



UNIDADE III

Desenvolvimento para
Dispositivos Móveis

Prof. MSc. Olavo Ito

Desenvolvimento de Aplicativo Android com API Gemini

Desenvolvimento de Aplicativo Android com API Gemini

- Objetivo: Criar um aplicativo Android para sugestões de roteiros turísticos personalizados.

Aplicativo TuristAI – Automação de Roteiros Turísticos

Princípio do Aplicativo:

- Automação do processo de consulta de roteiros turísticos.
- Utilização da Inteligência Artificial (IA) – API Gemini do Google.

Processo Automatizado:

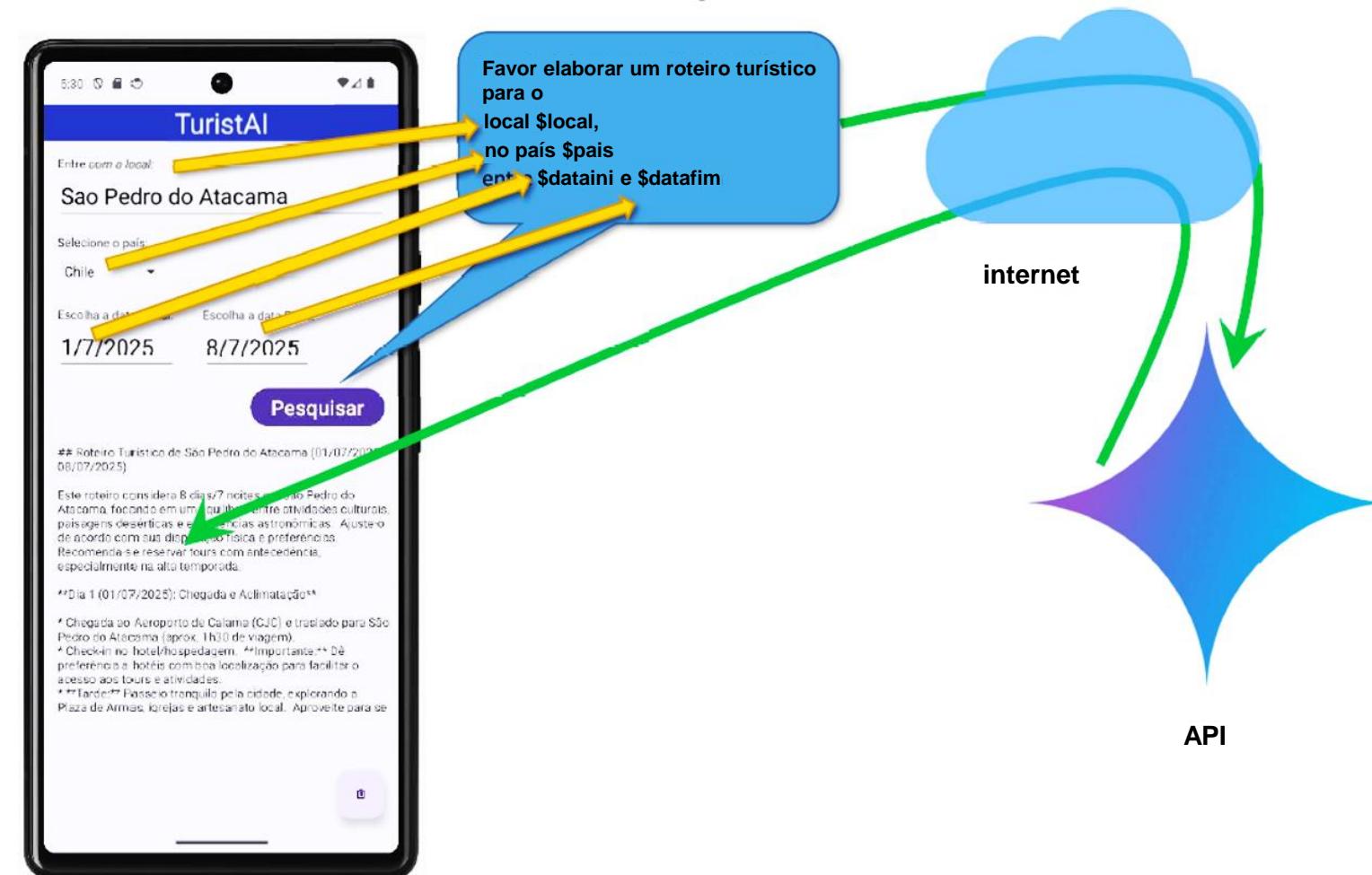
- Pergunta padrão: “Favor elaborar um roteiro turístico para o local \$local, no país \$pais entre \$dataini e \$datafim”.



Aplicativo TuristAI – Automação de Roteiros Turísticos

Fluxo de Informações:

- Envio da pergunta ao servidor do Gemini.
- Processamento e apresentação do resultado.
- Utilização de API para consultas programáticas.
- Resultado apresentado na região dedicada do aplicativo.



Criar o Projeto na Plataforma Android

Criar o Projeto na Plataforma Android

Técnica utilizada:

- *Views para uma tela estática com campos em localizações fixas.*

Benefícios:

- Interface de usuário simples e direta.
- Boa usabilidade com elementos fixos.

Passos iniciais:

- Criar um novo projeto.
- Definir o nome do aplicativo: ‘App TuristAI’.
- Pacote sugerido: com.example.appturistai.
- Linguagem de programação: Kotlin.
- Mínimo nível de API suportado.

Registrar o Projeto

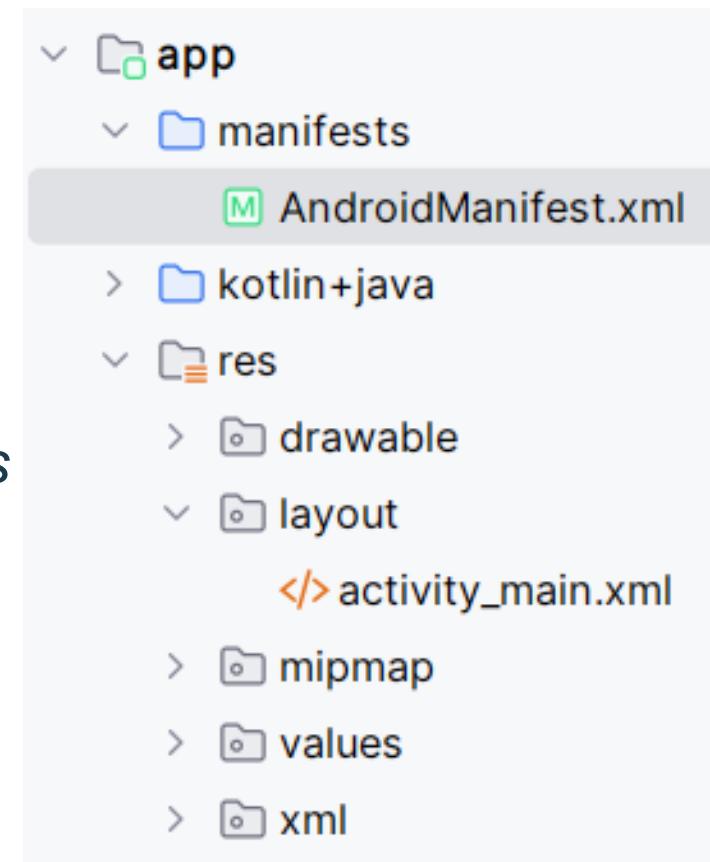
Registrar o Projeto

Publicação na Google Play:

- Informações essenciais no início do projeto.
- Localização das informações no projeto.

Arquivo AndroidManifest.xml:

- Localização: Raiz do conjunto de fontes do projeto.
- Função: “Carteira de identidade” do app, *fornecendo informações essenciais sobre o aplicativo ao sistema operacional.*
- Componentes declarados: Atividades, serviços, broadcast receivers, content providers.
 - Permissões, filtros de intent, configurações importantes.

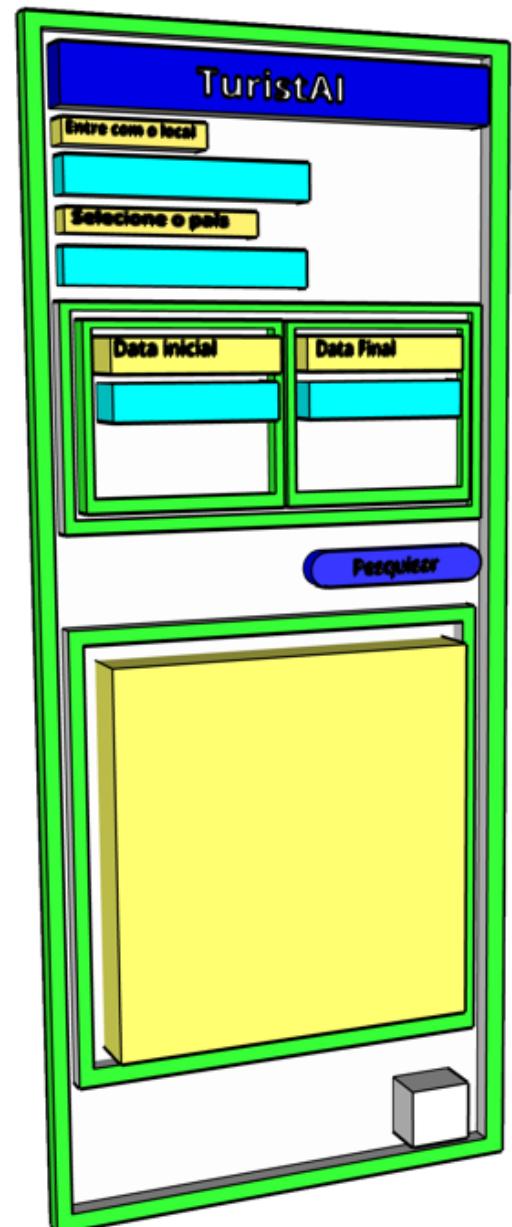


Implementar a Activity

Implementar a Activity

Activity principal:

- Facilita a navegação do usuário para o planejamento de sua viagem.
- Usuário informa o local e país de destino, escolhe as datas da viagem e inicia a pesquisa por roteiros turísticos.
- Área de resultados exibe o roteiro sugerido.
- Botão para limpar os dados inseridos.



Layout da Activity Principal

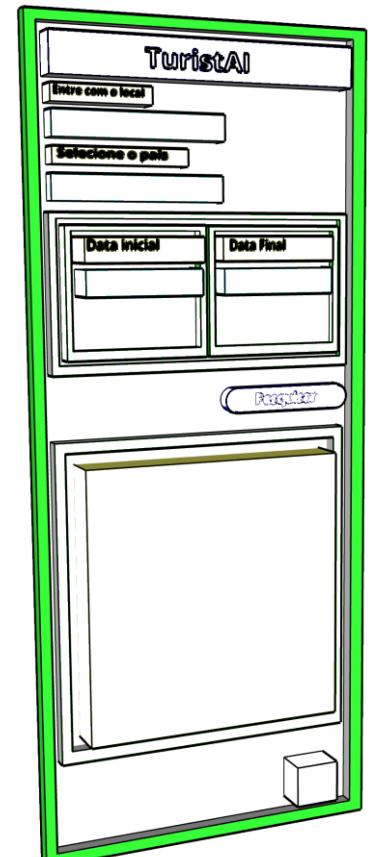
LinearLayout Principal: Organiza os elementos sequencialmente

- Orientação vertical.
- Organiza os elementos da tela em uma única coluna.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    .
    .
    .

</LinearLayout>
```



Cabeçalho do Aplicativo TuristAI

TextView na Parte Superior:

Mostra um texto na tela:

- Preenche a largura da tela (match_parent).
- Exibe o logo do aplicativo “TuristAI” centralizado.
- Fundo azul (#3F51B5).
- Texto branco (@color/white).

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:background="#3F51B5"  
    android:text="TuristAI"  
    android:textAlignment="center"  
    android:textColor="@color/white"  
    android:textSize="32dp" />
```



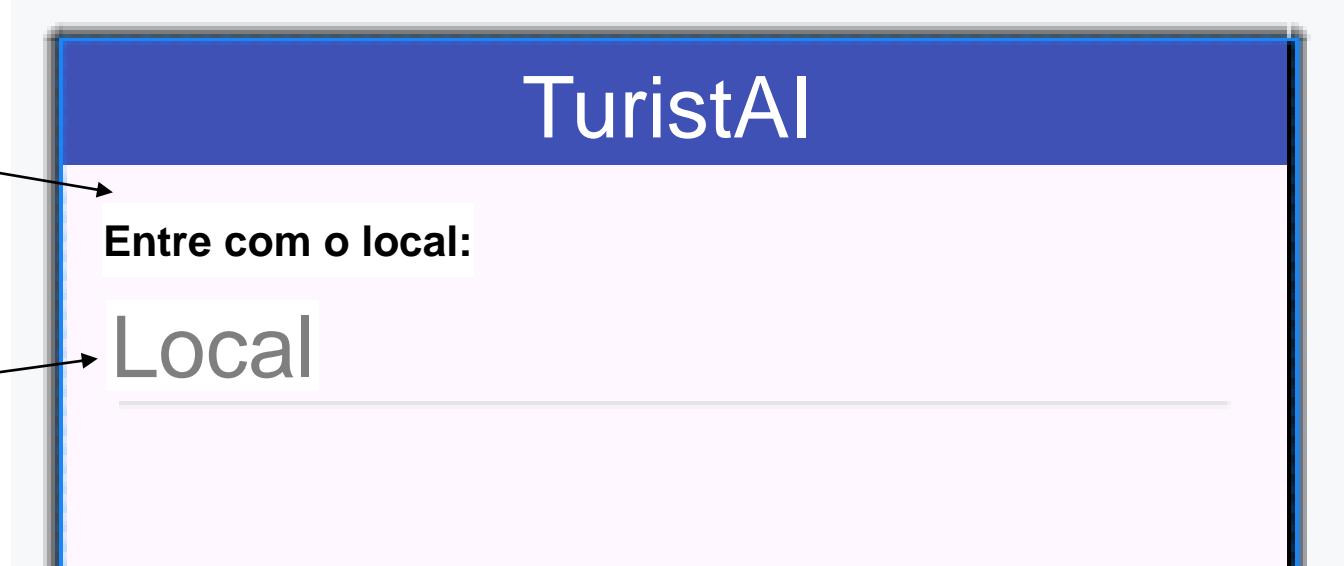
Local e País

TextView: Mostra um texto na tela

- Texto: “Entre com o local:”
- ID: @+id/textView

EditText: permite inserir e editar texto

- ID: @+id/etLocal
- Largura: fill_parent
- Dica: “Local”



