Ame FastForward

O que veremos hoje?

Agenda de hoje

- **Implementação de Testes no Android**
  - Criação dos Game: Jogo da Véia
  - Testes unitários
  - Testes de interface

Conhecendo nosso projeto

APP: Jogo da Velha

O **jogo da velha** ou **jogo do galo** como é conhecido em Portugal, é um jogo de regras extremamente simples, que não traz grandes dificuldades para seus jogadores e é facilmente aprendido.

O tabuleiro é uma matriz de três linhas por três colunas.

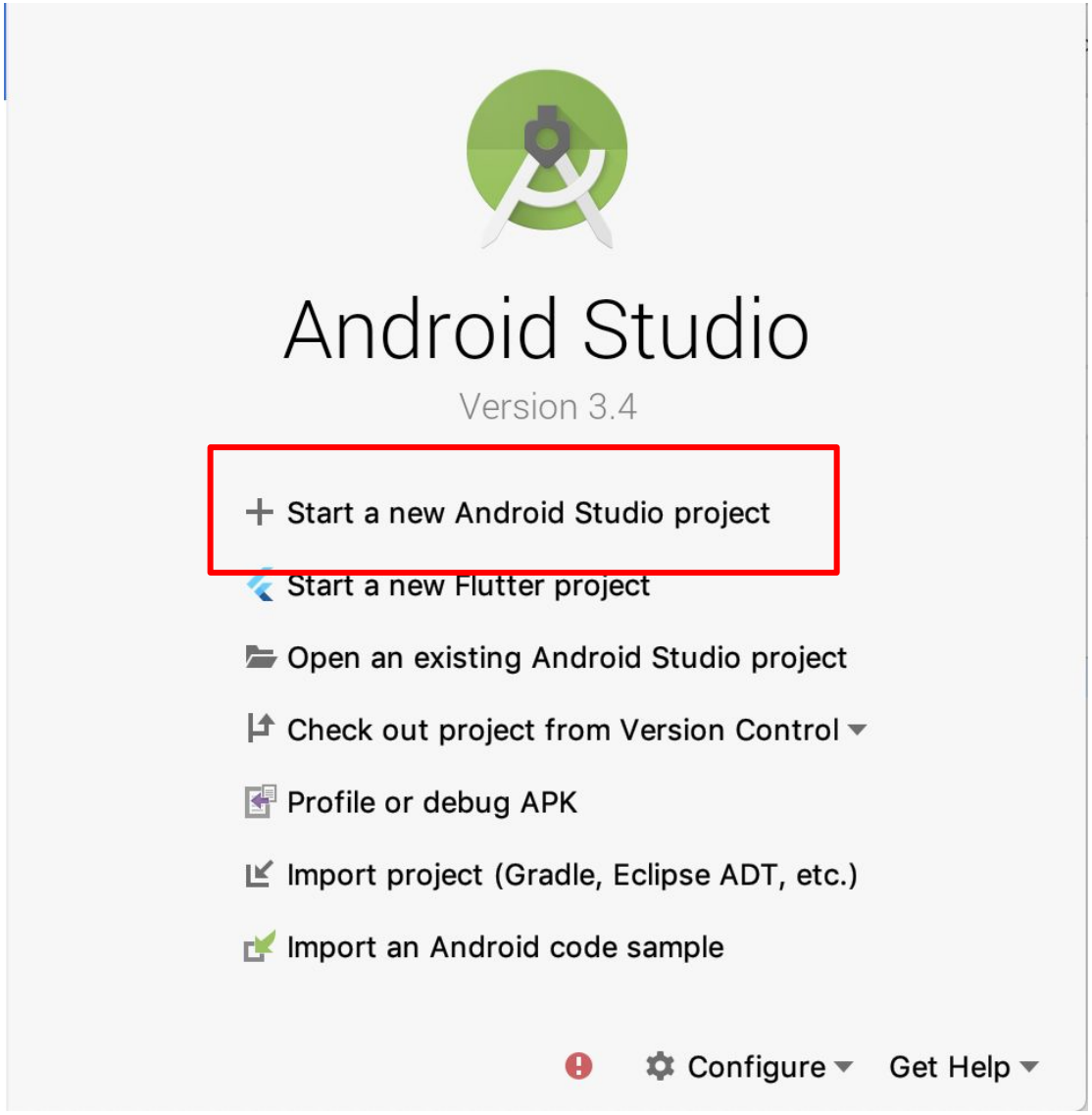
Dois jogadores escolhem uma marcação cada um, geralmente um círculo (O) e um xis (X).

Os jogadores jogam alternadamente, uma marcação por vez, numa lacuna que esteja vazia.

O objetivo é conseguir três círculos ou três xis em linha, quer horizontal, vertical ou diagonal , e ao mesmo tempo, quando possível, impedir o adversário de ganhar na próxima jogada.

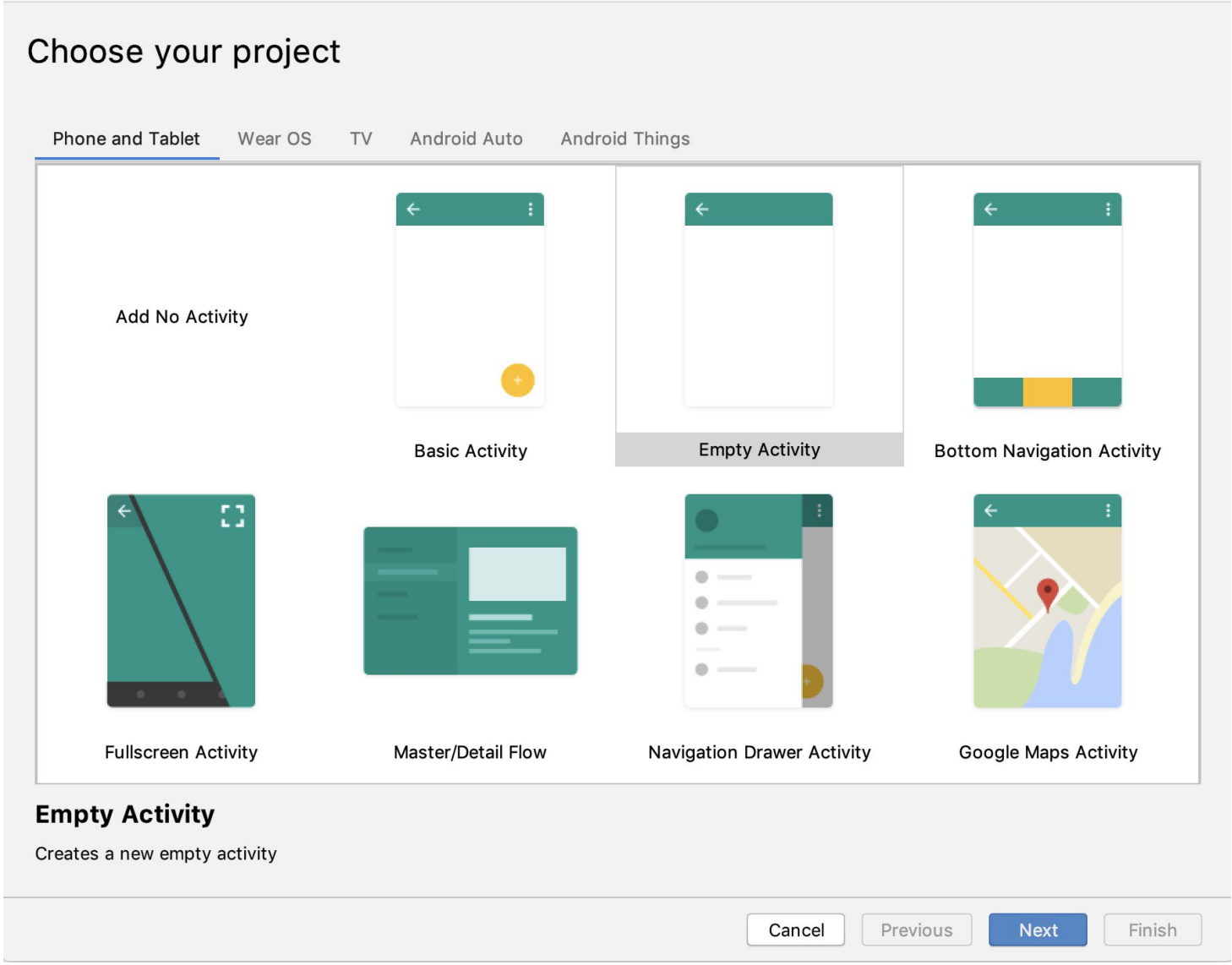
Criando nosso projeto

APP: Jogo da Véia - Criando o projeto



ame FASTFORWARD

APP: Jogo da Véia - Criando o projeto



ame FASTFORWARD

APP: Jogo da Véia - Criando o projeto

Configure your project

Name

Jogo da veia

Package name

br.com.heiderlopes.jogodaveia

Save location

/Users/heider.lopes/Downloads/Jogodaveia

Language

Kotlin

Minimum API level API 21: Android 5.0 (Lollipop)

Your app will run on approximately **85.0%** of devices.

