оповещения.

@Composable  
private fun FinalScoreDialog(  
   score: Int,  
   onPlayAgain: () -> Unit,  
   modifier: Modifier = Modifier  
) {

1. FinalScoreDialog()Обратите внимание на использование textлямбда-выражения параметра в функции scoreв качестве аргумента формата текста диалога.

text = { Text(stringResource(R.string.you\_scored, score)) }

1. В GameScreen.ktфайле в конце GameScreen()составной функции, после Columnблока, добавьте ifусловие для проверки gameUiState.isGameOver.
2. В ifблоке отобразите диалог оповещения. Выполните вызов для FinalScoreDialog()передачи scoreи обратного вызова gameViewModel.resetGame()события onPlayAgain.

if (gameUiState.isGameOver) {  
   FinalScoreDialog(  
       score = gameUiState.score,  
       onPlayAgain = { gameViewModel.resetGame() }  
   )  
}

Это resetGame()обратный вызов события, который передается GameScreenот ViewModel.

1. В GameViewModel.ktфайле вызываете resetGame()функцию, инициализируете \_uiStateи выбираете новое слово.

fun resetGame() {  
   usedWords.clear()  
   \_uiState.value = GameUiState(currentScrambledWord = pickRandomWordAndShuffle())  
}

1. Создайте и запустите свое приложение.
2. Играйте в игру до конца и наблюдайте за диалоговым окном с опциями « **Выйти из** игры» или **«Играть снова»** . Попробуйте параметры, отображаемые в диалоговом окне предупреждения.



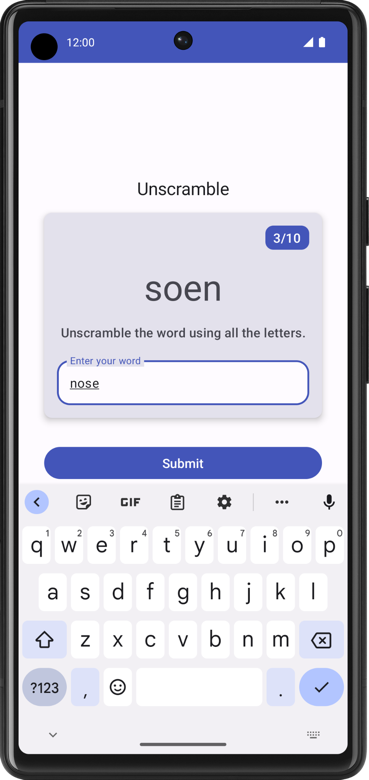
## [**10. Состояние при повороте устройства**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#9)

В предыдущих лабораторных работах вы узнали об изменениях конфигурации в Android. При изменении конфигурации Android перезапускает действие с нуля, выполняя все обратные вызовы запуска жизненного цикла.

В [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel" \t "_blank)нем хранятся данные, связанные с приложением, которые не уничтожаются, когда платформа Android уничтожает и воссоздает активность. ViewModelобъекты автоматически сохраняются и не уничтожаются, как экземпляр действия, во время изменения конфигурации. Содержащиеся в них данные становятся доступными сразу после рекомпозиции.

В этой задаче вы проверяете, сохраняет ли приложение состояние пользовательского интерфейса во время изменения конфигурации.

1. Запустите приложение и сыграйте несколько слов. Измените конфигурацию устройства с книжной на альбомную или наоборот.
2. Обратите внимание, что данные, сохраненные в ViewModelпользовательском интерфейсе состояния, сохраняются во время изменения конфигурации.





## [**12. Заключение**](https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-viewmodel-and-state#11)

Поздравляем! Вы завершили кодовую лабораторию. Теперь вы понимаете, как рекомендации по архитектуре приложений Android рекомендуют разделять классы с разными обязанностями и управлять пользовательским интерфейсом на основе модели.

Не забудьте поделиться своей работой в социальных сетях с хэштегом #AndroidBasics !

## **Узнать больше**

* [Руководство по архитектуре приложения | Android-разработчики](https://developer.android.com/topic/architecture)
* [слой пользовательского интерфейса | Android-разработчики](https://developer.android.com/topic/architecture/ui-layer)
* [Управляйте состоянием с помощью однонаправленного потока данных | Android-разработчики](https://developer.android.com/topic/architecture/ui-layer#udf)
* Путь обучения: [современная архитектура приложений для Android](https://developer.android.com/courses/pathways/android-architecture)