Local e País

TextView: exibe texto na tela

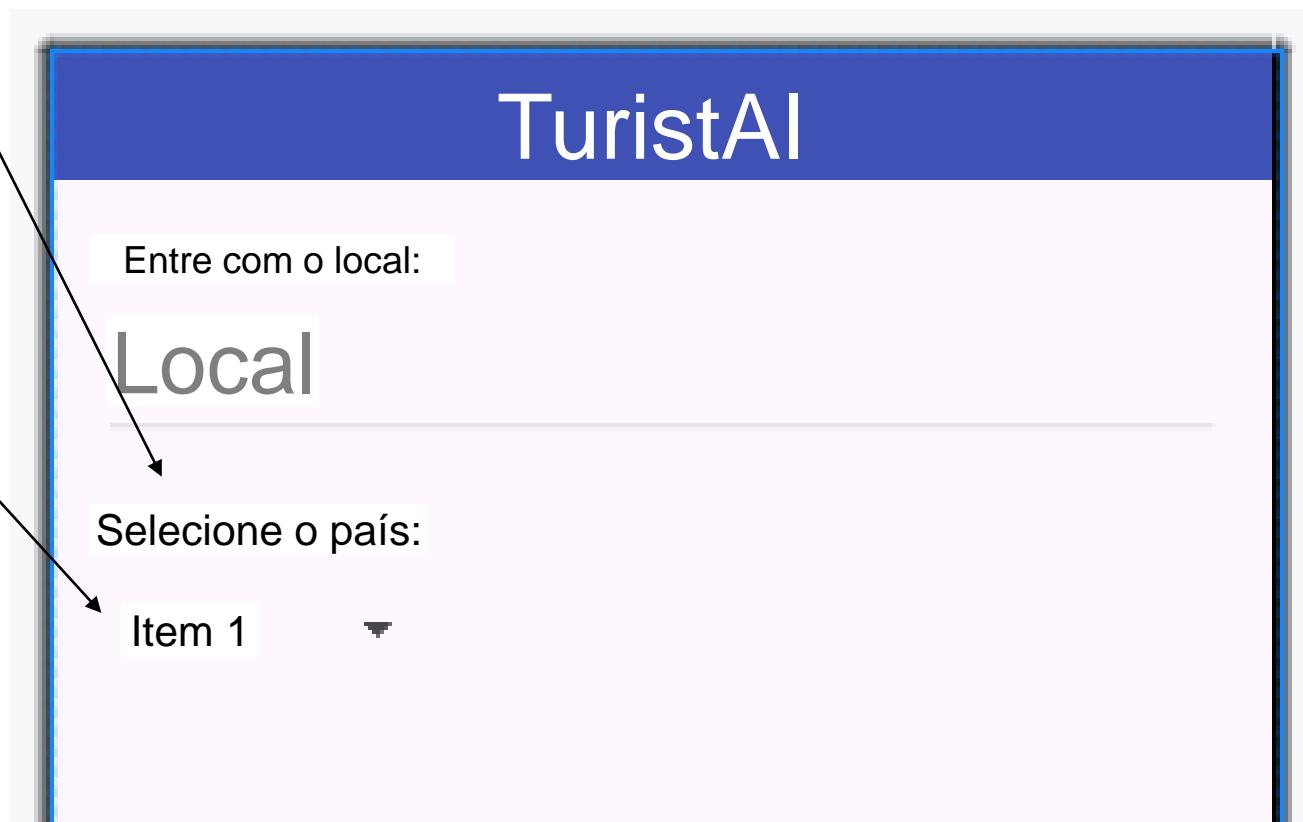
- Texto: “Selecione o país:”

Spinner: permite ao usuário selecionar um item de uma lista suspensa.

- ID: @+id/spinner

<Spinner

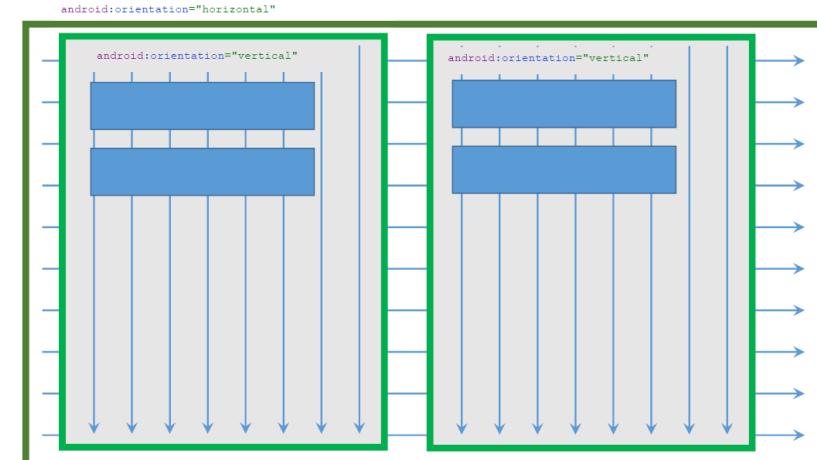
```
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="48dp"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp" />
```



Datas da Viagem

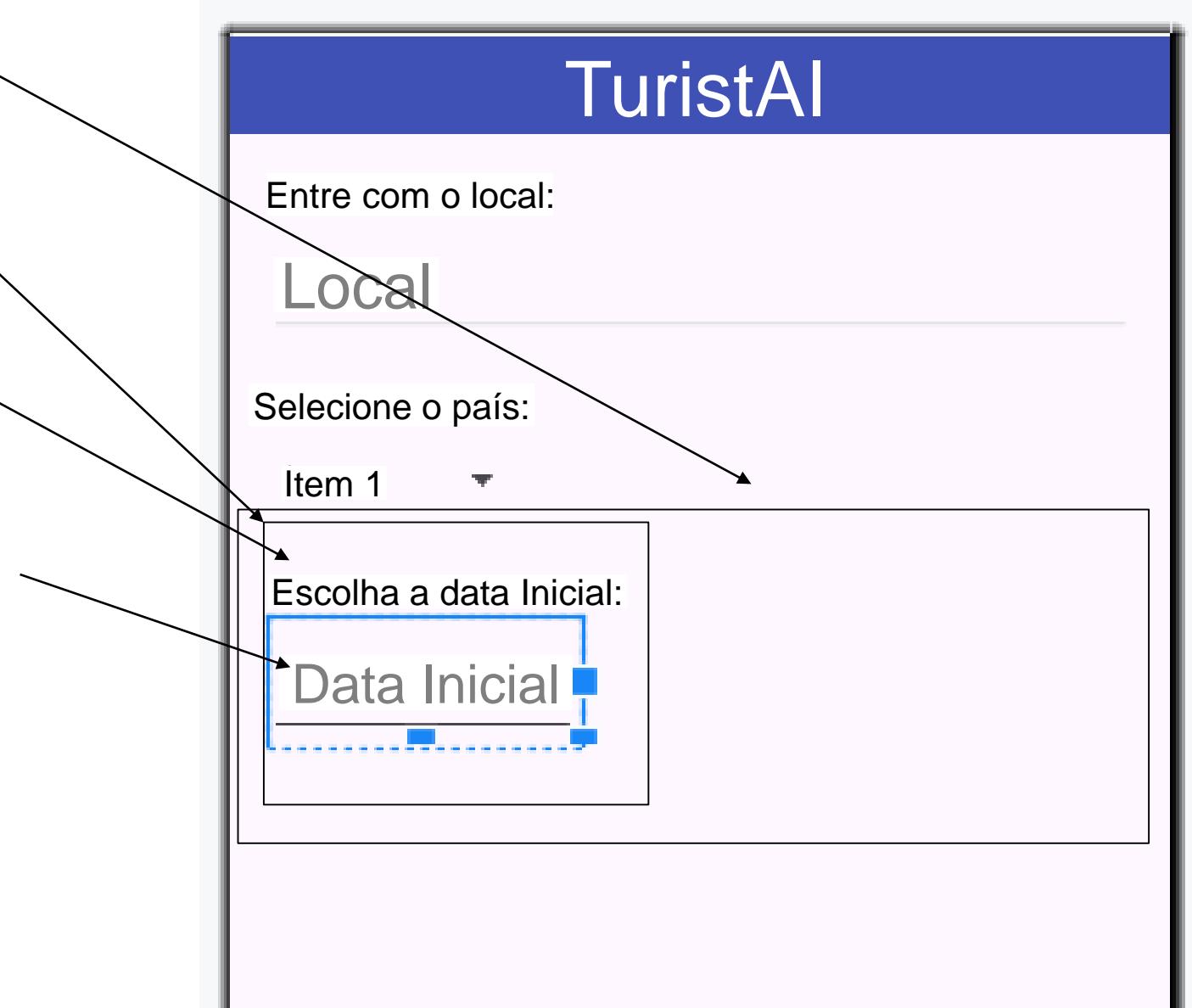
- **Posicionamento sofisticado:**
 - Texto da data inicial acima do campo de entrada do calendário.
 - Datas finais dispostas ao lado do conjunto inicial.

```
<LinearLayout  
    android:orientation="horizontal">  
    <LinearLayout  
        android:orientation="vertical">  
            <TextView  
                android:text="Escolha a data Inicial:" />  
            <EditText  
                android:hint="Data Inicial" />  
        </LinearLayout>  
  
        <LinearLayout  
            android:orientation="vertical">  
            <TextView  
                android:text="Escolha a data Final:" />  
            <EditText  
                android:hint="Data Final"  
            />  
        </LinearLayout>  
    </LinearLayout>
```



Datas da Viagem

- Um LinearLayout horizontal organiza os elementos referentes às datas da viagem.
- Um LinearLayout Vertical organiza a data Inicial.
- Um TextView com o texto “Escolha a data Inicial:”.
- Um EditText onde o usuário seleciona a data de início da viagem.



Datas da Viagem

- Repetimos para a Data Final

The screenshot shows a user interface for a travel booking platform named "TuristAI". At the top, there is a blue header bar with the brand name "TuristAI" in white. Below the header, there are two input fields: one for entering a destination ("Entre com o local:" followed by "Local") and one for selecting a country ("Selecione o país:" followed by a dropdown menu labeled "Item 1"). Further down, there are two date selection fields: "Escolha a data Inicial:" (with the placeholder "Data Inicial") and "Escolha a data Final:" (with the placeholder "Data Final"). The "Data Final" field is highlighted with a blue rectangular border, indicating it is the current focus or the next step in the process.

Botão de Pesquisa

- Um Button (botão) com o texto “Pesquisar” posicionado na parte direita da tela (android:layout_gravity="right").

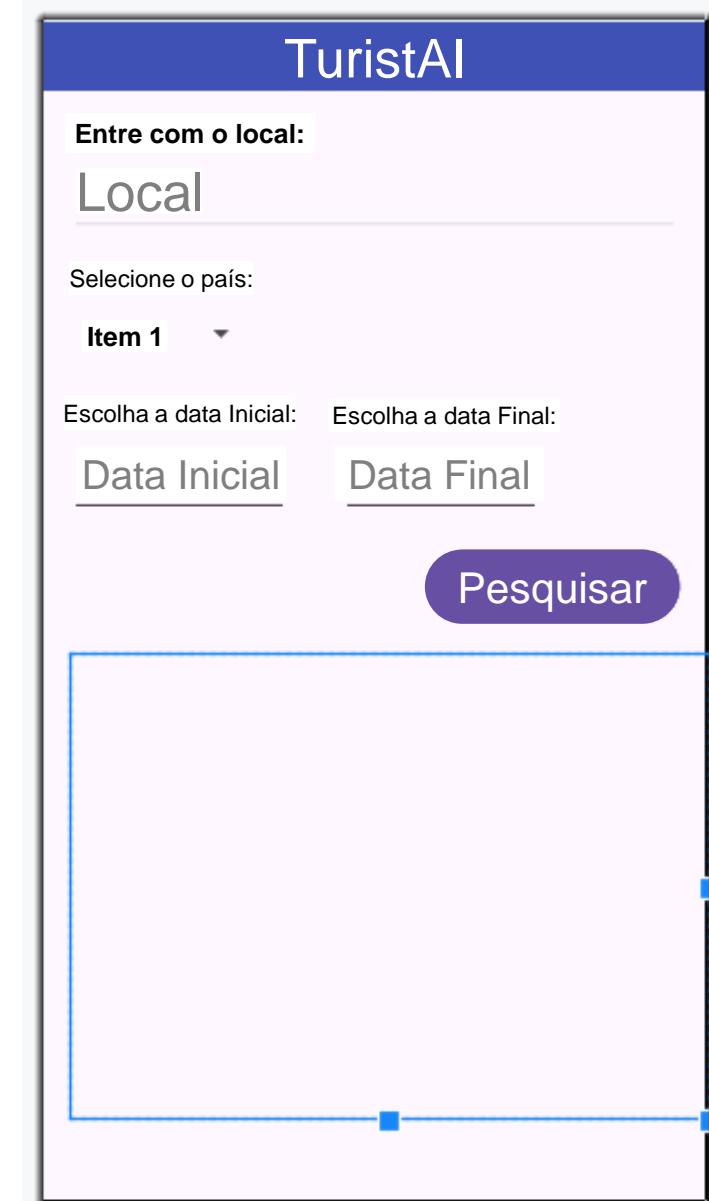
```
<Button  
    android:id="@+id/btOk"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="right"  
    android:layout_marginTop="16dp"  
    android:layout_marginEnd="16dp"  
    android:text="Pesquisar"  
    android:textSize="24sp" />
```



Área de Resultados

- Um (**ScrollView** área com scroll vertical), que permite a rolagem de conteúdo que excede o tamanho da tela, preenche a maior parte da tela (android:layout_weight="0.9").

```
<ScrollView  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="0.9"  
    android:layout_marginStart="16dp"  
    android:layout_marginTop="16dp"  
    android:layout_marginBottom="50dp">  
  
</ScrollView>
```



Botão Flutuante de Ação (FAB)

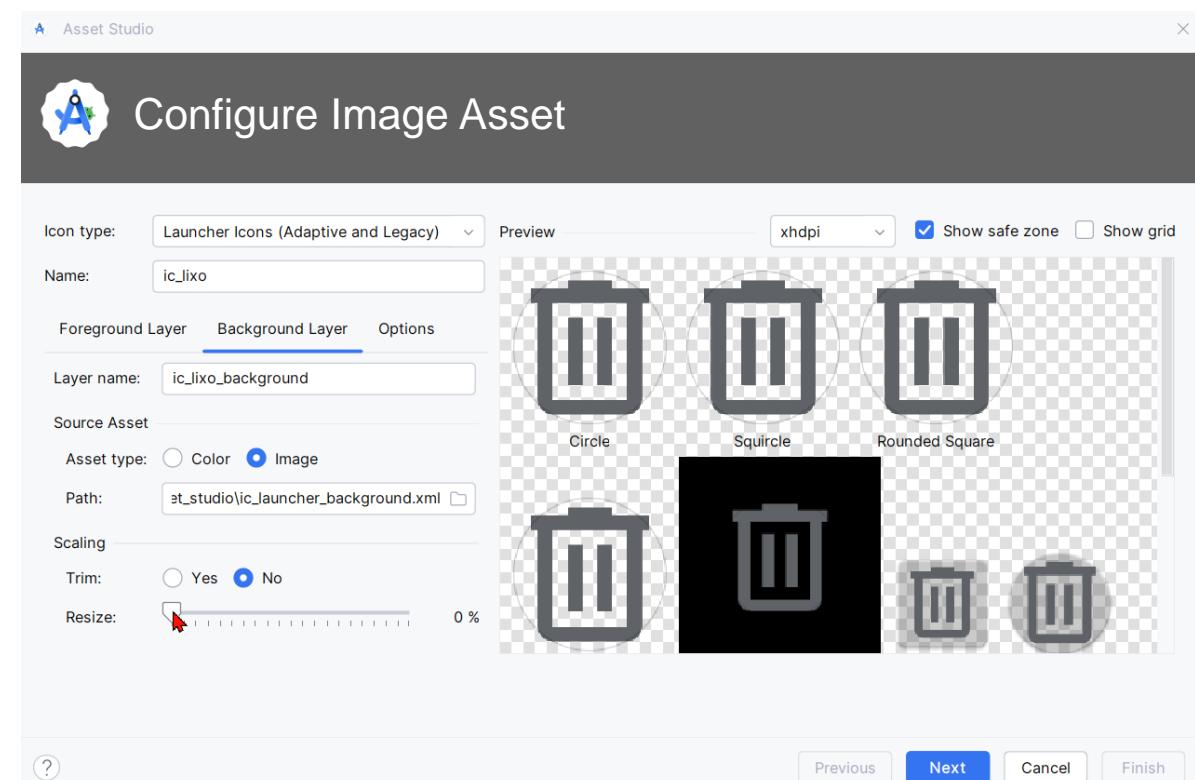
Imagen do Botão:

- Representa uma lixeira (`android:src="@drawable/ic_lixo"`).
- Função: Limpar os dados inseridos pelo usuário.