Help me choose

- [ ] This project will support instant apps
- [x] Use androidx.* artifacts

Empty Activity

Creates a new empty activity

Cancel | Previous | Next | Finish

ame FASTFORWARD

APP: Jogo da Véia - Dependências

Adicionando as dependências necessárias, abra o arquivo **build.gradle (app)** e adicione as seguintes linhas abaixo:

```
dependencies {
  testImplementation "android.arch.core:core-testing:1.1.1"

  //Biblioteca AAC
  implementation "androidx.lifecycle:lifecycle-extensions:2.0.0"

  //Componentes da biblioteca de design
  implementation 'com.google.android.material:material:1.0.0'
  implementation 'androidx.gridlayout:gridlayout:1.0.0'

  androidTestImplementation("com.schibsted.spain:barista:2.9.0") {
     exclude group: 'com.android.support', module: 'support-annotations'
  }
}
```

APP: Jogo da Véia - Resources

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="colorPrimary">#009688</color>
  <color name="colorPrimaryDark">#00796B</color>
  <color name="colorAccent">#FFB74D</color>
  <color name="cell_color">#009688</color>
</resources>
```

APP: Jogo da Véia - Resources

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

```xml
<resources>
  <string name="app_name">Jogo da Velha</string>
  <string name="player1_hint">Player 1</string>
  <string name="player2_hint">Player 2</string>
  <string name="game_dialog_title">New game</string>
  <string name="done">Done</string>
  <string name="game_dialog_empty_name">Name mustn\'t be empty</string>
  <string name="game_dialog_same_names">Names mustn\'t be the same</string>
  <string name="game_end_dialog_headline">The winner is</string>
</resources>
```

APP: Jogo da Véia - Resources

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

```
<resources>

  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>

  <style name="CellTextView">
    <item name="android:gravity">center</item>
    <item name="android:textSize">48sp</item>
    <item name="android:textStyle">bold</item>
  </style>

</resources>
```

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_00.xml**

```xml
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:bottom="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:right="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_01.xml**

```xml
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:bottom="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:right="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:left="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_02.xml**

```xml
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:bottom="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:left="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_10.xml**

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:bottom="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:top="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:right="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_11.xml**

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">


  <item>
    <shape android:shape="rectangle">
      <padding android:bottom="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <padding android:top="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>
```

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_11.xml**

```
  <item>
    <shape android:shape="rectangle">
      <padding android:left="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <padding android:right="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_12.xml**

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:left="2dp" /> <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:bottom="2dp" /><solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:top="2dp" /> <solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_20.xml**

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:top="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <padding android:right="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Resources

Dentro da pasta **drawable** crie o arquivo **cell_21.xml**

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:top="2dp" /><solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:left="2dp" /><solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <padding android:right="2dp" /><solid android:color="@color/cell_color" />
    </shape>
  </item>
  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

APP: Jogo da Véia - Resources

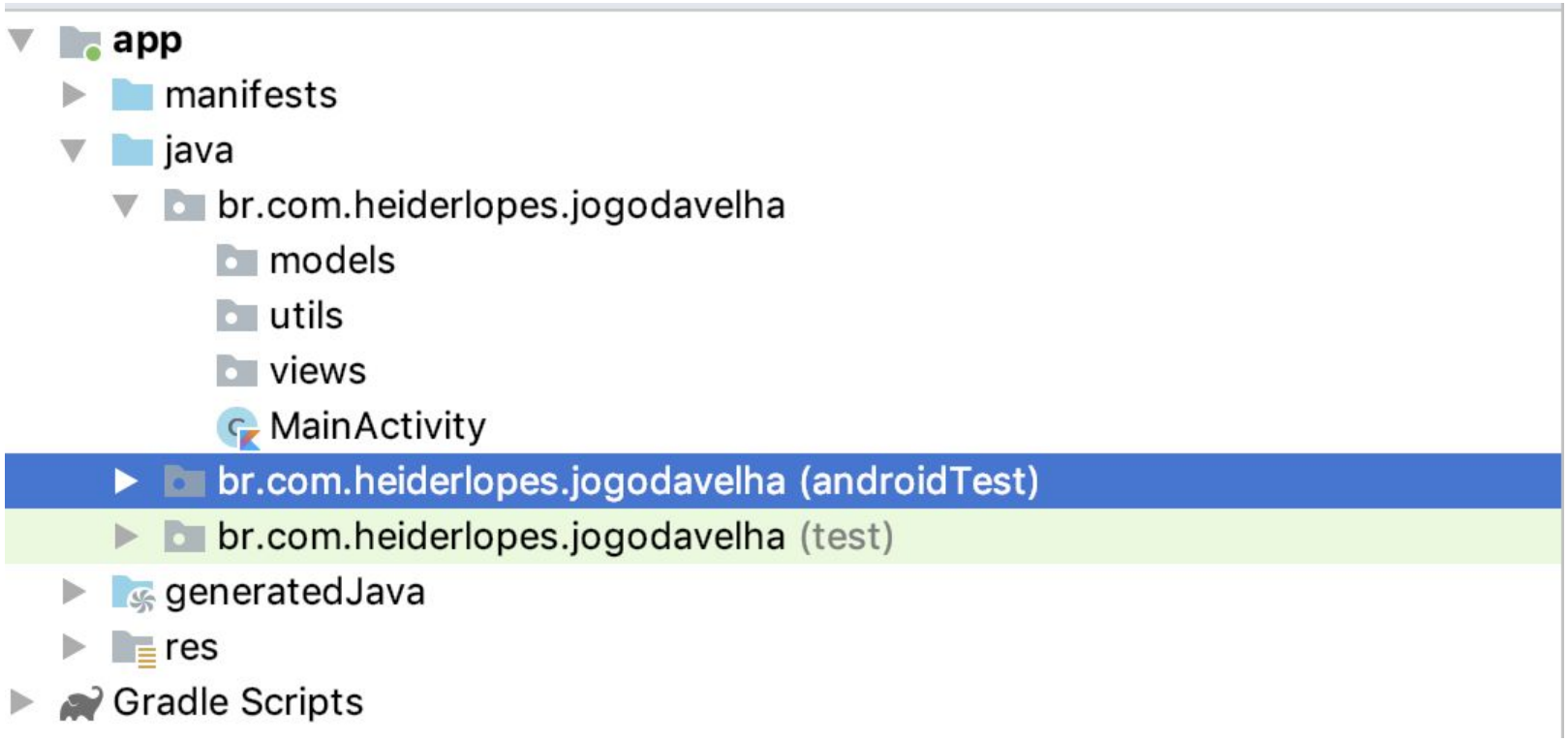
Dentro da pasta **drawable** crie o arquivo **cell_22.xml**

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape android:shape="rectangle">
      <padding android:top="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <padding android:left="2dp" />
      <solid android:color="@color/cell_color" />
    </shape>
  </item>