Criação do Ícone:

1. Na pasta res do projeto, clique com o botão direito e escolha New > Image Asset.
2. Renomeie e insira a nova imagem em path.
3. Na aba Background Layer, diminua o resize para 0% para que o fundo seja transparente.
4. Clique em Next e finalize.



Botão Flutuante de Ação (FAB)

- Um **FloatingActionButton** (botão flutuante de ação) posicionado na parte inferior direita da tela (`android:layout_gravity="bottom|right"`).



Programação e edição do MainActivity

Localizando o MainActivity

The screenshot shows the Android Studio interface. On the left, the Project tool window displays the project structure under the 'Android' tab. It includes the app module with its manifest and kotlin+java directory containing the MainActivity file, which is currently selected. Other visible files include com.example.appturistai (androidTest and test) and various resource directories (res: drawable, layout, mipmap, values, xml) and generated files. The bottom of the Project window shows Gradle Scripts. The main editor area on the right contains the MainActivity.kt code:

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContentView(R.layout.activity_main)  
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->  
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
            insets  
        }  
    }  
}
```

Variáveis de Trabalho

```
//Variáveis de Trabalho
var frase: String = "" //frase de envio ao API
var pais: String = "" //armazenar o país selecionado no
Spinner
```

Configuração da Entrada do Local

Configuração da Entrada do Local:

- etLocal: Campo de texto onde o usuário digita o local desejado.
- Variável do tipo EditText chamada etLocal.

```
var etLocal: EditText //campo para digitar o local
```

- Associação da variável etLocal ao elemento da interface com o ID etLocal.

```
etLocal = findViewById(R.id.etLocal) //localiza o campo com o id etLocal no layout
```

Configuração do Botão de Pesquisa

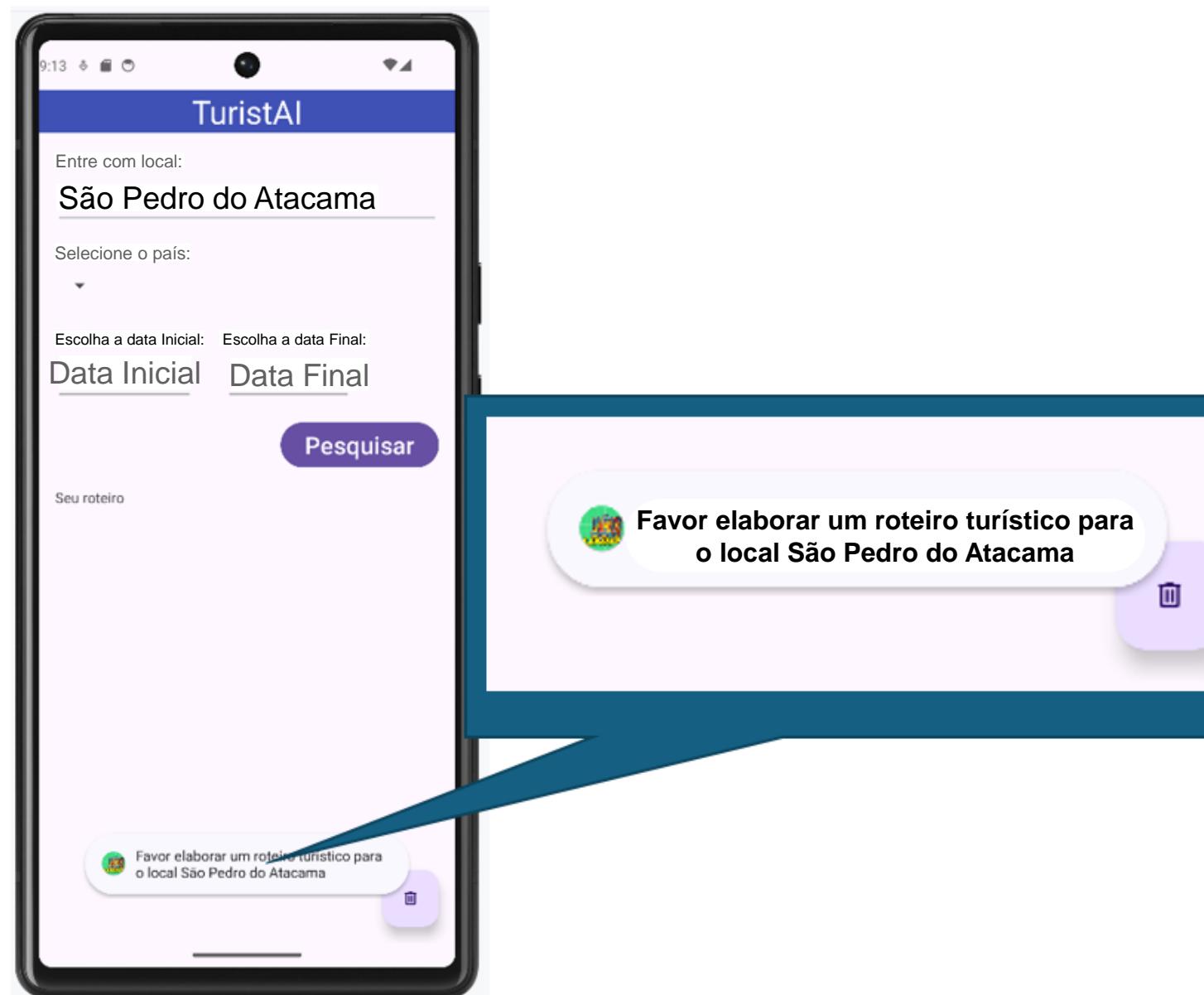
Configuração do Botão de Pesquisa

Configuração do Botão de Pesquisa:

- `btOk`: Uma variável do tipo Button chamada btOk. Um Button é um botão que, quando clicado, executa uma ação (Button).

```
btOk.setOnClickListener {  
    var local: String = etLocal.text.toString() //armazenar o local digitado  
    frase = "Favor elaborar um roteiro turístico para o local $local"  
    Toast.makeText(this, frase, Toast.LENGTH_LONG).show()  
}
```

Ações iniciais ao clicar o Botão de Pesquisa



Configuração da Lista de Países

Configuração da Lista de Países

Declaração do Array de Países:

- Define a lista de países disponíveis para o usuário selecionar.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
        //Variáveis de Trabalho
        var frase: String = "" //frase de envio ao API
        var pais: String = "" //armazenar o país selecionado no Spinner
        //Variáveis de local:
        var etLocal: EditText
        etLocal = findViewById(R.id.etLocal)
        var local: String //armazenar o local digitado
        //=====Tratamento da lista dePaíses
        //Variáveis de listaPaises:
    }

    var paises = arrayOf("Brasil", "Argentina", "Chile", "Uruguai", "Paraguai")

    listaPaises.adapter = adapter
    listaPaises.setListAdapter(adapter)
    listaPaises.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>?,
            view: View?,
            position: Int,
            id: Long
        ) {
            //retorna a posição do item selecionado
            pais = paises[position]
            Toast.makeText(this@MainActivity, "País Selecionado: $pais", Toast.LENGTH_SHORT)
                .show()
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {
            TODO("Not yet implemented")
        }
    } //=====Tratamento do botão Pesquisar
    var btOk: Button
    btOk = findViewById(R.id.btOk)
    btOk.setOnClickListener {
        var local: String = etLocal.text.toString() //armazenar o local digitado
        frase = "Favor elaborar um roteiro turístico para o local $local"
        Toast.makeText(this, frase, Toast.LENGTH_LONG).show()
    }
}
```

Tratamento do Spinner

No processo para o tratamento do Spinner, são necessárias as seguintes etapas:

1. Criar o spinner.
2. Criar e configurar o adapter (converte os dados em uma forma visual para o Spinner).
3. Associar o adapter ao Spinner.
4. Tratar a seleção do usuário.
5. Tratar do item não selecionado.

```
//Variáveis de listaPaises:  
var países = arrayOf("Brasil", "Argentina", "Chile", "Uruguai", "Paraguai")  
var listaPaises: Spinner  
listaPaises = findViewById(R.id.spinner)  
//O spinner precisa receber um adapter para que ele possa exibir os dados  
var adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, países)  
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)  
listaPaises.setAdapter(adapter)
```

Tratamento do Spinner

- Tratar a seleção do usuário:
onItemSelected
- Tratar do item não selecionado
onNothingSelected

```
listaPaises.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(
        parent: AdapterView<*>?, view: View?, position: Int, id: Long
    ) {
        //retorna a posição do item selecionado
        pais = paises[position]
        Toast.makeText(this@MainActivity, "País Selecionado: $pais", Toast.LENGTH_SHORT)
            .show()
    }
    override fun onNothingSelected(parent: AdapterView<*>?) {
        TODO("Not yet implemented")
    }
}
```

Tratamento de Data Inicial e Final

Tratamento de Data Inicial e Final

- Inicialização
- `etDataIni`: Representa o campo de texto onde o usuário insere a data inicial da viagem (EditText).
- `dataini`: Armazena a data inicial da viagem como string.
- `etDataFim`: Representa o campo de texto onde o usuário insere a data final da viagem (EditText).
- `datafim`: Armazena a data final da viagem como string.

```
//=====Tratamento da data inicial e final
var etDataIni: EditText
var etDataFim: EditText
var dataini: String
var datafim: String
//
var dataAux: String
var cal: Calendar
//
var datePickerDialog: DatePickerDialog
etDataIni = findViewById(R.id.etDataIni)
etDataFim = findViewById(R.id.etDataFim)
cal = Calendar.getInstance()
etDataIni.setOnClickListener {
    showDatePickerDialog(this, etDataIni, cal)
}
etDataFim.setOnClickListener {
    showDatePickerDialog(this, etDataFim, cal)
}

var txtSaida: TextView
txtSaida = findViewById(R.id.txtRetorno)
```

//etDataIni é o id do objeto Data Inicial
//etDataFim é o id do objeto Data Final
//cal é o objeto Calendar

Objeto Calendar

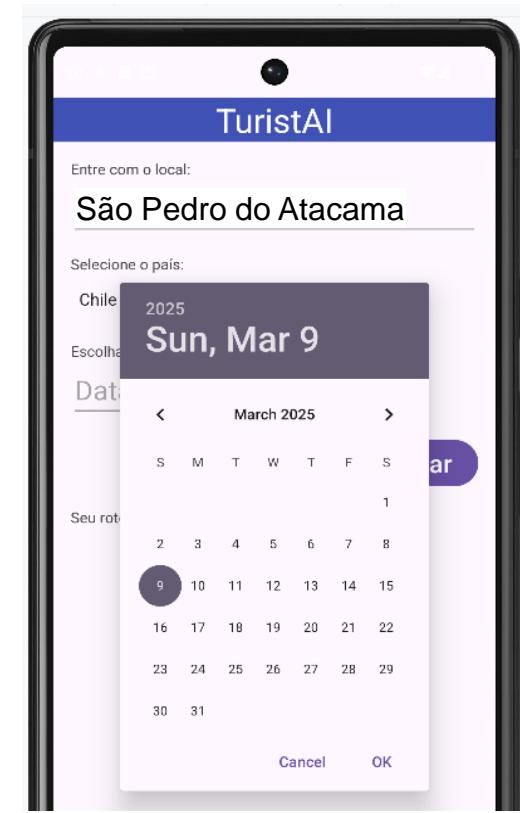
- **cal:** Objeto `Calendar` usado para manipulação de datas.

```
//  
var cal: Calendar  
//  
    var datePickerDialog: DatePickerDialog  
    etDataIni = findViewById(R.id.etDataIni) //etDat  
    etDataFim = findViewById(R.id.etDataFim) //etDat  
    cal = Calendar.getInstance() //cal  
    etDataIni.setOnClickListener {  
        showDatePickerDialog(this, etDataIni, cal)  
    }  
    etDataFim.setOnClickListener {  
        showDatePickerDialog(this, etDataFim, cal)  
    }  
var txtSaida: TextView  
txtSaida = findViewById(R.id.txtRetorno)
```

OnDateSetListener

- datePickerDialog.show(): Exibição do DatePickerDialog

```
fun showDatePickerDialog(context: Context, editText: EditText) {
    val cal = Calendar.getInstance()
    val ano = cal.get(Calendar.YEAR)
    val mes = cal.get(Calendar.MONTH)
    val dia = cal.get(Calendar.DAY_OF_MONTH)
    val datePickerDialog = DatePickerDialog(
        context,
        { view, ano, mes, dia ->
            val dataSelecionada = "$dia/${mes + 1}/$ano"
            editText.setText(dataSelecionada)
            Toast.makeText(context, dataSelecionada,
                Toast.LENGTH_SHORT).show()
        },
        ano,
        mes,
        dia
    )
    datePickerDialog.show()
}
```



Interatividade

O aplicativo TuristAI utiliza quais interfaces?

- a) Kotlin e Android Studio.
- b) Java e Views.
- c) Kotlin, Jetpack Compose.
- d) Javascript, Jetpack Compose e Android Studio.
- e) Android Studio.

Resposta

O aplicativo TuristAI utiliza quais interfaces?

- a) Kotlin e Android Studio.
- b) Java e Views.
- c) Kotlin, Jetpack Compose.
- d) Javascript, Jetpack Compose e Android Studio.
- e) Android Studio.

Saída de Texto

Declaração da Saída de Texto

Preparação do TextView:

- txtSaida: TextView para exibir o resultado da consulta ao Gemini.
- Definição da variável txtSaida.
- Associação ao elemento da interface com o ID txtRetorno.

```
//=====Definindo o TextView de saída
var txtSaida: TextView
txtSaida = findViewById(R.id.txtRetorno)
```

IA – implementar a API Gemini:

Implementar a API Gemini

Limitações dos Dispositivos Móveis:

- Capacidade de processamento e armazenamento limitada.

Função das APIs:

- *Conectar o aplicativo a serviços em nuvem para processar e gerar informações.*
- Utilizar servidores com alto poder de processamento.
- Garantir acesso a um banco de dados vasto e atualizado.

Etapas:

1. Configurar o Ambiente de Desenvolvimento.
2. Obter uma Chave de API.
3. Implementar a Lógica de Consulta.
4. Obter a Resposta.
 - Página de apoio ao API Gemini.
 - No Google, procurar palavras-chave:
 - Documentação Gemini API Android Kotlin português

The screenshot shows a Google search results page. The search query is "Documentação Gemini API Android Kotlin português". The top result is a link to a tutorial titled "Tutorial: primeiros passos com a API Gemini". The snippet below the title explains that the tutorial demonstrates how to access the API directly from an Android app using the Google AI client library.

Google

Documentação Gemini API Android Kotlin português >

Todas Vídeos Shopping Imagens Notícias Web Livros Mais

Gemini Developer API
https://ai.google.dev › tutorials › android_quickstart › u... :

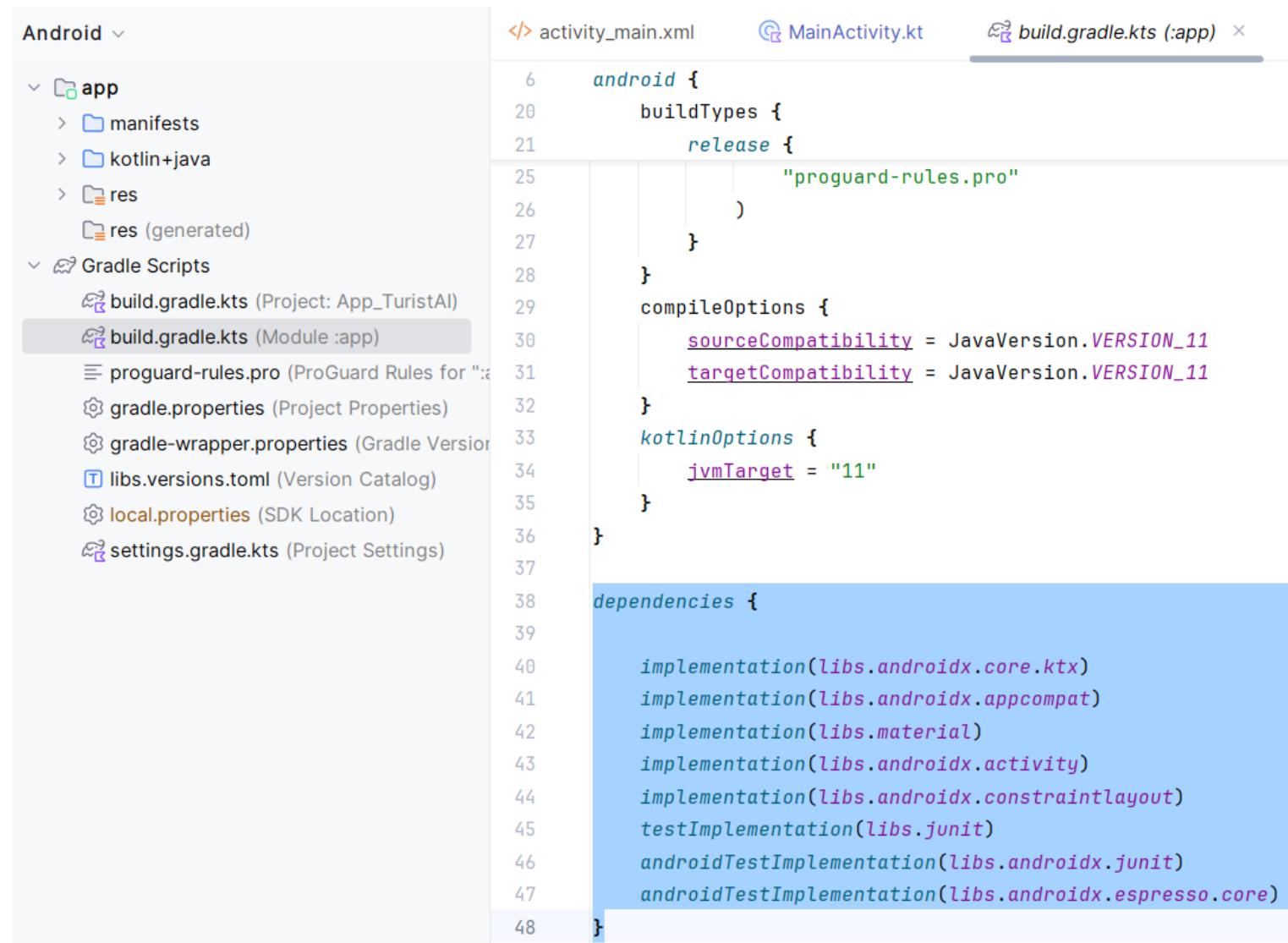
[Tutorial: primeiros passos com a API Gemini](#)

Este tutorial demonstra como acessar a API Gemini diretamente do seu App Android usando o SDK cliente da IA do Google para Android. Você pode usar isso SDK do ...

Configurar o Ambiente de Desenvolvimento

Adicionar Dependência do SDK:

- Editar o arquivo **build.gradle** do aplicativo.
- Incluir as dependências do SDK copiadas do site.



The screenshot shows the Android Studio interface. On the left, the project structure is displayed under the 'Android' tab, showing the app module with its sub-directories (app, manifests, kotlin+java, res) and files (build.gradle.kts, proguard-rules.pro, gradle.properties, gradle-wrapper.properties, libs.versions.toml, local.properties, settings.gradle.kts). The 'build.gradle.kts' file is currently selected. On the right, the code editor displays the contents of the build.gradle.kts file, which is written in Kotlin. The file contains configuration for build types, compile options, Kotlin options, and dependencies, including implementations for androidx.core.ktx, androidx.appcompat, androidx.material, androidx.activity, androidx.constraintlayout, junit, and espresso.core.

```
activity_main.xml MainActivity.kt build.gradle.kts (:app)  
Android  
└ app  
    ├── manifests  
    ├── kotlin+java  
    ├── res  
    └── res (generated)  
Gradle Scripts  
└ build.gradle.kts (Module :app)  
    └ build.gradle.kts (Project: App_TuristAI)  
        ├── proguard-rules.pro (ProGuard Rules for "App_TuristAI")  
        ├── gradle.properties (Project Properties)  
        ├── gradle-wrapper.properties (Gradle Version)  
        ├── libs.versions.toml (Version Catalog)  
        ├── local.properties (SDK Location)  
        └── settings.gradle.kts (Project Settings)  
  
6 android {  
20     buildTypes {  
21         release {  
25             "proguard-rules.pro"  
26         }  
27     }  
28 }  
29 compileOptions {  
30     sourceCompatibility = JavaVersion.VERSION_11  
31     targetCompatibility = JavaVersion.VERSION_11  
32 }  
33 kotlinOptions {  
34     jvmTarget = "11"  
35 }  
36 }  
37  
38 dependencies {  
39     implementation(libs.androidx.core.ktx)  
40     implementation(libs.androidx.appcompat)  
41     implementation(libs.material)  
42     implementation(libs.androidx.activity)  
43     implementation(libs.androidx.constraintlayout)  
44     testImplementation(libs.junit)  
45     androidTestImplementation(libs.androidx.junit)  
46     androidTestImplementation(libs.androidx.espresso.core)  
47 }  
48 }
```

Implementar a Lógica de Consulta

Definição da Função:

- **fun chamaGemini(txtSaida: TextView, frase: String):**

Define uma função chamada chamaGemini que recebe dois parâmetros: txtSaida (um TextView onde o resultado será exibido) e frase (uma String que será usada como prompt para a API).

```
fun chamaGemini(txtSaida: TextView, frase: String) {  
    var generativeModel =  
        GenerativeModel(  
            // Specify a Gemini model appropriate for your use  
            modelName = "gemini-1.5-flash",  
            // Access your API key as a Build Configuration va  
            apiKey = "AIzaSyB3te0Ggf33T8wq6pMRc8GmVPAXXXXXX"  
        )  
    val prompt = frase  
    MainScope().launch {  
        val response = generativeModel.generateContent(prompt)  
        txtSaida.setText(response.text)  
    }  
}
```

Chamada à Função chamaGemini

Implementação no Botão:

- chamaGemini(txtSaida, frase)

```
btOk.setOnClickListener {
    dataini = etDataIni.text.toString()
    datafim = etDataFim.text.toString()
    local = etLocal.text.toString()
    frase =
        "Favor elaborar um roteiro turístico para o local
        $local, no país $pais entre $dataini e $datafim"
    Toast.makeText(this, frase, Toast.LENGTH_SHORT).show()
    chamaGemini (txtSaida, frase)
}
```

Pesquisar

Roteiro Turístico: San Pedro de Atacama (01/04/2025 - 10/04/2025)

Este roteiro considera 10 dias em San Pedro de Atacama, incluindo atividades diurnas e noturnas, com opções para diferentes níveis de aventura e orçamento. Ajuste-o de acordo com suas preferências e nível de condicionamento físico. **É crucial reservar tours e acomodações com antecedência,** principalmente para a alta temporada (abril pode ser alta temporada).

Dia 1 (01/04/2025): Chegada e Acomodação

* Chegada ao Aeroporto de Calama (CJC) e translado para San Pedro de Atacama (aprox. 1h30 de viagem). Considere um transfer privado ou ônibus.

* Check-in no hotel/hospedagem escolhida. Recomenda-se procurar opções próximas ao centro para facilitar o acesso a restaurantes e agências de turismo.

* Passeio tranquilo pelo centro de San Pedro: explore a Plaza



Configuração do Botão de Limpar

Botão de Limpeza:

- FAB (Floating Action Button).
- Elemento visual comum em aplicativos Android.
- Ícone circular ou quadrado.
- Realiza ações principais.

```
var fabLixo: View  
fabLixo = findViewById(R.id.fabLixo)  
fabLixo.setOnClickListener {  
    etLocal.setText("")  
    etDataIni.setText("")  
    etDataFim.setText("")  
    txtSaida.setText("")  
}
```

Projeto Pronto



Projeto 2 – Jogo

Introdução ao Jogo Snake

- Contexto histórico:
 - O clássico “jogo da cobrinha”, conhecido como Snake, tem raízes nos arcades de 1976, sob o nome Blockade.
 - Tornou-se um fenômeno cultural nos celulares Nokia.
 - Lançado inicialmente no Nokia 6110, em 1997.
- Popularidade e evolução:
 - Popularidade explodiu com o lançamento do Nokia 3310, em 2000.
 - Vendeu mais de 126 milhões de unidades.
 - Desenvolvido por Taneli Armanto.
 - Adaptado para as limitações dos celulares da época.



Fonte: Livro-texto.

Objetivo e Mecânica do Jogo

- Objetivo: Controlar uma cobra que cresce ao comer alimentos.
 - Primeira versão: Evitar colidir com as bordas e o corpo.
 - Segunda versão: Restrição de colisão com as bordas removidas.
- Mecânica: Cobra se move continuamente, jogador direciona com toques na tela.
- Evolução: Cobra aumenta de tamanho a cada alimento consumido.

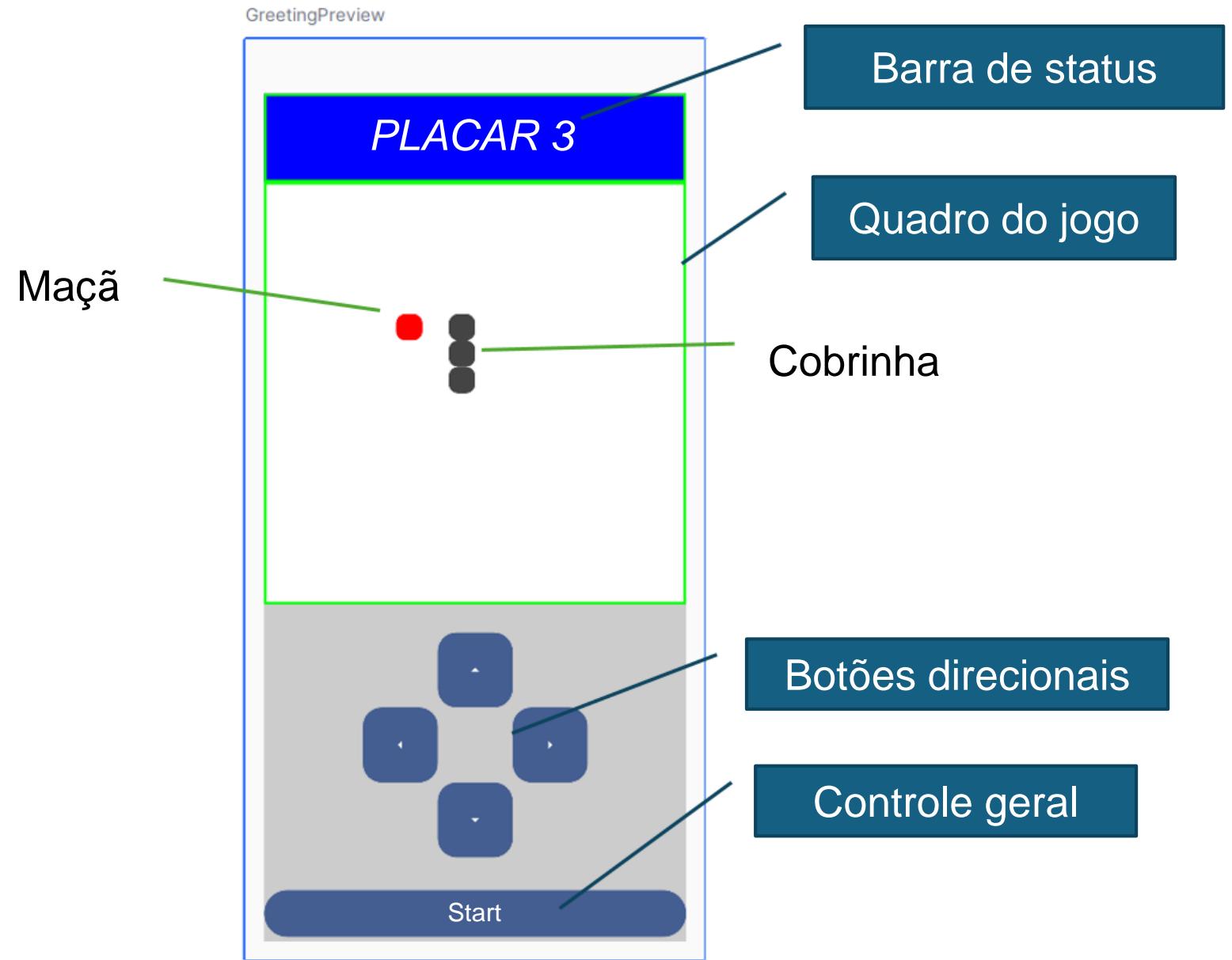
Desenvolvimento com Jetpack Compose

- Jetpack Compose permite criar interfaces de usuário de forma intuitiva e eficiente.
- Definição da interface do jogo de maneira declarativa.
- Facilita a visualização e manutenção do código.

Configuração do Projeto

- Inicie um novo projeto e escolha Empty Activity.
- Dê um nome ao projeto e mantenha todos os outros parâmetros.
- Espere o Gradle ser montado e deixe a tela em modo Split.
- Monte a visualização clicando em “Build & Refresh” ou combinando as teclas “Control-Shift-F5”.
 - Utilize um dispositivo ligado via cabo USB para o emulador ou use o emulador.