  <item>
    <shape android:shape="rectangle">
      <solid android:color="@android:color/white" />
    </shape>
  </item>
</layer-list>
```

ame FASTFORWARD

APP: Jogo da Véia - Estrutura do Projeto

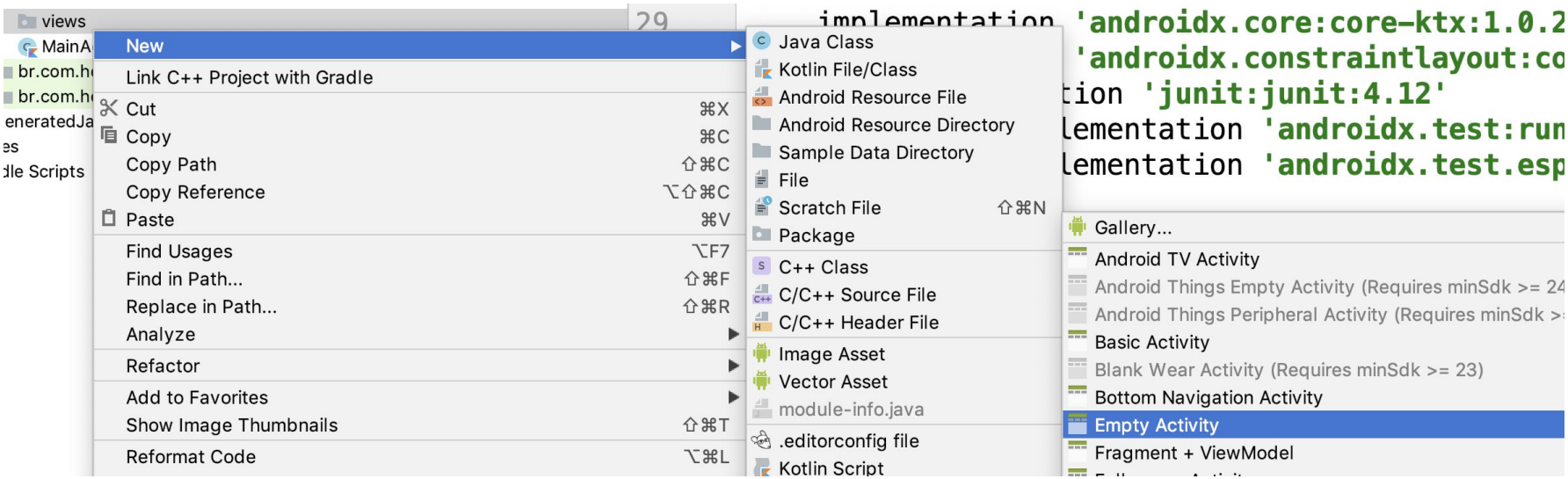
Crie a seguinte estrutura:



ame FASTFORWARD

APP: Jogo da Véia - Estrutura do Projeto

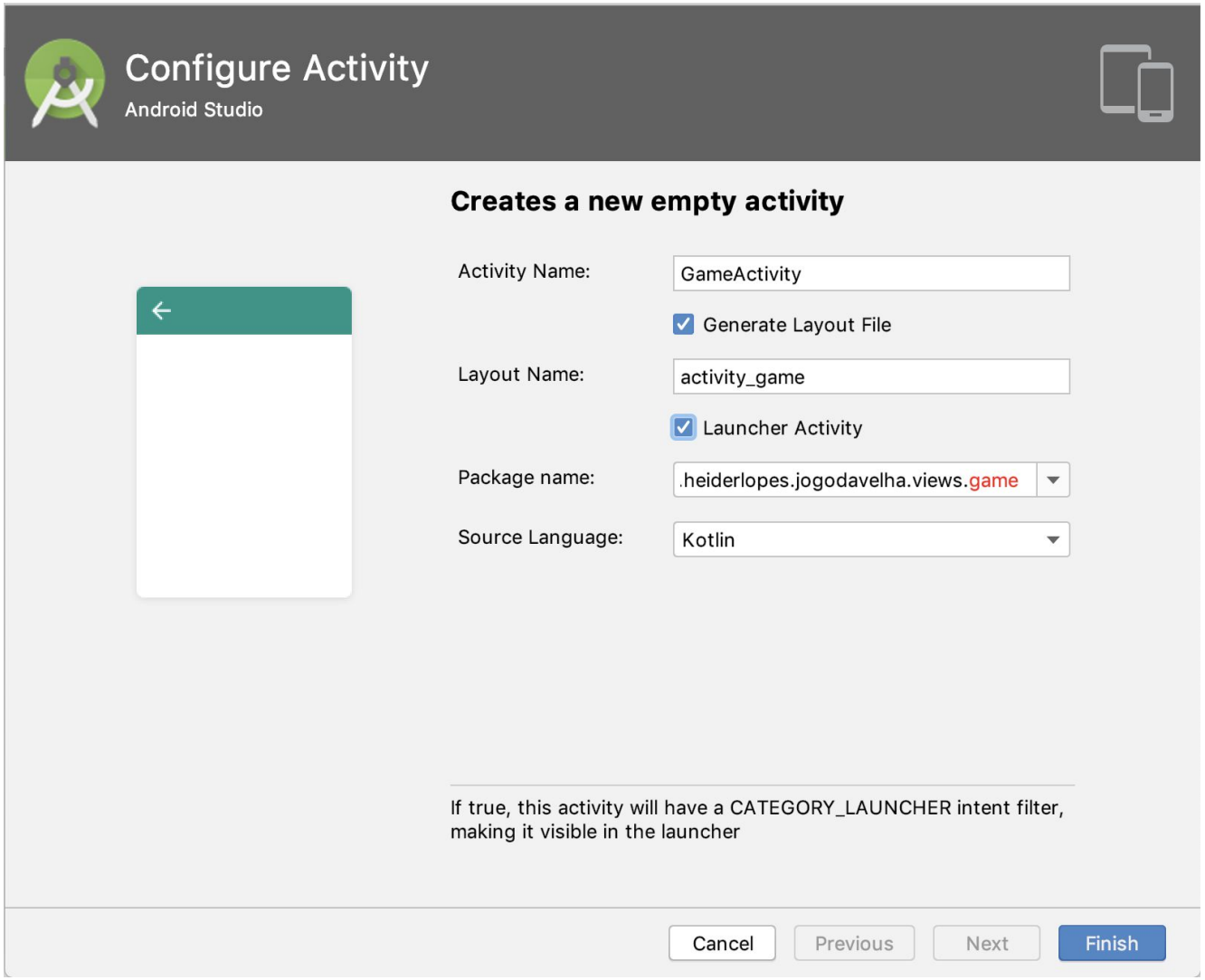
Crie uma nova **Empty Activity** chamada **GameActivity**



ame FASTFORWARD

APP: Jogo da Véia - Configurando o Jogo

Crie uma nova **Empty Activity** chamada **GameActivity**



ame FASTFORWARD

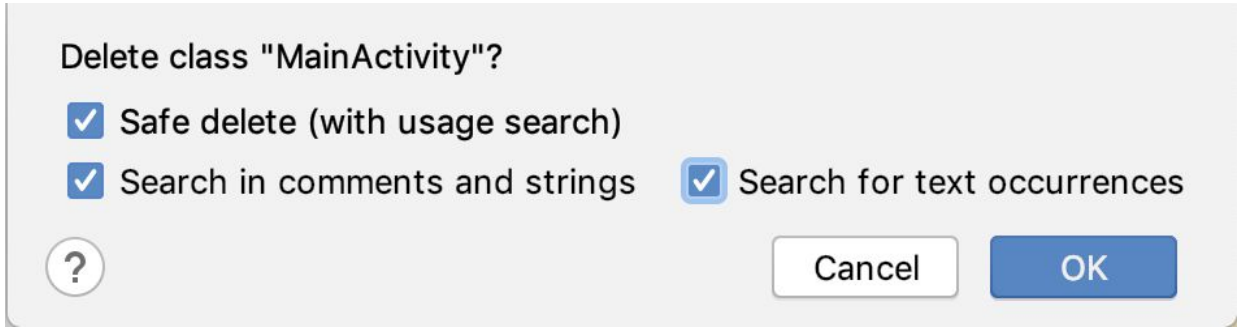
APP: Jogo da Véia - Configurando o Jogo

Crie as seguintes classes Kotlin dentro do package **views.game**

GameBeginDialog
GameEndDialog
GameViewModel

APP: Jogo da Véia - Configurando o Jogo

Remova a **MainActivity.kt** que foi criada automaticamente na criação do projeto.



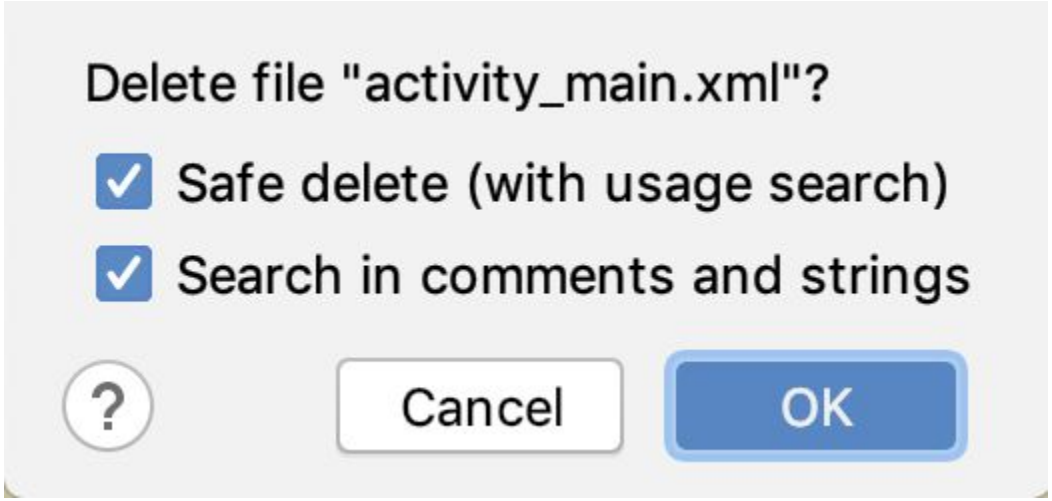
APP: Jogo da Véia - Configurando o Jogo

Abra o arquivo **AndroidManifest.xml** e remova a **MainActivity**

```
<activity android:name=".MainActivity">
  <intent-filter>
     <action android:name="android.intent.action.MAIN"/>

     <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
```

APP: Jogo da Véia - Configurando o Jogo

Remova o arquivo **activity_main.xml** do diretório **res** → **layout**



Criando os utilitários

APP: Jogo da Véia - Configurando o Jogo

Dentro do package **utils** crie um arquivo chamado **StringUtility.kt**

```
object StringUtility {

    fun stringFromNumbers(vararg numbers: Int): String {
        val sNumbers = StringBuilder()
        for (number in numbers)
            sNumbers.append(number)
        return sNumbers.toString()
    }

    @JvmStatic fun isNullOrEmpty(value: String?): Boolean {
        return value == null || value.isEmpty()
    }
}
```

Criando as models

APP: Jogo da Véia - Models

Dentro do package **models** adicione uma classe Kotlin chamada **Player** e adicione o seguinte código:

```
data class Player(var name: String, var value: String)
```

APP: Jogo da Véia - Models

Dentro do package **models** adicione uma classe Kotlin chamada **Cell** e adicione o seguinte código:

```kotlin
data class Cell(var player: Player) {

  val isEmpty: Boolean
      get() = player.value.isEmpty()
}
```

APP: Jogo da Véia - Models

Dentro do package **models** adicione uma classe **JAVA** chamada **Game** e adicione os seguintes códigos:

APP: Jogo da Véia - Models - Game

```
import android.util.Log;
import androidx.lifecycle.MutableLiveData;

public class Game {

  private static final String TAG = Game.class.getSimpleName();
  private static final int BOARD_SIZE = 3;

  public Player player1;
  public Player player2;

  public Player currentPlayer = player1;
  public Cell[][] cells;

  public MutableLiveData<Player> winner = new MutableLiveData<>();
```

APP: Jogo da Véia - Models - Game

```
public Game(String playerOne, String playerTwo) {
  cells = new Cell[BOARD_SIZE][BOARD_SIZE];
  player1 = new Player(playerOne, "x");
  player2 = new Player(playerTwo, "o");
  currentPlayer = player1;
}
```

APP: Jogo da Véia - Models - Game

```
public boolean hasGameEnded() {
  if (hasThreeSameHorizontalCells() || hasThreeSameVerticalCells() ||
hasThreeSameDiagonalCells()) {
      winner.setValue(currentPlayer);
      return true;
  }

  if (isBoardFull()) {
      winner.setValue(null);
      return true;
  }

  return false;
}
```

APP: Jogo da Véia - Models - Game

```
public boolean hasThreeSameHorizontalCells() {
  try {
    for (int i = 0; i < BOARD_SIZE; i++)
      if (areEqual(cells[i][0], cells[i][1], cells[i][2]))
        return true;

    return false;
  } catch (NullPointerException e) {
    Log.e(TAG, e.getMessage());
    return false;
  }
}
```

APP: Jogo da Véia - Models - Game

```
public boolean hasThreeSameVerticalCells() {
  try {
    for (int i = 0; i < BOARD_SIZE; i++)
      if (areEqual(cells[0][i], cells[1][i], cells[2][i]))
        return true;
    return false;
  } catch (NullPointerException e) {
    Log.e(TAG, e.getMessage());
    return false;
  }
}
```

APP: Jogo da Véia - Models - Game

```
public boolean hasThreeSameDiagonalCells() {
  try {
    return areEqual(cells[0][0], cells[1][1], cells[2][2]) ||
           areEqual(cells[0][2], cells[1][1], cells[2][0]);
  } catch (NullPointerException e) {
    Log.e(TAG, e.getMessage());
    return false;
  }
}
```

APP: Jogo da Véia - Models - Game

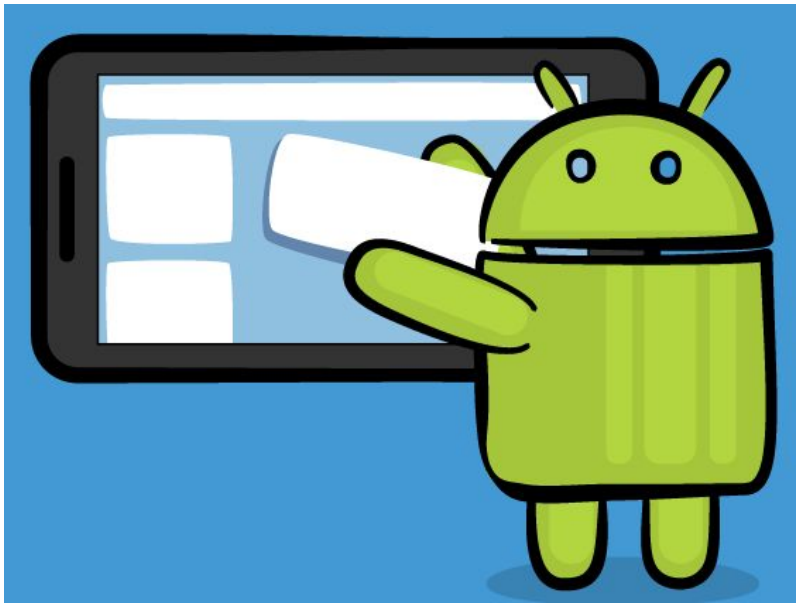
```
public boolean isBoardFull() {
  for (Cell[] row : cells)
    for (Cell cell : row)
      if (cell == null || cell.isEmpty())
        return false;
  return true;
}
```

APP: Jogo da Véia - Models - Game

```
private boolean areEqual(Cell... cells) {
  if (cells == null || cells.length == 0)
    return false;

  for (Cell cell : cells)
    if (cell == null || StringUtility.isNullOrEmpty(cell.getPlayer().getValue()))
      return false;

  Cell comparisonBase = cells[0];
  for (int i = 1; i < cells.length; i++)
    if (!comparisonBase.getPlayer().getValue().equals(cells[i].getPlayer().getValue()))
      return false;

  return true;
}
```

APP: Jogo da Véia - Models - Game

```
public void switchPlayer() {
    currentPlayer = currentPlayer == player1 ? player2 : player1;
  }

  public void reset() {
    player1 = null;
    player2 = null;
    currentPlayer = null;
    cells = null;
  }
}
```



Criando os layouts dos dialogs

APP: Jogo da Véia - Layouts

Dentro da pasta **layout** crie um arquivo chamado **game_begin_dialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp">
  <com.google.android.material.textfield.TextInputLayout
      android:id="@+id/player1Layout"
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/et_player1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/player1_hint"
        android:inputType="textPersonName"
        android:maxLines="1"
        android:singleLine="true" />
  </com.google.android.material.textfield.TextInputLayout>
```

ame FASTFORWARD

APP: Jogo da Véia - Layouts

Dentro da pasta **layout** crie um arquivo chamado **game_begin_dialog.xml**

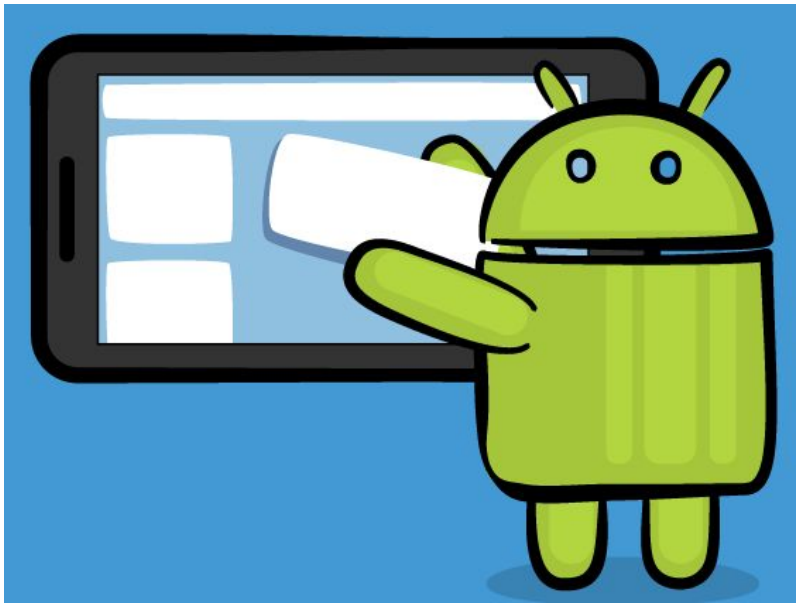
```
<com.google.android.material.textfield.TextInputLayout
        android:id="@+id/player2Layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

    <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/et_player2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/player2_hint"
            android:inputType="textPersonName"
            android:maxLines="1"
            android:singleLine="true" />

</com.google.android.material.textfield.TextInputLayout>

</LinearLayout>
```

APP: Jogo da Véia - Layouts

Dentro da pasta **layout** crie um arquivo chamado **game_end_dialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp">
  <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:padding="8dp"
      android:text="@string/game_end_dialog_headline" />
  <TextView
      android:id="@+id/tvWinner"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:padding="8dp"
      android:textStyle="bold"
      tools:text="Heider Lopes" />
</LinearLayout>
```

ame FASTFORWARD



Programando nossos Dialogs

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
class GameBeginDialog : DialogFragment() {

  private lateinit var player1Layout: TextInputLayout
  private lateinit var player2Layout: TextInputLayout

  private lateinit var player1EditText: TextInputEditText
  private lateinit var player2EditText: TextInputEditText

  private var player1: String = ""
  private var player2: String = ""

  private lateinit var rootView: View
  private lateinit var activity: GameActivity
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
  initViews()
  val alertDialog = AlertDialog.Builder(context)
      .setView(rootView)
      .setTitle(R.string.game_dialog_title)
      .setCancelable(false)
      .setPositiveButton(R.string.done, null)
      .create()
  alertDialog.setCanceledOnTouchOutside(false)
  alertDialog.setCancelable(false)
  alertDialog.setOnShowListener { onDialogShow(alertDialog) }
  return alertDialog
}
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
private fun initViews() {
  rootView = LayoutInflater.from(context)
      .inflate(R.layout.game_begin_dialog, null, false)

  player1Layout = rootView.findViewById(R.id.player1Layout)
  player2Layout = rootView.findViewById(R.id.player2Layout)

  player1EditText = rootView.findViewById(R.id.et_player1)
  player2EditText = rootView.findViewById(R.id.et_player2)
  addTextWatchers()
}
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
private fun onDialogShow(dialog: AlertDialog) {
  val positiveButton = dialog.getButton(AlertDialog.BUTTON_POSITIVE)
  positiveButton.setOnClickListener { onDoneClicked() }
}

private fun onDoneClicked() {
  if (isAValidName(player1Layout, player1) and
      isAValidName(player2Layout, player2)
  ) {
      activity.onPlayersSet(player1, player2)
      dismiss()
  }
}
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
private fun isAValidName(layout: TextInputLayout?, name: String?): Boolean {
  if (TextUtils.isEmpty(name)) {
      layout?.isErrorEnabled = true
      layout?.error = getString(R.string.game_dialog_empty_name)
      return false
  }

  if (player1.equals(player2, ignoreCase = true)) {
      layout?.isErrorEnabled = true
      layout?.error = getString(R.string.game_dialog_same_names)
      return false
  }

  layout?.isErrorEnabled = false
  layout?.error = ""
  return true
}
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
private fun addTextWatchers() {
  player1EditText.addTextChangedListener(object : TextWatcher {
     override fun beforeTextChanged(s: CharSequence, start: Int, count: Int, after: Int) {}
     override fun onTextChanged(s: CharSequence, start: Int, before: Int, count: Int) {}
     override fun afterTextChanged(s: Editable) {
        player1 = s.toString()
     }
  })
  player2EditText.addTextChangedListener(object : TextWatcher {
     override fun beforeTextChanged(s: CharSequence, start: Int, count: Int, after: Int) {}
     override fun onTextChanged(s: CharSequence, start: Int, before: Int, count: Int) {}
     override fun afterTextChanged(s: Editable) {
        player2 = s.toString()
     }
  })
}
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameBeginDialog** e adicione o seguinte código:

```
companion object {

    fun newInstance(activity: GameActivity): GameBeginDialog {
        val dialog = GameBeginDialog()
        dialog.activity = activity
        return dialog
    }
  }
}
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameEndDialog** e adicione o seguinte código:

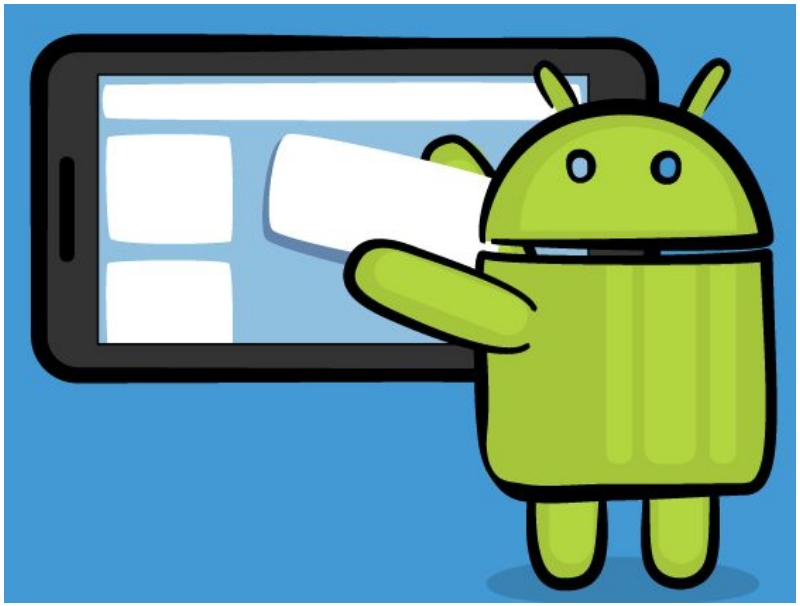
```
class GameEndDialog : DialogFragment() {

  private var rootView: View? = null
  private var activity: GameActivity? = null
  private var winnerName: String? = null

  override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
     initViews()
     val alertDialog = AlertDialog.Builder(context)
        .setView(rootView)
        .setCancelable(false)
        .setPositiveButton(R.string.done) { _, _ -> onNewGame() }
        .create()
     alertDialog.setCanceledOnTouchOutside(false)
     alertDialog.setCancelable(false)
     return alertDialog
  }
```

APP: Jogo da Véia - Código Fonte

Abra o arquivo **GameEndDialog** e adicione o seguinte código:

```
private fun initViews() {
    rootView = LayoutInflater.from(context)
        .inflate(R.layout.game_end_dialog, null, false)
    (rootView!!.findViewById(R.id.tvWinner) as TextView).text = winnerName
  }

  private fun onNewGame() {
    dismiss()
    activity?.promptForPlayers()
  }

  companion object {
    fun newInstance(activity: GameActivity, winnerName: String): GameEndDialog {
        val dialog = GameEndDialog()
        dialog.activity = activity
        dialog.winnerName = winnerName
        return dialog
    }
  }
}
```

ame FASTFORWARD



Programando nosso Game

APP: Jogo da Véia - DataBinding

Abra o arquivo **GameActivity.kt** e adicione os seguintes métodos:

```
fun onPlayersSet(player1: String, player2: String) {

}

fun promptForPlayers() {

}
```

APP: Jogo da Véia - DataBinding

Abra o arquivo **build.gradle (app)** e adicione o databinding

```
android {

  dataBinding {
     enabled = true
  }

}
```

APP: Jogo da Véia - ViewModel

Abra o arquivo **GameViewModel** e adicione o seguinte código:

```
class GameViewModel : ViewModel() {
  lateinit var cells: ObservableArrayMap<String, String>
  private lateinit var game: Game
  val winner: LiveData<Player>
     get() = game.winner

  fun init(player1: String, player2: String) {
     game = Game(player1, player2)
     cells = ObservableArrayMap()
  }
  fun onClickedCellAt(row: Int, column: Int) {
     if (game.cells[row][column] == null) {
        game.cells[row][column] = Cell(game.currentPlayer)
        cells[stringFromNumbers(row, column)] = game.currentPlayer.value
        if (game.hasGameEnded())
           game.reset()
        else
           game.switchPlayer()
     }
  }
}
```

ame FASTFORWARD

APP: Jogo da Véia - Layout

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

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

  <data>

    <variable
        name="gameViewModel"
        type="br.com.heiderlopes.jogodaveia.views.game.GameViewModel" />
  </data>

  <androidx.gridlayout.widget.GridLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white"
    android:padding="32dp">
```

APP: Jogo da Véia - Layout

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

```
<TextView
    android:id="@+id/cell_00"
    style="@style/CellTextView"
    android:background="@drawable/cell_00"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(0, 0)}"
    android:text='@{gameViewModel.cells["00"]}'
    app:layout_column="1"
    app:layout_columnWeight="1"
    app:layout_row="1"
    app:layout_rowWeight="1" />
<TextView
    android:id="@+id/cell_01"
    style="@style/CellTextView"
    android:background="@drawable/cell_01"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(0, 1)}"
    android:text='@{gameViewModel.cells["01"]}'
    app:layout_column="2"
    app:layout_columnWeight="1"
    app:layout_row="1"
    app:layout_rowWeight="1" />
```

APP: Jogo da Véia - Layout

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

```
<TextView
    android:id="@+id/cell_02"
    style="@style/CellTextView"
    android:background="@drawable/cell_02"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(0, 2)}"
    android:text='@{gameViewModel.cells["02"]}'
    app:layout_column="3"
    app:layout_columnWeight="1"
    app:layout_row="1"
    app:layout_rowWeight="1" />
<TextView
    android:id="@+id/cell_10"
    style="@style/CellTextView"
    android:background="@drawable/cell_10"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(1, 0)}"
    android:text='@{gameViewModel.cells["10"]}'
    app:layout_column="1"
    app:layout_columnWeight="1"
    app:layout_row="2"
    app:layout_rowWeight="1" />
```

APP: Jogo da Véia - Layout

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

```
<TextView
    android:id="@+id/cell_11"
    style="@style/CellTextView"
    android:background="@drawable/cell_11"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(1, 1)}"
    android:text='@{gameViewModel.cells["11"]}'
    app:layout_column="2"
    app:layout_columnWeight="1"
    app:layout_row="2"
    app:layout_rowWeight="1" />
<TextView
    android:id="@+id/cell_12"
    style="@style/CellTextView"
    android:background="@drawable/cell_12"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(1, 2)}"
    android:text='@{gameViewModel.cells["12"]}'
    app:layout_column="3"
    app:layout_columnWeight="1"
    app:layout_row="2"
    app:layout_rowWeight="1" />
```

APP: Jogo da Véia - Layout

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

```
<TextView
    android:id="@+id/cell_20"
    style="@style/CellTextView"
    android:background="@drawable/cell_20"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(2, 0)}"
    android:text='@{gameViewModel.cells["20"]}'
    app:layout_column="1"
    app:layout_columnWeight="1"
    app:layout_row="3"
    app:layout_rowWeight="1" />
<TextView
    android:id="@+id/cell_21"
    style="@style/CellTextView"
    android:background="@drawable/cell_21"
    android:onClick="@{() -> gameViewModel.onClickedCellAt(2, 1)}"
    android:text='@{gameViewModel.cells["21"]}'
    app:layout_column="2"
    app:layout_columnWeight="1"
    app:layout_row="3"
    app:layout_rowWeight="1" />
```

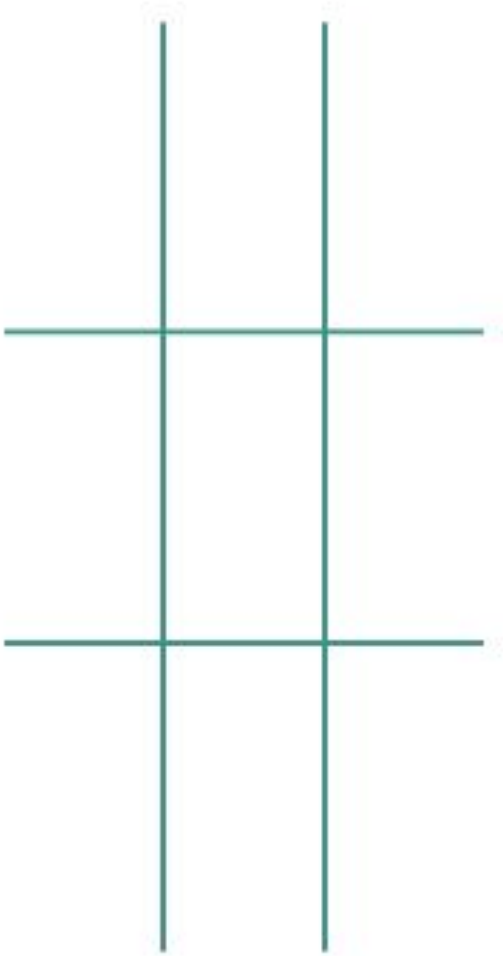
APP: Jogo da Véia - Layout

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