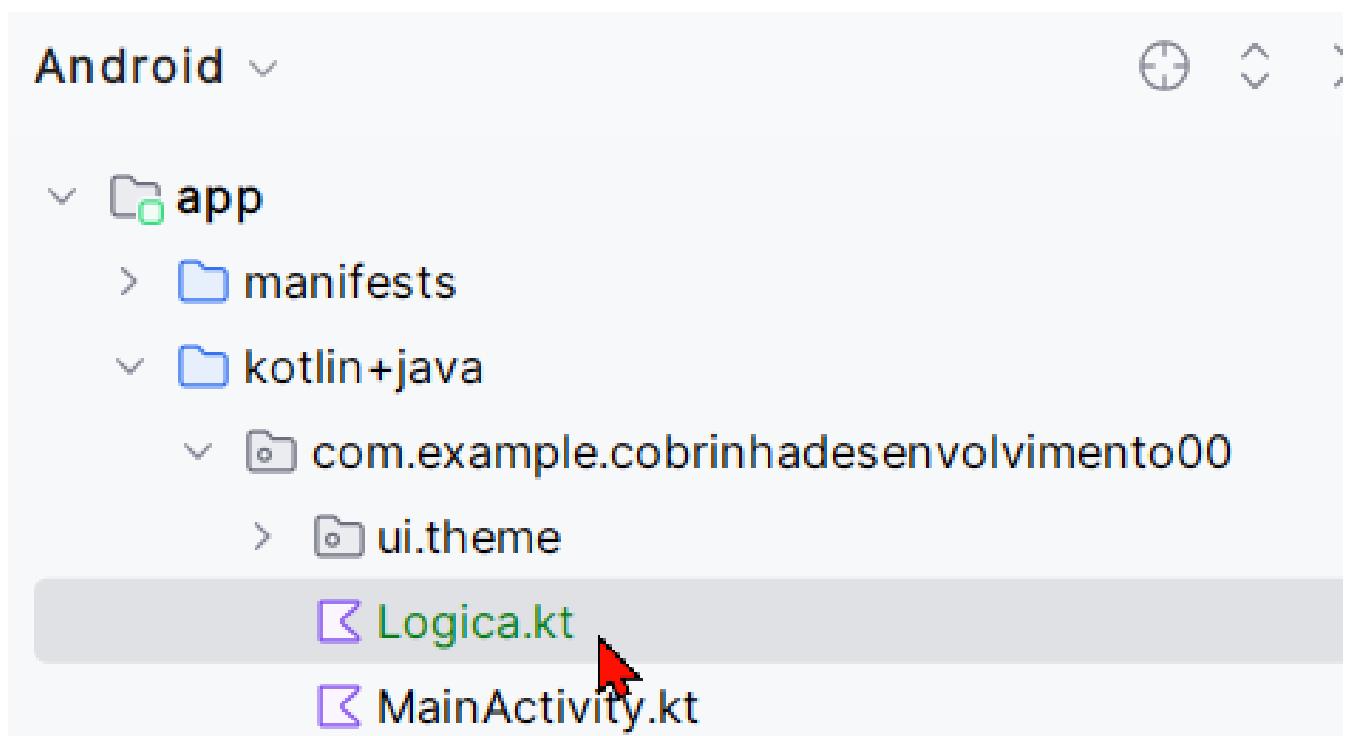
Criar o cenário do jogo



Configurando o Projeto

Configurando o Projeto

- Utilizando o Jetpack Compose para montar a lógica e a tela simultaneamente.
- Separação da lógica em um novo arquivo para facilitar a montagem.
- Para uma melhor organização do nosso projeto, definimos que as telas do aplicativo estarão na MainActivity e a lógica de funcionamento isolada em uma pasta dedicada chamada Logica.
- A pasta Logica conterá a classe Game() com a lógica principal do jogo, *como o cálculo da pontuação e o movimento da cobra.*



class Game() com Compose e Kotlin

class Game()

Na pasta Logica, crie um programa simples:

- **class Game(){}:** Declara uma nova classe chamada Game.
- **var contador = 0:** Propriedade contador do tipo Int, inicializada com 0.
- **fun roda(){}:** Declara uma função chamada roda.
- **while(contador<1000){}**: Loop while que continua enquanto contador < 1000.
- **contador++:** Incrementa o valor de contador em 1 a cada iteração.

```
class Game(){  
    // variáveis  
    var contador = 0  
    // ação  
  
    fun roda(){  
        while(contador<1000){  
            contador++  
        }  
    }  
}
```

Criando o Composable JogoCobrinha

Na MainActivity, crie um componente composable chamado JogoCobrinha:

- **@Composable fun JogoCobrinha():** Declara uma função composta.
- **val game = Game():** Cria uma instância da classe Game.
- **var texto = "":** Declara uma variável texto.
- **game.roda():** Chama o método roda da classe Game.
- **texto = "Jogo da Cobrinha \${game.contador}"**: Atualiza o valor de texto.
- **Text(text = texto)**: Exibe o texto na tela.

```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = Game()
    var texto = ""
    // controle
    game.roda()
    // tela
    texto="Jogo da Cobrinha ${game.contador}"
    Text(text =texto)
}
```

Alterando a Função Greeting

Altere a função Greeting para chamar a função JogoCobrinha:

```
    @Composable
    fun Greeting(name: String, modifier: Modifier = Modifier) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(
                text = "Hello $name!",
                modifier = modifier
            )
            JogoCobrinha()
        }
    }
```

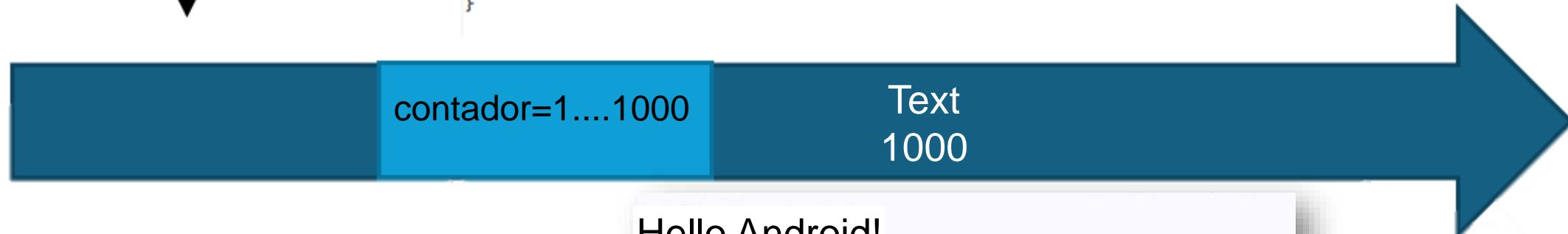
Executando o Emulador

- Execute o emulador e observe que na tela só aparece o número final: 1000.

```
fun JogoCobrinha() {  
    // variáveis  
    val game = Game()  
    var texto = ""  
    // controle  
    game.roda()  
}
```



```
class Game(){  
    // variáveis  
    var contador = 0  
    // ação  
    fun roda(){  
        while(contador<1000){  
            contador++  
        }  
    }  
}
```



Jogo da Cobrinha 1000

Interatividade

Assinale a alternativa **incorreta** sobre a criação da pasta chamada Logica.kt no projeto do jogo além da pasta MainActivity.kt.

- a) Proporciona uma melhor organização do projeto.
- b) A pasta MainActivity.kt privilegia as telas.
- c) A pasta Logica.kt é dedicada ao funcionamento do aplicativo.
- d) A pasta Logica.kt conterá a classe Game.
- e) O Jetpack Compose necessita separar a tela da Lógica de programação.

Resposta

Assinale a alternativa **incorreta** sobre a criação da pasta chamada Logica.kt no projeto do jogo além da pasta MainActivity.kt.

- a) Proporciona uma melhor organização do projeto.
- b) A pasta MainActivity.kt privilegia as telas.
- c) A pasta Logica.kt é dedicada ao funcionamento do aplicativo.
- d) A pasta Logica.kt conterá a classe Game.
- e) O Jetpack Compose necessita separar a tela da Lógica de programação.

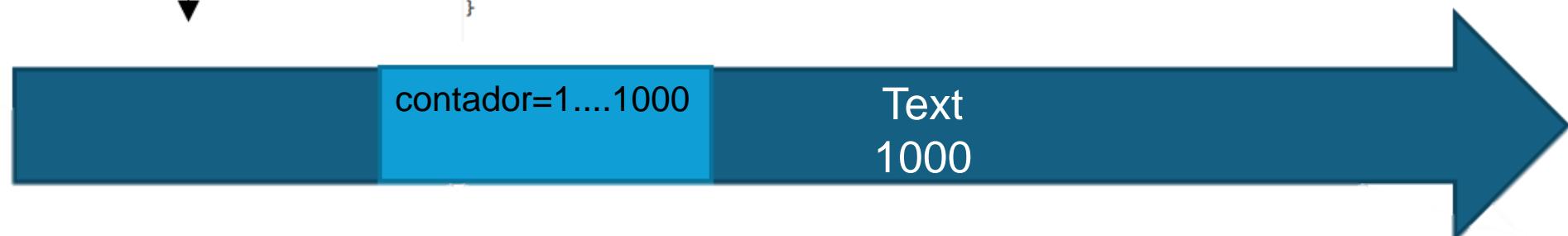
Corrotinas

Problema de Sincronia

- A lógica e a exibição não funcionam em sincronia.
- A tela só é atualizada após o final da função roda.

```
fun JogoCobrinha() {  
    // variáveis  
    val game = Game()  
    var texto = ""  
    // controle  
    game.roda()  
}
```

```
class Game(){  
    // variáveis  
    var contador = 0  
    // ação  
    fun roda(){  
        while(contador<1000){  
            contador++  
        }  
    }  
}
```



Introdução às Corrotinas

- Corrotinas permitem a execução de tarefas de forma concorrente dentro de um único fluxo de controle.
- Simplificam o código assíncrono, tornando-o mais legível e fácil de manter.
- Conceitos-chave das Corrotinas:
 - Lançamento de Corrotinas: usando funções como `launch` ou `async`.
 - Suspensão: funções de suspensão (`suspend functions`) podem ser pausadas e retomadas.
 - Escopos de Corrotinas: definem o contexto no qual as Corrotinas são executadas.

Implementando Corrotinas no Exemplo

- Na função JogoCobrinha(), substitua a instanciação da classe Game
(val game = Game())
- Envolva a chama da função roda()
com o LaunchedEffect

```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = remember{Game()}
    var texto = ""
    // controle
    game.roda()
    // tela
    texto="Jogo da Cobrinha ${game.contador}"
    Text(text =texto)
}
```

```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = remember{Game()}
    var texto = ""
    // controle
    LaunchedEffect(game) {
        game.roda()
    }
    // tela
    texto="Jogo da Cobrinha ${game.contador}"
    Text(text =texto)
}
```

Executando o Emulador

Hello Android!

Jogo da cobrinha 0

```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = remember{Game()}
    var texto = ""

    // controle
    LaunchedEffect(game) {
        game.roda()
    }
}
```

Text
0

Processamento JogoCobrinha

contador=1....1000

```
class Game() {
    // variáveis
    var contador = 0
    // ação
    fun roda(){
        while(contador<1000){
            contador++
        }
    }
}
```

tempo

Problema de Sincronização

- A corrotina foi criada, mas o valor do texto não é atualizado.
- Problemas de sincronização entre os processamentos.
- Usando `mutableStateOf`:
 - Cria um estado observável para contador. O Compose monitora constantemente o valor de contador e, quando ele é modificado, o Compose *dispara uma recomposição, atualizando a interface do usuário de acordo com o novo valor.*
 - Modifique a variável contador da classe Game.

```
class Game () {  
    // variáveis  
    var contador by mutableStateOf(0)  
    // ação  
    fun roda () {  
        while(contador<1000) {  
            contador++  
        }  
    }  
}
```

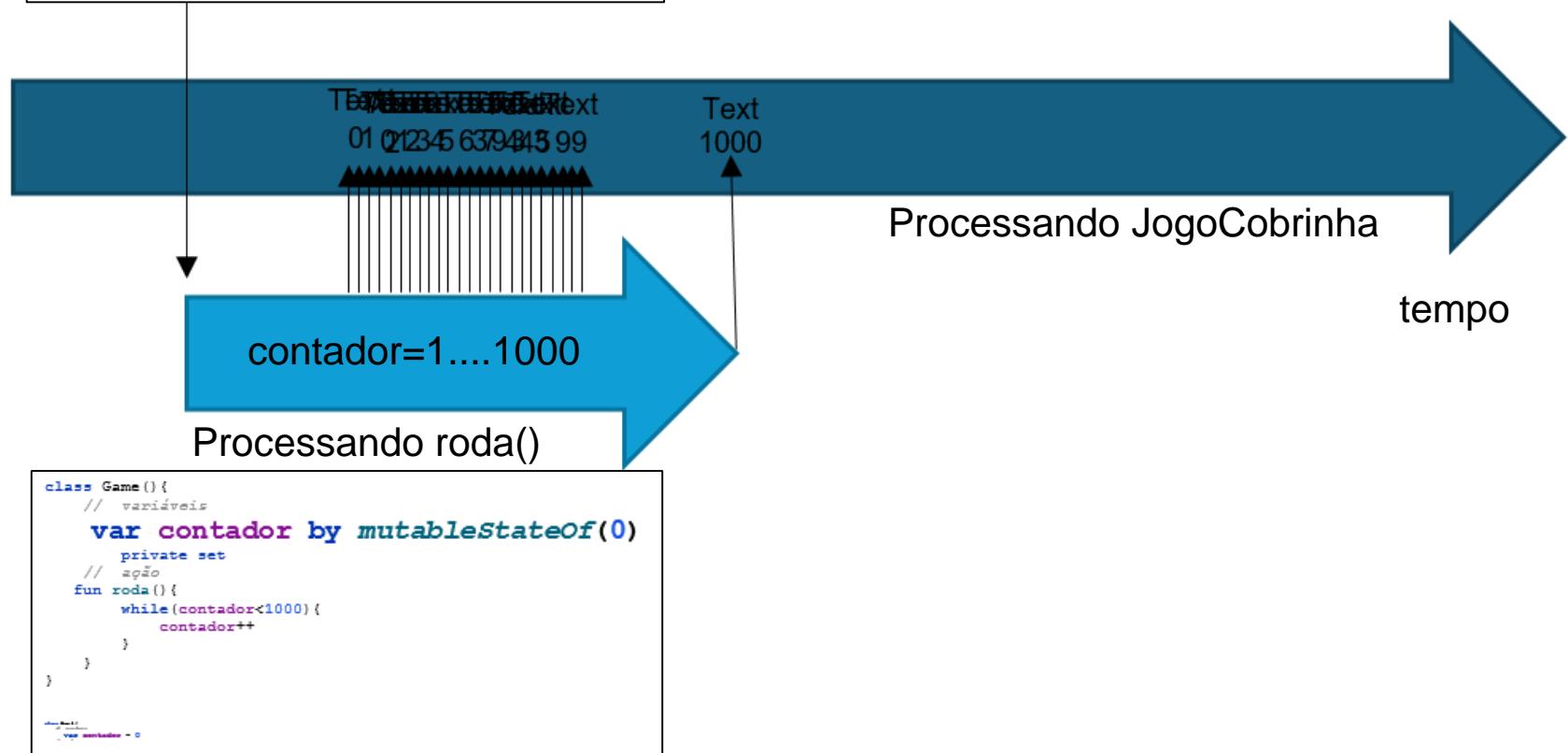
Executando o Emulador

Hello Android!