```
<TextView
        android:id="@+id/cell_22"
        style="@style/CellTextView"
        android:background="@drawable/cell_22"
        android:onClick="@{() -> gameViewModel.onClickedCellAt(2, 2)}"
        android:text='@{gameViewModel.cells["22"]}'
        app:layout_column="3"
        app:layout_columnWeight="1"
        app:layout_row="3"
        app:layout_rowWeight="1" />

  </androidx.gridlayout.widget.GridLayout>

</layout>
```

APP: Jogo da Véia - Layout

Teremos o seguinte resultado:



ame FASTFORWARD

APP: Jogo da Véia - Layout

Programando nossa **GameActivity.kt**

```
class GameActivity : AppCompatActivity() {

  lateinit var gameViewModel: GameViewModel

  override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     promptForPlayers()
  }

  fun promptForPlayers() {
     val dialog = GameBeginDialog.newInstance(this)
     dialog.isCancelable = false
     dialog.show(supportFragmentManager, GAME_BEGIN_DIALOG_TAG)
  }

  fun onPlayersSet(player1: String, player2: String) {
     initDataBinding(player1, player2)
  }
```

APP: Jogo da Véia - Layout

Programando nossa **GameActivity.kt**

```
private fun initDataBinding(player1: String, player2: String) {
  val activityGameBinding =
DataBindingUtil.setContentView<ActivityGameBinding>(this, R.layout.activity_game)
  gameViewModel = ViewModelProviders.of(this).get(GameViewModel::class.java)
  gameViewModel.init(player1, player2)
  activityGameBinding.gameViewModel = gameViewModel
  setUpOnGameEndListener()
}

private fun setUpOnGameEndListener() {
  gameViewModel.winner.observe(this, Observer { this.onGameWinnerChanged(it) })
}
```

APP: Jogo da Véia - Layout

Programando nossa **GameActivity.kt**

```
@VisibleForTesting
  fun onGameWinnerChanged(winner: Player?) {
      val winnerName = if (winner == null || winner.name.isEmpty()) NO_WINNER else
winner.name
      val dialog = GameEndDialog.newInstance(this, winnerName)
      dialog.isCancelable = false
      dialog.show(supportFragmentManager, GAME_END_DIALOG_TAG)
  }

  companion object {

      private val GAME_BEGIN_DIALOG_TAG = "game_dialog_tag"
      private val GAME_END_DIALOG_TAG = "game_end_dialog_tag"
      private val NO_WINNER = "No one"
  }
}
```

Criando nossos testes automatizados

APP: Jogo da Véia - Testes Automatizados

Atualmente temos vários serviços e plataformas de **Integração Contínua (Continuous Integration)** gerando builds a todo momento como: **Jenkins, Travis CI, Circle CI, BitBucket Pipelines**, testar é algo essencial e obrigatório para que consigamos montar uma cultura de desenvolvimento ágil e de **Entrega Contínua (Continuous Delivery)** de produtos de qualidade para nossos clientes.

E no ambiente cheio de mudanças é necessário garantir que determinada funcionalidade se comporte como esperado. Além disso, com uma cobertura boa de testes, conseguimos garantir que atualizações no sistema não quebrem funcionalidades já existentes.

Com testes automatizados, basicamente temos um roteiro documentado e pronto de como aquela funcionalidade deve se comportar.

APP: Jogo da Véia - Testes Automatizados - O que testar?

Depende das características do projeto, como escopo, arquitetura e tipos de testes (unitário, de estresse, funcional, etc). Mas como regra geral, devemos testar:

Se determinada feature se comporta da maneira esperada
Cenários possíveis que o sistema pode assumir
Se determinada regra de negócio está correta
Se determinada dependência satisfaz a classe

Testes podem denunciar que sua arquitetura precisa ser refatorada. Por exemplo, se o seu projeto está muito difícil de testar de forma isolada, provavelmente a sua arquitetura está muito acoplada, quebrando os princípios SOLID. Existem várias abordagens de testes como o TDD (Test-Driven Development), BDD (Behavior-Driven Development), ATDD (Acceptance Test-Driven Development) que auxiliam para a construção e/ou evolução arquiteturas resilientes.

APP: Jogo da Véia - Testes Automatizados - Estrutura de Testes

No Android Studio temos a seguinte estrutura:

src/androidTest: Esta pasta contém todos os testes de UI, conhecidos também como Testes de Instrumentação (Instrumentation Tests). Esses tipos de testes comumente simulam a interação do usuário com as várias partes do sistema, como clique em botão, arraste, rolagem na tela (scroll); ou asserção de elementos na estrutura do layout, como checagem de visibilidade, se o componente pertence à hierarquia, se está corretamente alinhado e muito mais; ou ainda se determinada ação (Intent) deve ser executada.

src/test: contém os testes unitários para classes de modelo, de comunicação com API REST, persistência, etc. Muitas vezes, para simular o comportamento de dependências de uma classe utilizamos o conceito de mock objects.

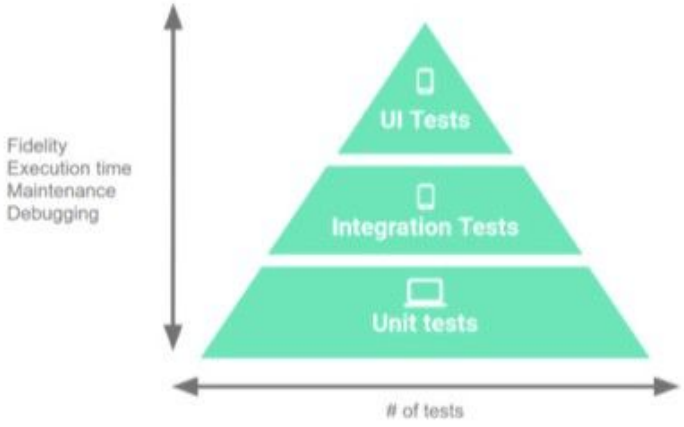
A pirâmide de testes

APP: Jogo da Véia - Testes Automatizados - Estrutura de Testes

A pirâmide de testes poderia ser resumida como uma estratégia de divisão de sua suíte de testes.

Nessa divisão, as camadas mais inferiores contêm testes mais rápidos, baratos e isolados ao comportamento da aplicação.

Na parte superior da pirâmide teremos testes mais lentos, caros e integrados.



APP: Jogo da Véia - Testes Automatizados - Estrutura de Testes

Dentro do Android Studio nós vamos encontrar duas estruturas de pacotes de testes:

test (não instrumentados): pode ser executado sem necessidade de carregar os recursos do Android, ou seja, sem necessidade de carregar um emulador ou um dispositivo real. Esses testes são executados na JVM (Java Virtual Machine) e normalmente estão relacionados às camadas mais baixas da pirâmide.

androidTest (instrumentados): necessariamente vai precisar de um emulador ou um dispositivo real e aqui normalmente colocamos os testes de integração ou end-to-end. Esses testes se encaixam nas camadas mais altas da pirâmide.

APP: Jogo da Véia - Testes Automatizados - Estrutura de Testes

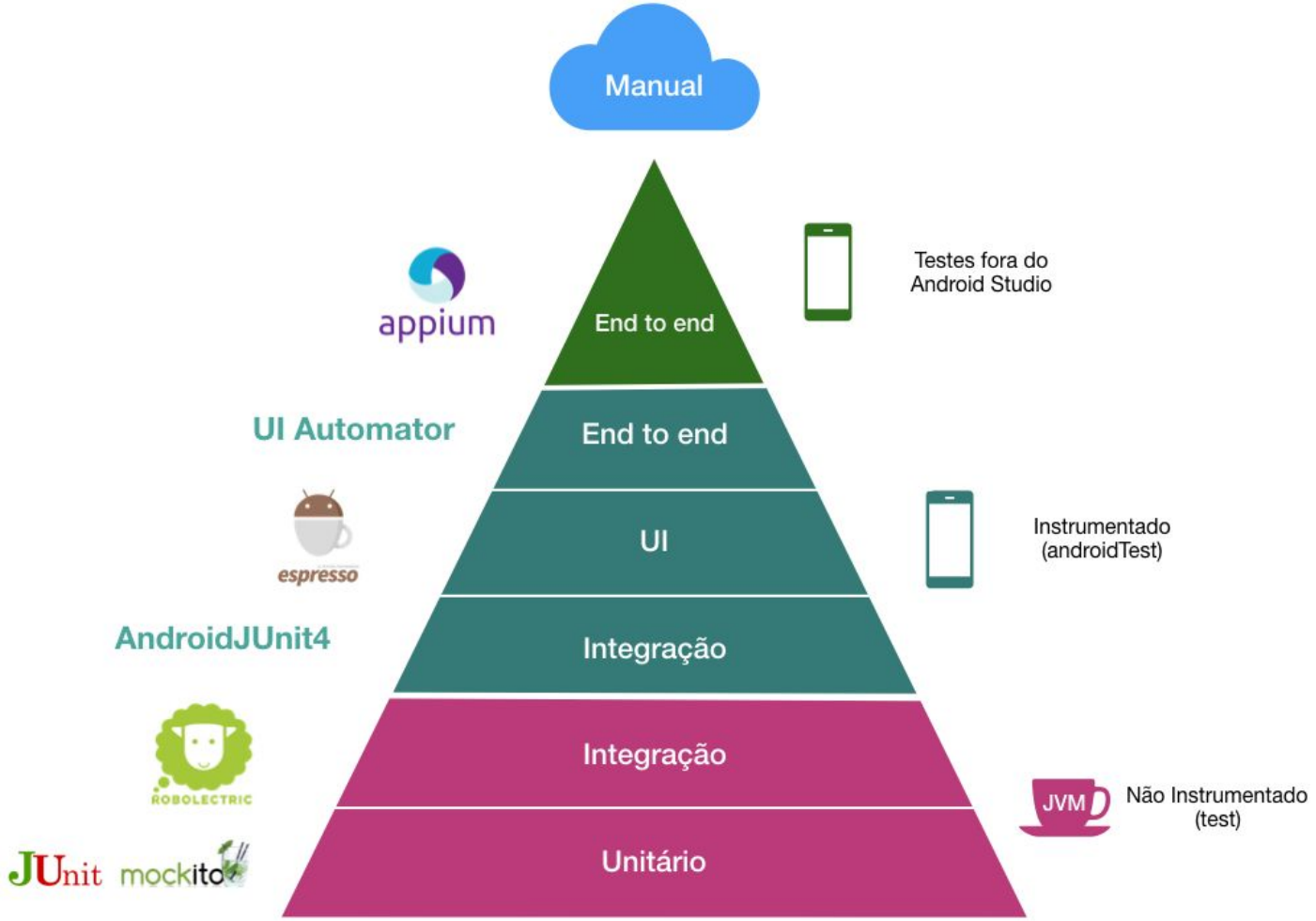
O Google também possui sua própria definição de pirâmide para Android, que está dividida nas camadas **small**, **medium** e **large**. De acordo com eles, a divisão ideal de testes para Android seria:

70% testes small ou unitários
20% testes medium ou integrados
10% testes large ou end-to-end

Essas definições são apenas sugestões do Google, a distribuição da sua pirâmide de testes vai depender da necessidade do seu projeto!

APP: Jogo da Véia - Testes Automatizados - Quais ferramentas?

Podemos destacar as seguintes ferramentas para construção de testes:



https://medium.com/android-dev-br/explorando-a-pir%C3%A2mide-de-testes-no-android-parte-1-18ea135808df

APP: Jogo da Véia - Testes Automatizados - Quais ferramentas?

Essa pirâmide está dividida em 4 blocos:

Nível mais baixo: Camada não instrumentada no ambiente de desenvolvimento Android. Inclui testes unitários com JUnit e Mockito e testes unitários e integrados com Robolectric.

Nível intermediário: Camada instrumentada no ambiente de desenvolvimento Android. Inclui testes de UI e integração com Espresso e AndroidJUnit4, e também testes end-to-end com UI Automator.

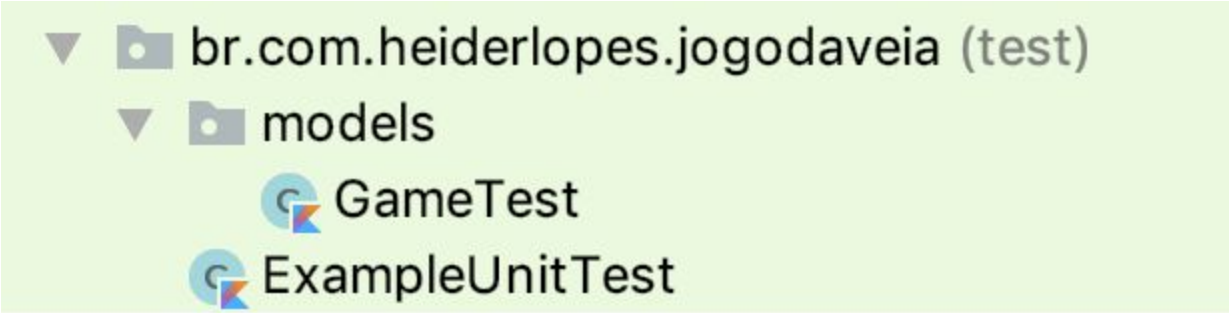
Nível mais alto: Camada end-to-end com ferramentas de terceiros fora do ambiente de desenvolvimento Android. Nessa camada podemos escrever testes com tecnologias e linguagens diferentes.

Nível fora da pirâmide: Testes manuais

Criando nossos testes unitários

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Crie a seguinte package e classe conforme imagem abaixo:



APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Configurando os objetos antes de executar o teste