Jogo da cobrinha 1000

```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = remember{Game()}
    var texto = ""

    // controle
    LaunchedEffect(game) {
        game.rola()
    }
}
```

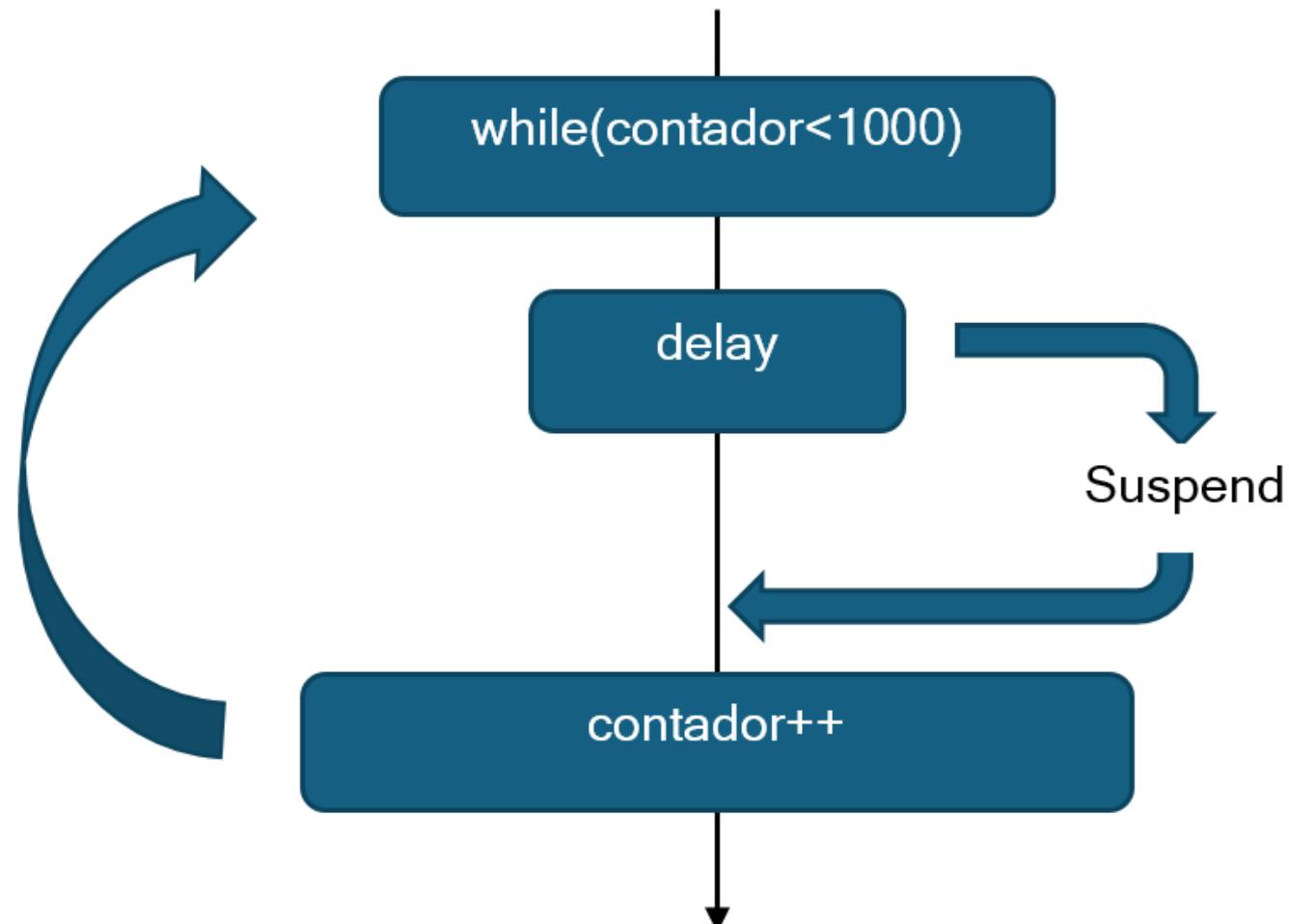


Problema de Atualização Rápida

- O mutableState envia atualizações muito rapidamente, sobrecarregando a UI.
- A interface do usuário exibe apenas o estado final.

Função delay:

- A função delay é usada em corrotinas para pausar a execução por um período de tempo sem bloquear a thread em que a corrotina está sendo executada.

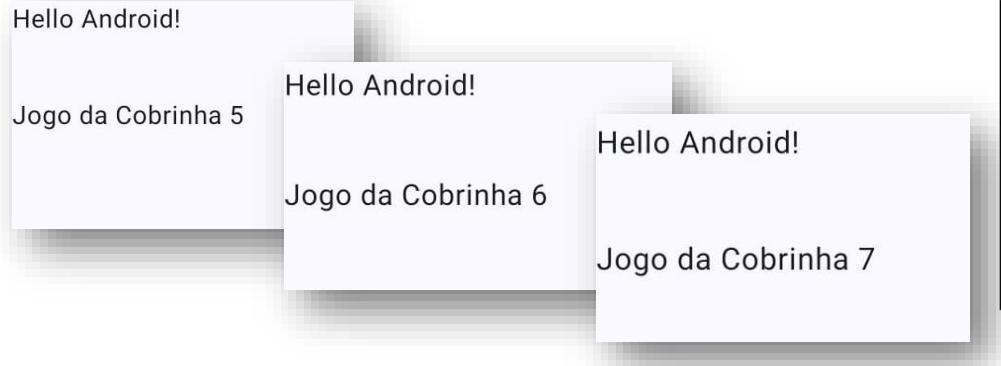


Usando a Função Delay

Altere a função roda para incluir delay:

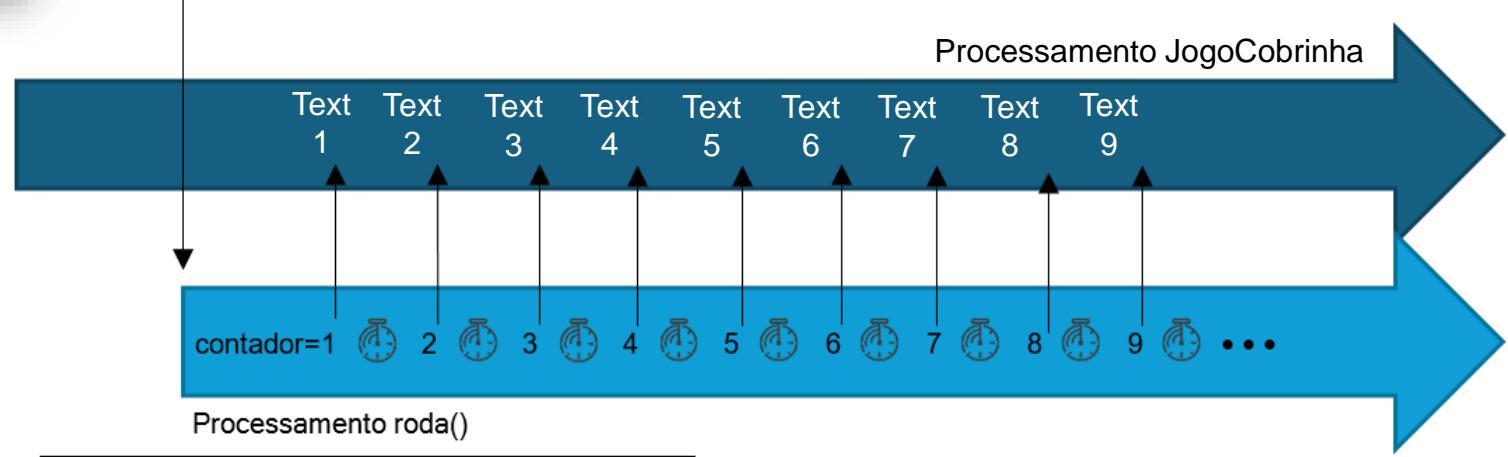
```
suspend fun roda () {  
    while(contador<1000) {  
        delay(1000)  
        contador++  
    }  
}
```

Executando o Emulador



```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = remember{Game()}
    var texto = ""

    // controle
    LaunchedEffect(game) {
        game.rola()
    }
}
```



```
class Game() {
    // variáveis
    var contador by mutableStateOf(0)
        private set
    // ação
    suspend fun roda() {
        while(contador<1000){
            delay(1000)
            contador++
        }
    }
}
```

Controlando a Execução com um Botão

- Vamos adicionar uma nova variável para controlar o jogo chamada jogoRodando. A ideia é que o contador só comece a funcionar quando essa variável for definida como true, assim iniciaremos com false.
- Envolver o `LaunchedEffect` com um `if` para pausar ou retomar a execução.

```
var jogoRodando by remember { mutableStateOf(false) }
```

```
if (jogoRodando) {  
    LaunchedEffect(game) {  
        game.roda()  
    }  
}
```

A qualquer momento
'false' o game.roda é
paulado

A qualquer momento
'true' o game.roda
retoma a execução

Adicionando um Botão e Ajustando a Interface no Jetpack Compose

- Adicionar um botão que altera o valor de jogoRodando entre true e false.

```
@Composable
fun JogoCobrinha() {
    // variáveis
    val game = remember{Game()}
    var jogoRodando by remember { mutableStateOf(false) }
    var texto = ""
    // controle
    if (jogoRodando) {
        LaunchedEffect(game) {
            game.roda()
        }
    }
    // tela
    texto="Jogo da Cobrinha ${game.contador} "
    Text(text =texto)
    Button(
        onClick = { jogoRodando = !jogoRodando },
        modifier = Modifier.fillMaxWidth(),
    ) {
        Text(if (jogoRodando) "Pause" else "Start")
    }
}
```

Ajustando a Interface

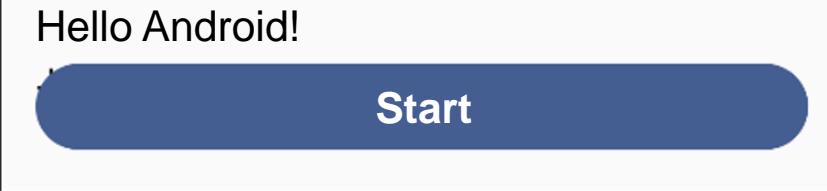
- Para garantir que o jogo se adapte a qualquer formato de tela dos dispositivos móveis, envolveremos a parte visual com BoxWithConstraints.

```
BoxWithConstraints (Modifier.padding(16.dp)) {  
    texto = "Jogo da Cobrinha ${game.contador}"  
    Text(text = texto)  
    Button(  
        onClick = { jogoRodando = !jogoRodando },  
        modifier = Modifier.fillMaxWidth(),  
    ) {  
        Text(if (jogoRodando) "Pause" else "Start")  
    }  
}
```

Ajustando a Interface

- Para corrigir o problema, utiliza-se da anotação `@SuppressLint("UnusedBoxWithConstraintsScope")` para suprimir um aviso do lint do Android Studio.`

GreetingPreview

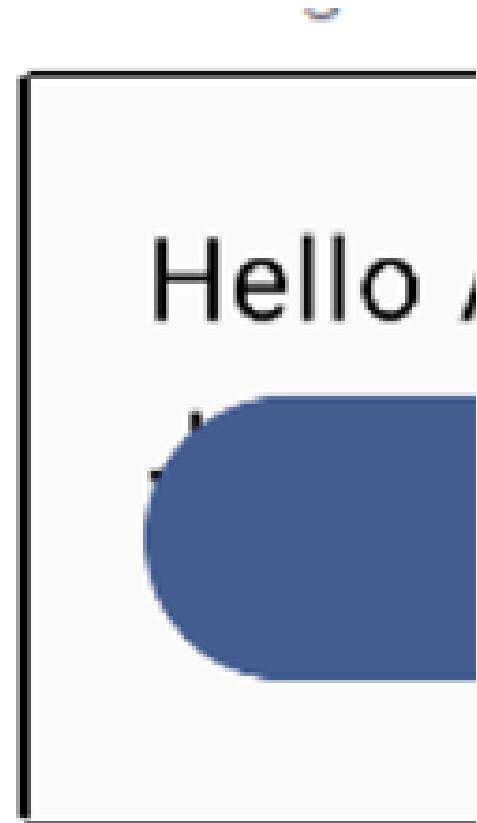


```
@SuppressLint("UnusedBoxWithConstraintsScope")
@Composable
fun JogoCobrinha() {
    // variáveis
    var jogoRodando by remember { mutableStateOf(false) }
    val game = remember{Game()}
    var texto = ""

    // controle
    if (jogoRodando) {
        LaunchedEffect(game) {
            game.roda()
        }
    }
    // tela
    BoxWithConstraints() {
        texto = "Jogo da Cobrinha ${game.contador} "
        Text(text = texto)
        Button(
            onClick = { jogoRodando = !jogoRodando },
            modifier = Modifier.fillMaxWidth(),
        ) {
            Text(if (jogoRodando) "Pause" else "Start")
        }
    }
}
```

Ajustando a Interface

- Se observamos direito, o Botão Start e o Texto “Jogo da Cobrinha” estão sobrepostos.
- Para corrigir, envolver os dois componentes com uma `Column`, organizando-os verticalmente, e já definindo que ela terá um fundo cinza claro, com os seus elementos posicionados a partir do topo, um após o outro, e centralizados horizontalmente.



Ajustando a Interface (organização vertical)

- Envolver com Column

```
BoxWithConstraints(){  
    Column(  
        modifier = Modifier.background(Color.LightGray),  
        verticalArrangement = Arrangement.Top,  
        horizontalAlignment = Alignment.CenterHorizontally  
    ) {  
        texto = "Jogo da Cobrinha ${game.contador} "  
        Text(text = texto)  
        Button(  
            onClick = { jogoRodando = !jogoRodando },  
            modifier = Modifier.fillMaxWidth(),  
        ) {  
            Text(if (jogoRodando) "Pause" else "Start")  
        }  
    }  
}
```

GreetingPreview

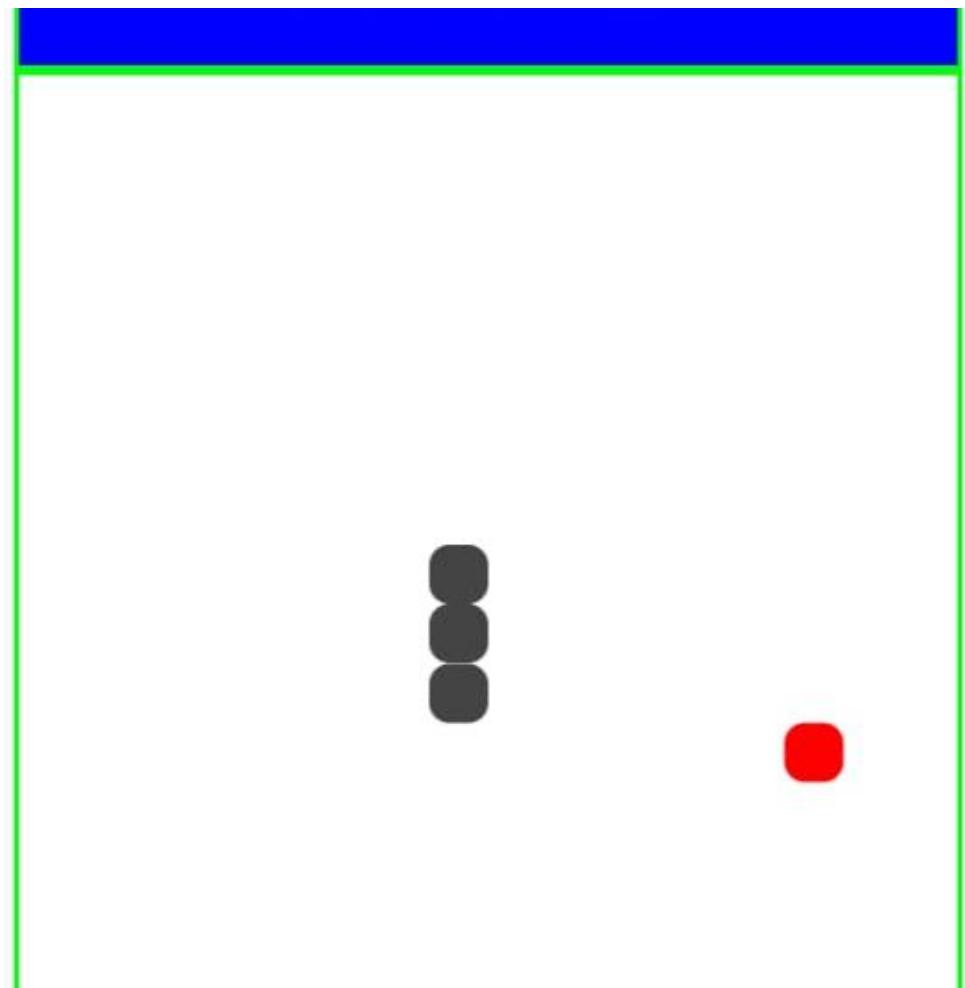
Hello Android!
Jogo da Cobrinha 0

Start

Desenhar o Objeto Principal

Criando o Tabuleiro do Jogo da Cobrinha

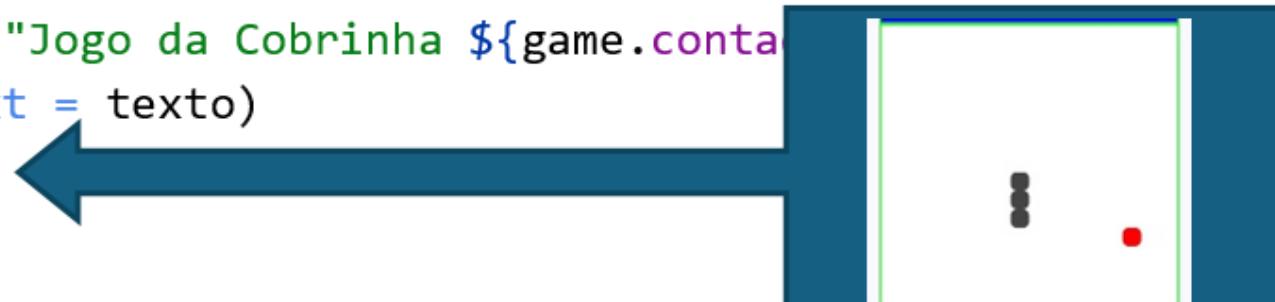
- Nosso jogo se passa em um tabuleiro de 16x16 quadradinhos.
- A cobrinha se movimenta e os alimentos aparecem nesse tabuleiro.