```
class GameTest {

  private lateinit var game: Game
  private lateinit var player: Player
  private lateinit var anotherPlayer: Player

  @Before
  @Throws(Exception::class)
  fun setUp() {
     game = Game("Heider", "William")
     player = game.player1
     anotherPlayer = game.player2
  }
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células diagonais da esquerda

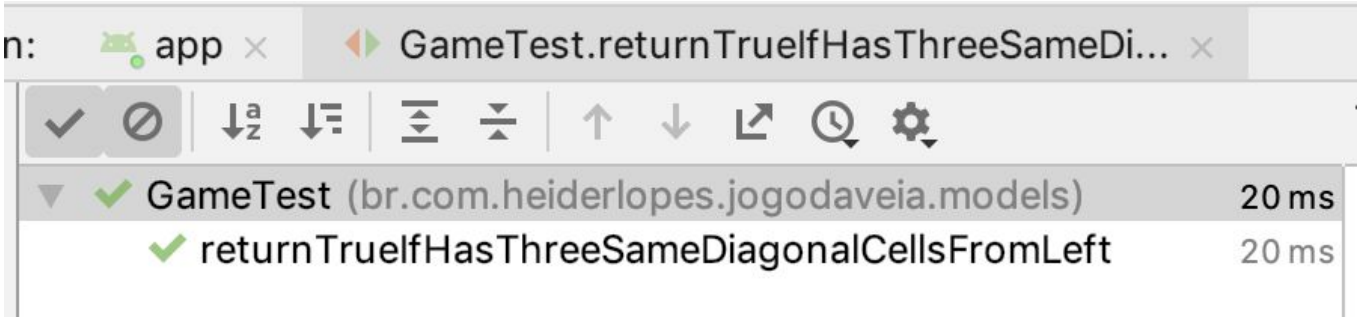
```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameDiagonalCellsFromLeft() {
    val cell = Cell(player)
    game.cells[0][0] = cell
    game.cells[1][1] = cell
    game.cells[2][2] = cell
    val hasThreeSameDiagonalCells = game.hasThreeSameDiagonalCells()
    Assert.assertTrue(hasThreeSameDiagonalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Rodando o teste criado:

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameDiagonalCellsFromLeft() {
    val cell = Cell(player)
    game.cells[0][0] = cell
    game.cells[1][1] = cell
    game.cells[2][2] = cell
    val hasThreeSameDiagonalCells = game.hasThreeSameDiagonalCells()
    Assert.assertTrue(hasThreeSameDiagonalCells)
}
```

Resultado do teste:



ame FASTFORWARD

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células horizontais na linha 1

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameHorizontalCellsAtRow1() {
  val cell = Cell(player)
  game.cells[0][0] = cell
  game.cells[0][1] = cell
  game.cells[0][2] = cell
  val hasThreeSameHorizontalCells = game.hasThreeSameHorizontalCells()
  assertTrue(hasThreeSameHorizontalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células horizontais na linha 2

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameHorizontalCellsAtRow2() {
  val cell = Cell(player)
  game.cells[1][0] = cell
  game.cells[1][1] = cell
  game.cells[1][2] = cell
  val hasThreeSameHorizontalCells = game.hasThreeSameHorizontalCells()
  assertTrue(hasThreeSameHorizontalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células horizontais na linha 3

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameHorizontalCellsAtRow3() {
  val cell = Cell(player)
  game.cells[2][0] = cell
  game.cells[2][1] = cell
  game.cells[2][2] = cell
  val hasThreeSameHorizontalCells = game.hasThreeSameHorizontalCells()
  assertTrue(hasThreeSameHorizontalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar falso se não tiver três mesmas células horizontais

```
@Test
@Throws(Exception::class)
fun returnFalseIfDoesNotHaveThreeSameHorizontalCells() {
  val cell = Cell(player)
  val anotherCell = Cell(anotherPlayer)
  game.cells[0][0] = cell
  game.cells[0][1] = cell
  game.cells[0][2] = anotherCell
  val hasThreeSameHorizontalCells = game.hasThreeSameHorizontalCells()
  assertFalse(hasThreeSameHorizontalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células verticais na coluna 1

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameVerticalCellsAtColumn1() {
  val cell = Cell(player)
  game.cells[0][0] = cell
  game.cells[1][0] = cell
  game.cells[2][0] = cell
  val hasThreeSameVerticalCells = game.hasThreeSameVerticalCells()
  assertTrue(hasThreeSameVerticalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células verticais na coluna 2

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameVerticalCellsAtColumn2() {
  val cell = Cell(player)
  game.cells[0][1] = cell
  game.cells[1][1] = cell
  game.cells[2][1] = cell
  val hasThreeSameVerticalCells = game.hasThreeSameVerticalCells()
  assertTrue(hasThreeSameVerticalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células verticais na coluna 3

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameVerticalCellsAtColumn3() {
  val cell = Cell(player)
  game.cells[0][2] = cell
  game.cells[1][2] = cell
  game.cells[2][2] = cell
  val hasThreeSameVerticalCells = game.hasThreeSameVerticalCells()
  assertTrue(hasThreeSameVerticalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células diagonais da esquerda

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameDiagonalCellsFromLeft() {
    val cell = Cell(player)
    game.cells[0][0] = cell
    game.cells[1][1] = cell
    game.cells[2][2] = cell
    val hasThreeSameDiagonalCells = game.hasThreeSameDiagonalCells()
    Assert.assertTrue(hasThreeSameDiagonalCells)
  }
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar verdadeiro se tiver três mesmas células diagonais da direita