Inserindo o Quadro na Activity

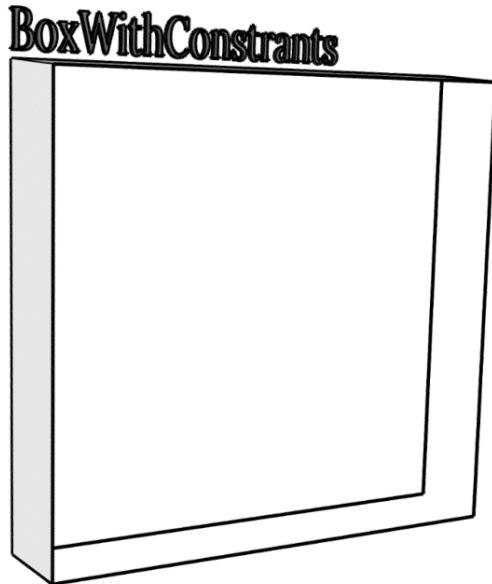
- Utilizando o Jetpack Compose, vamos inserir o quadro do jogo dentro da nossa activity.

```
BoxWithConstraints(){  
    Column() {  
        texto = "Jogo da Cobrinha ${game.conta}  
        Text(text = texto)  
        Button(  
            onClick = { jogoRodando = !jogoRodando },  
            modifier = Modifier.fillMaxWidth(),  
        ) {  
            Text(if (jogoRodando) "Pause" else "Start")  
        }  
    }  
}
```



Envolvendo o Quadro com BoxWithConstraints

- Na função JogoCobrinha, entre o Text e o Button, acrescente um BoxWithConstraints:

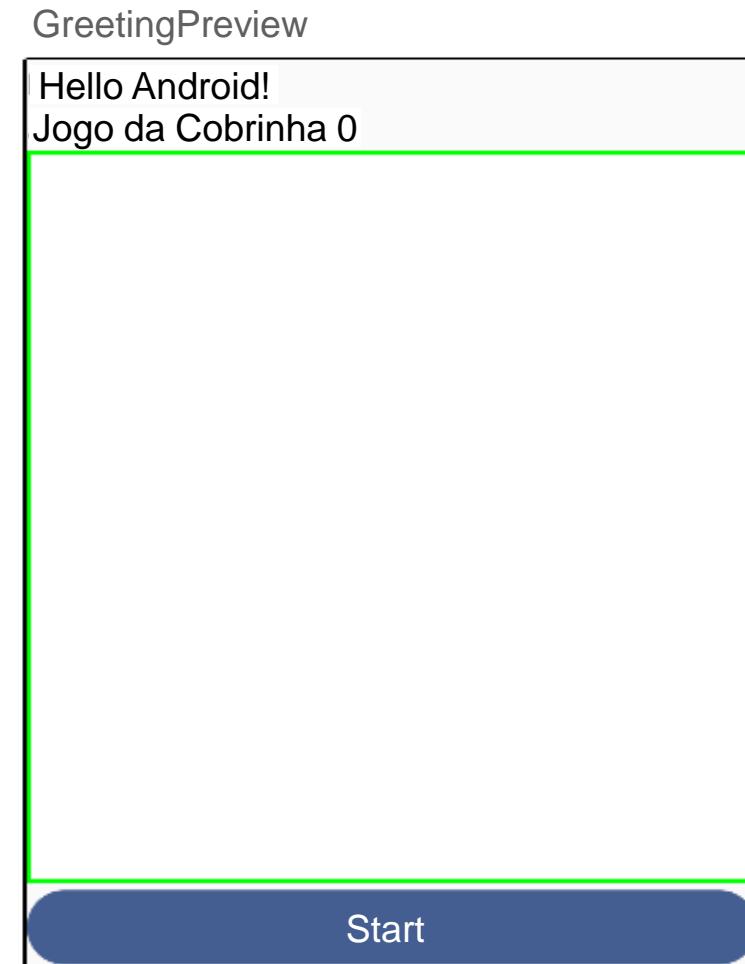
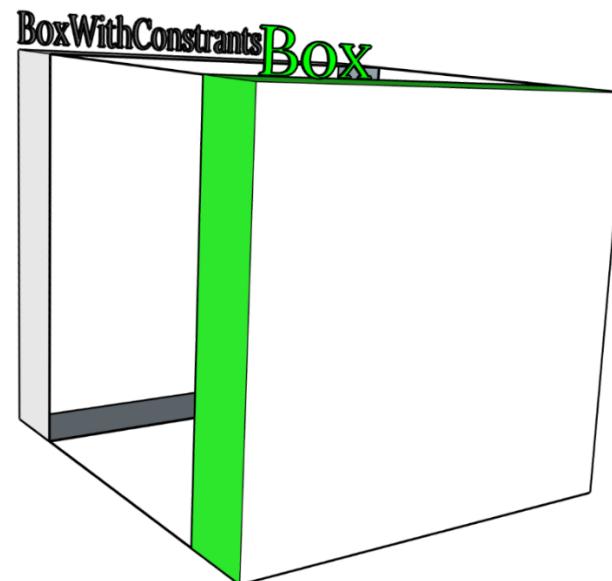


```
BoxWithConstraints() {
    Column(
        modifier = Modifier.background(Color.LightGray),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        texto = "Jogo da Cobrinha ${game.contador} "
        Text(text = texto)
BoxWithConstraints() {}
        Button(
            onClick = { jogoRodando = !jogoRodando },
            modifier = Modifier.fillMaxWidth(),
        ) {
            Text(if (jogoRodando) "Pause" else "Start")
        }
    }
}
```

Criando um Box Quadrado

- Dentro do BoxWithConstraints, insira um Box quadrado:

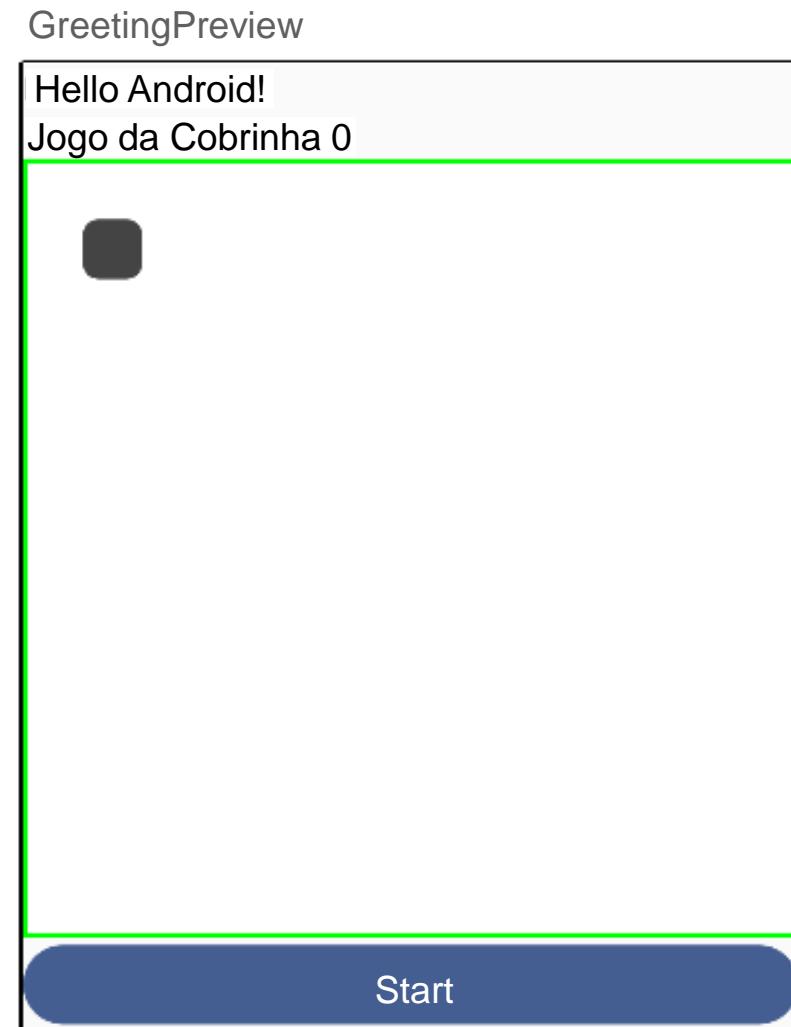
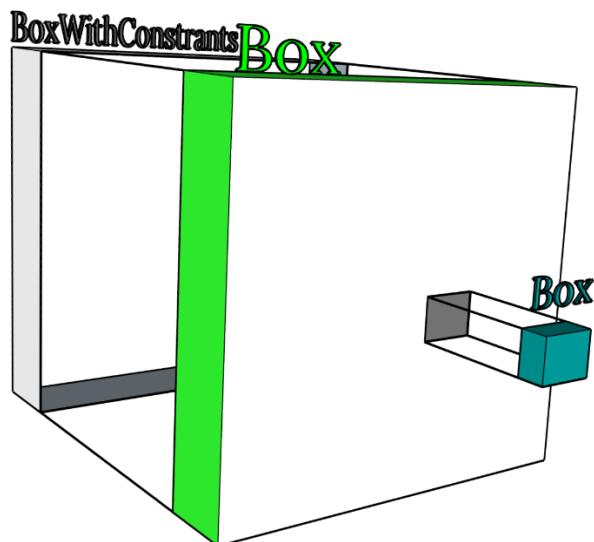
```
BoxWithConstraints(){  
    Box(Modifier  
        .size(maxWidth)  
        .background(Color.White)  
        .border(2.dp, Color.Green)){  
    }  
}
```



Adicionando a Cabeça da Cobrinha

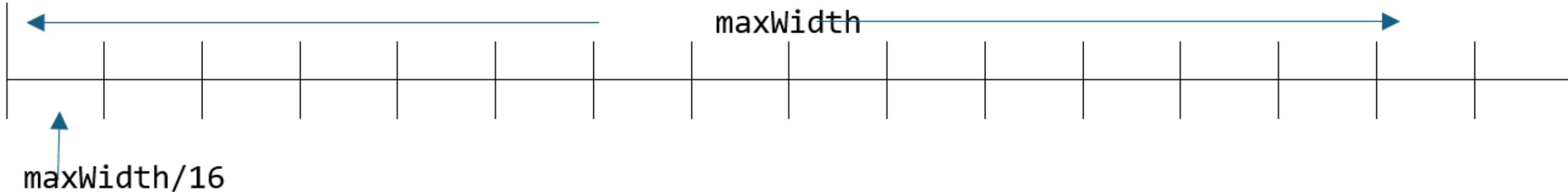
- Dentro do Box que forma o quadro do jogo, crie um Box quadrado para a cabeça da cobrinha

```
Box(  
    Modifier  
        .size(maxWidth)  
        .background(Color.White)  
        .border(2.dp, Color.Green)  
) {}  
  
Box(  
    modifier = Modifier  
        .offset(x = 30.dp, y = 30.dp)  
        .size(30.dp)  
        .background(  
            Color.DarkGray,  
            Shapes().small  
        )  
)
```



Definindo a Dimensão do Ponto

- Dividimos a largura em pixels do dispositivo por 16 para definir a dimensão padrão.
- Utilizamos o BoxWithConstraints para obter a propriedade maxWidth.



Criamos uma variável chamada `dimensaoPonto` que armazena 1/16 do valor da largura da tela do dispositivo:

```
// tela
BoxWithConstraints () {
    val dimensaoPonto = maxWidth/16
    Column () {
```

Implementação da animação

Implementação da Animação

- A tela está pronta para implementarmos a animação.
- A mecânica de mover a cabeça da cobrinha é similar aos princípios da Física.
- Utilizamos o modificador offset para posicionar o Box (cabeça da cobrinha) em coordenadas específicas.