```
@Test
@Throws(Exception::class)
fun returnTrueIfHasThreeSameDiagonalCellsFromRight() {
  val cell = Cell(player)
  game.cells[0][2] = cell
  game.cells[1][1] = cell
  game.cells[2][0] = cell
  val hasThreeSameDiagonalCells = game.hasThreeSameDiagonalCells()
  assertTrue(hasThreeSameDiagonalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste que irá retornar falso se não tiver três mesmas células diagonais

```
@Test
@Throws(Exception::class)
fun returnFalseIfDoesNotHaveThreeSameDiagonalCells() {
  val cell = Cell(player)
  val anotherCell = Cell(anotherPlayer)
  game.cells[0][2] = cell
  game.cells[1][1] = cell
  game.cells[2][0] = anotherCell
  val hasThreeSameDiagonalCells = game.hasThreeSameDiagonalCells()
  assertFalse(hasThreeSameDiagonalCells)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste para verificar se ocorre o fim do jogo se tiver três mesmas células horizontais

```
@Test
fun endGameIfHasThreeSameHorizontalCells() {
  val cell = Cell(player)
  game.cells[0][0] = cell
  game.cells[0][1] = cell
  game.cells[0][2] = cell
  val hasGameEnded = game.hasGameEnded()
  assertTrue(hasGameEnded)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste para verificar se ocorre o fim do jogo se tiver três mesmas células verticais

```
@Test
fun endGameIfHasThreeSameVerticalCells() {
  val cell = Cell(player)
  game.cells[0][0] = cell
  game.cells[1][0] = cell
  game.cells[2][0] = cell
  val hasGameEnded = game.hasGameEnded()
  assertTrue(hasGameEnded)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste para verificar se ocorre o fim do jogo se tiver três mesmas células diagonais.

```
@Test
fun endGameIfHasThreeSameDiagonalCells() {
  val cell = Cell(player)
  game.cells[0][0] = cell
  game.cells[1][1] = cell
  game.cells[2][2] = cell
  val hasGameEnded = game.hasGameEnded()
  assertTrue(hasGameEnded)
}
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste para verificar se ocorre o final do jogo se o tabuleiro estiver cheio.

```
@Test
fun endGameIfBoardIsFull() {
  val cell1 = Cell(Player("1", "x"))
  val cell2 = Cell(Player("2", "o"))
  val cell3 = Cell(Player("1", "x"))
  val cell4 = Cell(Player("2", "o"))
  val cell5 = Cell(Player("1", "x"))
  val cell6 = Cell(Player("2", "o"))
  val cell7 = Cell(Player("1", "x"))
  val cell8 = Cell(Player("2", "o"))
  val cell9 = Cell(Player("1", "x"))
```

APP: Jogo da Véia - Testes Automatizados - Testando a classe Game

Iremos criar um teste para verificar se ocorre o final do jogo se o tabuleiro estiver cheio.

```
  game.cells[0][0] = cell1
  game.cells[0][1] = cell2
  game.cells[0][2] = cell3
  game.cells[1][0] = cell4
  game.cells[1][1] = cell5
  game.cells[1][2] = cell6
  game.cells[2][0] = cell7
  game.cells[2][1] = cell8
  game.cells[2][2] = cell9
  val isBoardFull = game.isBoardFull
  assertTrue(isBoardFull)
}
```

Criando nossos testes de interface

APP: Jogo da Véia - Testes Automatizados - Testando a interface

O Barista torna o desenvolvimento do teste da interface do usuário mais rápido, fácil e previsível.

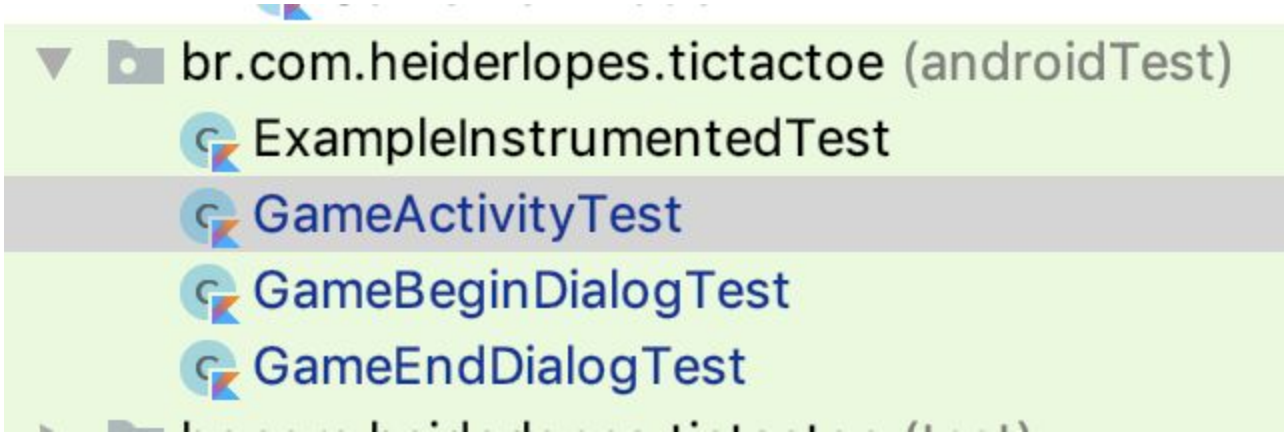
Construído sobre o Espresso, ele fornece uma API simples e detectável, removendo a maior parte do clichê e verbosidade das tarefas comuns do Espresso.

Documentação disponível em:

https://github.com/SchibstedSpain/Barista

APP: Jogo da Véia - Testes Automatizados - Testando a interface

Crie as seguintes classes Kotlin para testarmos a nossa interface



APP: Jogo da Véia - Testes Automatizados - GameActivityTest

```
class GameActivityTest {

  @Rule @JvmField
  var activityRule: ActivityTestRule<GameActivity> = ActivityTestRule(GameActivity::class.java)
  private val player1 = Player("Heider", "x")
  private val player2 = Player("William", "o")
  @Test
  fun end_game_when_one_player_wins() {
    writeTo(R.id.et_player1, player1.name)
    writeTo(R.id.et_player2, player2.name)
    clickDialogPositiveButton()

    clickOn(R.id.cell_00)
    clickOn(R.id.cell_10)
    clickOn(R.id.cell_01)
    clickOn(R.id.cell_11)
    clickOn(R.id.cell_02)

    assertDisplayed(R.id.tv_winner)
    assertDisplayed(player1.name)
  }
}
```

ame FASTFORWARD

APP: Jogo da Véia - Testes Automatizados - GameBeginDialogTest

Neste teste verificamos se exibe a mesma mensagem de nomes iguais se os nomes forem iguais.

```
@Test
@Throws(Exception::class)
fun display_same_names_message_if_names_same() {
  writeTo(R.id.et_player1, "Heider")
  writeTo(R.id.et_player2, "Heider")

  clickDialogPositiveButton()

  assertDisplayed(R.string.game_dialog_same_names)
}
```

APP: Jogo da Véia - Testes Automatizados - GameBeginDialogTest

Neste teste verificamos se exibe a mensagem de nome vazio se um nome estiver vazio.

```
@Test
@Throws(Exception::class)
fun display_empty_name_message_if_one_name_empty() {
  writeTo(R.id.et_player1, "")
  writeTo(R.id.et_player2, "William")

  clickDialogPositiveButton()

  assertDisplayed(R.string.game_dialog_empty_name)
}
```

APP: Jogo da Véia - Testes Automatizados - GameBeginDialogTest

Neste teste verificamos se o dialog é fechado após digitarmos nomes válidos

```
@Test
@Throws(Exception::class)
fun close_dialog_after_names_valid() {
  writeTo(R.id.et_player1, "Heider 1")
  writeTo(R.id.et_player2, "William")

  clickDialogPositiveButton()

  assertNotExist(R.id.player1Layout)
}
```

APP: Jogo da Véia - Testes Automatizados - GameEndDialogTest

Neste teste verificamos se exibe vencedor quando o jogo termina.

```
class GameEndDialogTest {

  @Rule @JvmField
  var activityRule: ActivityTestRule<GameActivity> =
ActivityTestRule(GameActivity::class.java)

  private fun givenGameEnded() {
    activityRule.activity.onGameWinnerChanged(Player("Heider", "x"))
  }

  @Test
  @Throws(Exception::class)
  fun display_winner_when_game_ends() {
     givenGameEnded()

     assertDisplayed(R.id.tv_winner)
  }
}
```

APP: Jogo da Véia - Testes Automatizados - GameEndDialogTest

Neste teste verificamos se exibe o dialog para começar um novo jogo após clicar em Done do jogo finalizado.

```
@Test
@Throws(Exception::class)
fun display_begin_dialog_when_done_clicked() {
  givenGameEnded()

  clickDialogPositiveButton()

  assertNotExist(R.id.tv_winner)
  assertDisplayed(R.id.et_player1)
}
```

Copyright © 2019 Heider Lopes e William Cisang
Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, dos Instrutores.