Alterando o Offset do Box

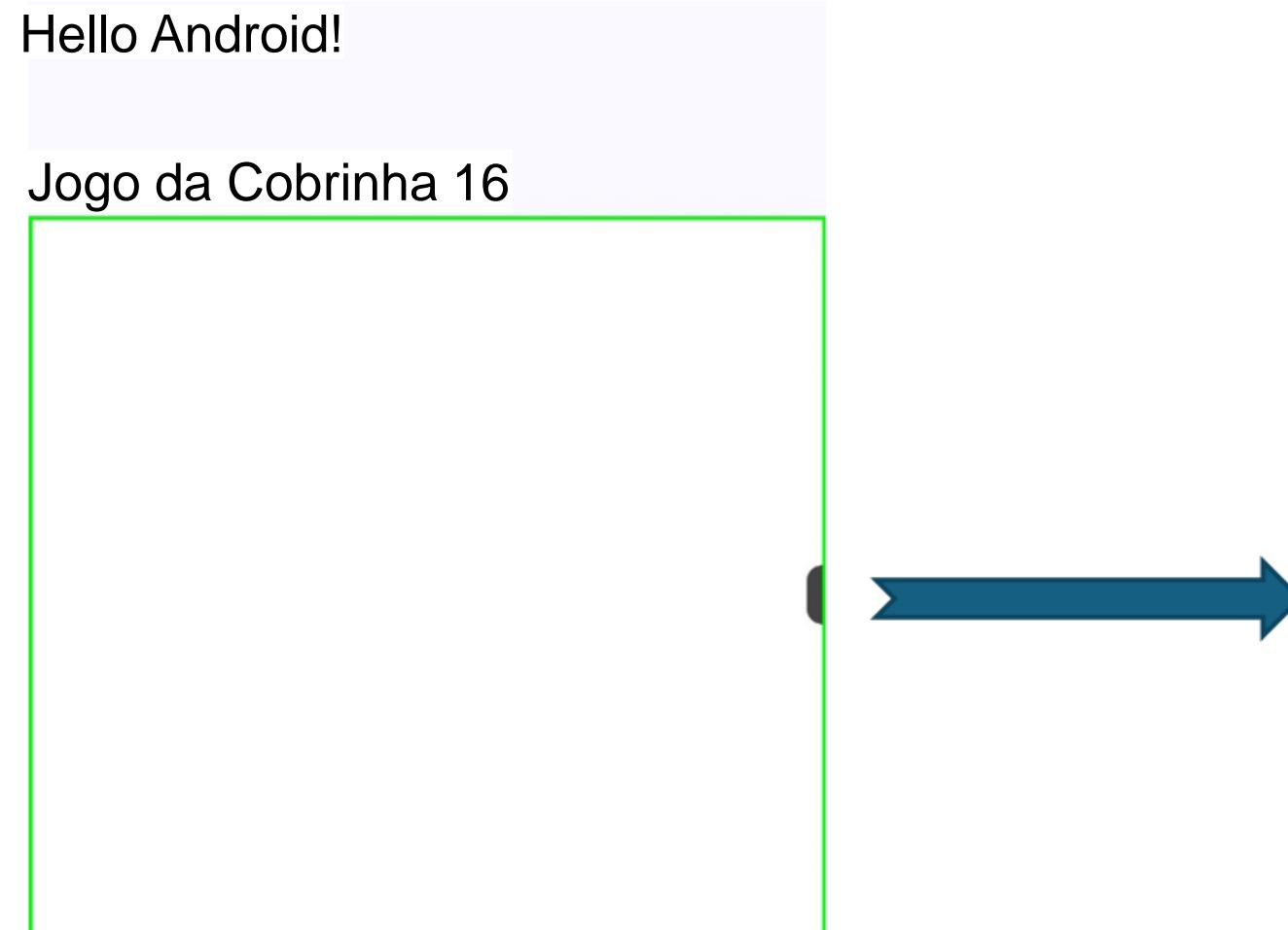
- Em nosso aplicativo, vamos alterar o offset para posicionar a posição de x e y do nosso Box cinza dentro do Box pai.

```
Box(  
    modifier = Modifier  
        .offset(x = game.contador.dp , y = 30.dp)  
        .size(30.dp)  
        .background(  
            Color.DarkGray,  
            Shapes().small  
        )  
)
```

Ajustando o Tamanho e Offset do Box

Ajustamos o tamanho do Box e alteramos o offset:

```
.offset(x = dimensaoPonto * game.contador , y = dimensaoPonto * 7)  
.size( dimensaoPonto )
```



Testando com Passo Negativo

- Testamos com o passo negativo para ver se o ponto surge na parede direita ao alcançar a parede esquerda:
contador--
- Neste ponto, vemos que o quadrado sai imediatamente da tela e vai desaparecendo para fora do quadro.

```
while(contador<1000) {  
    contador++  
    contador=(contador + 16) % 16  
    delay(500)  
}
```

Renomeando a Variável Contador

- Renomeamos a variável contador para posicaoX na lógica.
- Alteramos a condição para true a fim de garantir a continuidade da programação.

```
class Game(){  
    // variáveis  
    var posicaoX by mutableStateOf(0)  
    // ação  
    suspend fun roda(){  
        while(true){  
            posicaoX++  
            posicaoX=(posicaoX + 16) % 16  
            delay(500)  
        }  
    }  
}
```

Interatividade

Qual é o principal motivo para utilizar corrotinas no jogo Snake?

- a) Aumentar a velocidade de processamento do jogo.
- b) Reduzir o consumo de memória.
- c) Permitir que a interface do usuário seja atualizada enquanto o jogo está em execução.
- d) Simplificar a criação de animações complexas.
- e) Melhorar a compatibilidade com diferentes dispositivos.

Resposta

Qual é o principal motivo para utilizar corrotinas no jogo Snake?

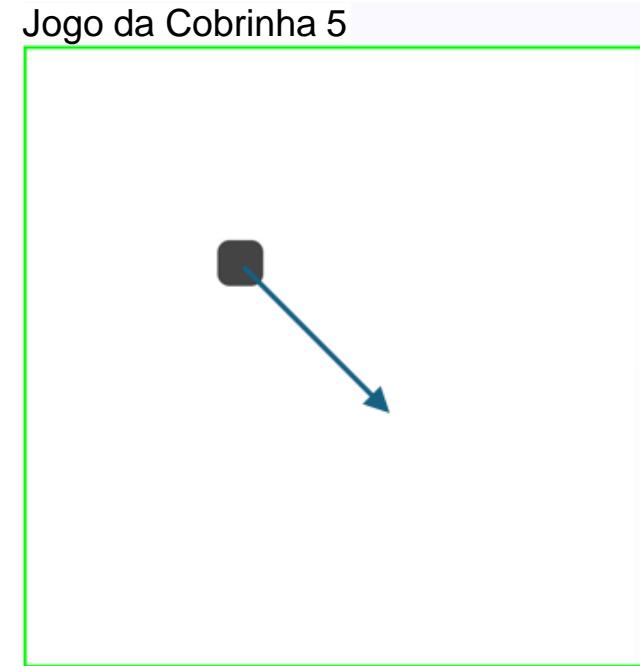
- a) Aumentar a velocidade de processamento do jogo.
- b) Reduzir o consumo de memória.
- c) Permitir que a interface do usuário seja atualizada enquanto o jogo está em execução.
- d) Simplificar a criação de animações complexas.
- e) Melhorar a compatibilidade com diferentes dispositivos.

Mover o objeto através do comando do usuário

Mover o Objeto através do Comando do Usuário

- Atualmente, estamos trabalhando apenas com a direção X, mantendo o valor de Y constante em 7.
- Para implementar a mudança de direção, precisaremos também considerar a direção Y.
- Na classe Game()

```
class Game(){  
    // variáveis  
    var posicaoX by mutableStateOf(0)  
    var posicaoY by mutableStateOf(7)  
    // ação  
    suspend fun roda(){  
        while(true){  
            posicaoX++  
            posicaoY++  
            posicaoX=(posicaoX + 16) % 16  
            posicaoY=(posicaoY + 16) % 16  
            delay(500)  
        }  
    }  
}
```



Utilizando a Classe Pair

- A classe Pair permite agrupar dois valores, como as coordenadas (x, y) da posição.
- Crie a variável direcaoAtual do tipo Pair na classe Game:

```
var direcaoAtual by mutableStateOf(Pair(1,0))
```

- Utilize os métodos first e second da classe Pair na função roda:
 - direcaoAtual.first
 - direcaoAtual.second

The diagram illustrates the refactoring of a function from using separate variables for position and direction to using a single Pair object. On the left, the original code uses `posicaoX` and `posicaoY` with update statements. On the right, the refactored code uses a `Pair` object `posicao` and updates its `first` and `second` components. Arrows show the mapping from the old variables to the new ones.

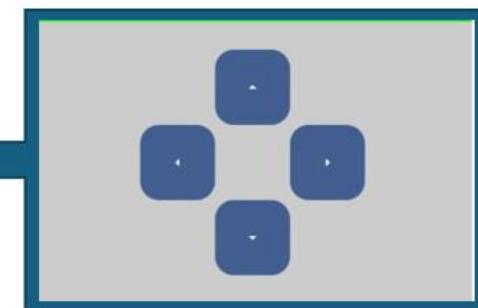
```
suspend fun roda(){
    while(true){
        posicaoX++
        posicaoY++
        posicaoX=(posicaoX + 16) % 16
        posicaoY=(posicaoY + 16) % 16
        delay(500)
    }
}
```

```
suspend fun roda(){
    while(true){
        posicaoX=posicaoX + direcaoAtual.first
        posicaoY=posicaoY + direcaoAtual.second
        posicaoX=(posicaoX + 16) % 16
        posicaoY=(posicaoY + 16) % 16
        delay(500)
    }
}
```

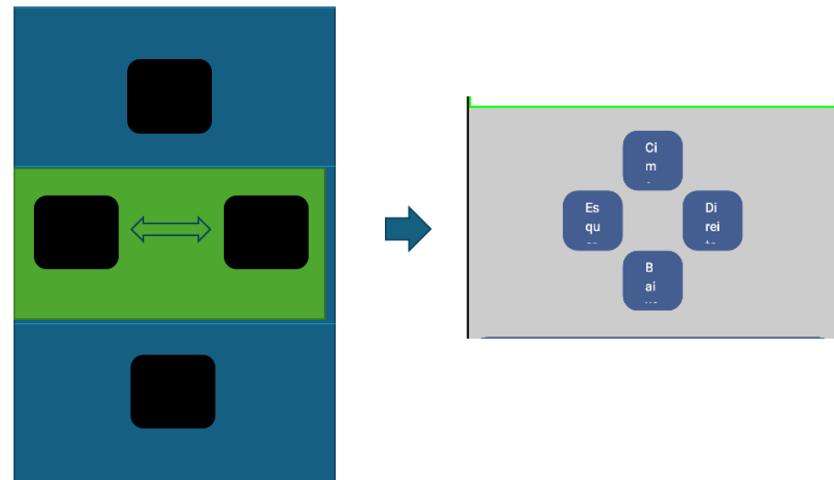
Implementando Botões de Direção

- Os botões estarão entre o quadro do jogo e o botão de Start/Pause.

```
// tela
BoxWithConstraints(){
    val dimensaoPonto = maxWidth/16
    Column() {
        texto = "Jogo da Cobrinha ${game.posicaoX} "
        Text(text = texto)
        //Quadro do jogo
        BoxWithConstraints(){
            Box(
                Modifier.(...))
            {
                Box(
                    modifier = Modifier.(...))
            }
        }
    }
    //Liga e desliga o jogo
    Button(
        onClick = { jogoRodando = !jogoRodando },
        modifier = Modifier.fillMaxWidth(),
    ) {...}
}
```



Adicionando os Botões de Direção



Adicione um botão abaixo da Row para definir a direção do movimento para baixo.

```
//botões de direção
Column() {
    Button(
        onClick = { game.direcaoAtual = Pair(0, -1) }) {
        Text("Cima")
    }
    Row{
        Button(
            onClick = { game.direcaoAtual = Pair(-1, 0) },)
            Text( "esquerda")
        }
        Spacer(modifier = tamBotao)
        Button(
            onClick = { game.direcaoAtual = Pair(1, 0) })
            Text( "direita")
        }
        Button(
            onClick = { game.direcaoAtual=Pair(0,1) })
            Text( "baixo")
    }
}
```

Criar obstáculos

Criar Obstáculos

- Vamos dar forma ao corpo da nossa cobra.
- Até agora, tínhamos apenas um ponto se movendo pela tela.
- Utilizaremos a coleção mutableListOf para criar o corpo da cobra.
- Isso permite adicionar, remover ou modificar elementos durante a execução do programa.

Vamos criar inicialmente uma cobra com três pontos:

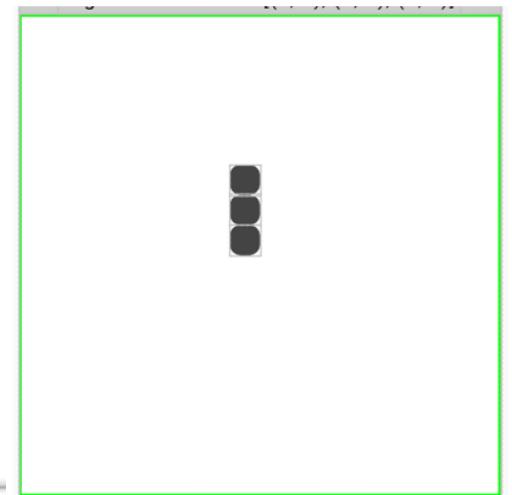
```
var cobra by mutableStateOf(  
    mutableListOf(Pair(7,7),Pair(7,6),Pair(7,5))  
)  
  
// variáveis  
var direcaoAtual by mutableStateOf(Pair(1,0))  
var cobra by mutableStateOf(  
    mutableListOf(Pair(7,7),Pair(7,6),Pair(7,5))  
)  
var posicaoX by mutableStateOf(cobra[0].first)  
var posicaoY by mutableStateOf(cobra[0].second)
```

Mostrando o Corpo da Cobra na Tela

- Variável que irá receber o corpo da cobra:
- Utilize o laço forEach para visualizar cada segmento do corpo da cobra:

```
// tela
BoxWithConstraints() {
    val dimensaoPonto = maxWidth/16
    val tamBotao = Modifier.size(64.dp)
    var corpoCobra = game.cobra
```

```
corpoCobra.forEach {
    corpo->Box(
        modifier = Modifier
            .offset(
                x = dimensaoPonto * corpo.first,
                y = dimensaoPonto * corpo.second
            )
            .size(30.dp)
            .background(
                Color.DarkGray,
                Shapes().small
            )
    )
}
```



Criando a Variável novaCabeca

- Crie uma variável auxiliar do tipo Pair para armazenar as coordenadas da nova posição da cabeça da cobra na classe Game:

```
var novaCabeca by mutableStateOf(Pair(posicaoX, posicaoY))
```

- Dentro do laço da função roda, atualize a variável novaCabeca, coloque a cabeça no começo da lista que representa o corpo da cobra e limite o tamanho.

```
// ação
suspend fun roda() {
    while(true) {
        posicaoX = posicaoX + direcaoAtual.first
        posicaoY = posicaoY + direcaoAtual.second
        posicaoX = (posicaoX + 16) % 16
        posicaoY = (posicaoY + 16) % 16
        novaCabeca = Pair(posicaoX, posicaoY)
        cobra.add(0, novaCabeca)
        cobra = cobra.take(3).toMutableList()
        delay(500)
    }
}
```

Tratamento da Comida

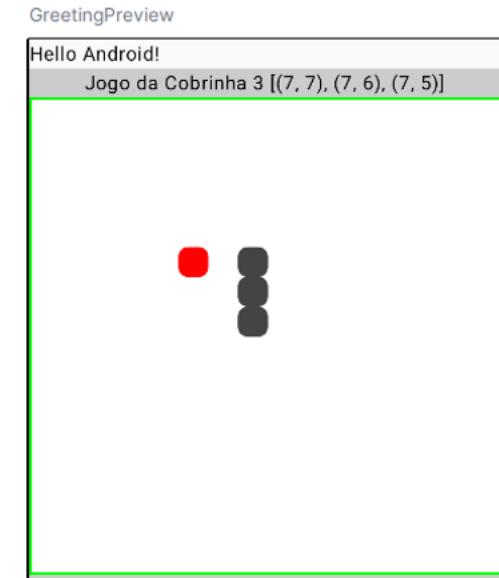
- Neste jogo, o desafio é comer as maçãs.
- A maçã representa uma recompensa e um desafio crescente.
- À medida que a cobra consome maçãs, ela aumenta de tamanho, dificultando os movimentos e aumentando a probabilidade de colisão com ela mesma.
- Criando a primeira maçã:
 - Ao criar a primeira maçã, ela ficará em uma posição fixa para facilitar a compreensão.
 - Em versões futuras, a maçã poderá ser colocada em uma posição aleatória.

```
class Game () {  
    // variáveis  
    var cobra by mutableStateOf(  
        mutableListOf(Pair(7,7), Pair(7,6), Pair(7,5))  
    )  
    var direcaoAtual by mutableStateOf(Pair(0,1))  
    var posicaoX by mutableStateOf(cobra[0].first)  
    var posicaoY by mutableStateOf(cobra[0].second)  
    var novaCabeca by mutableStateOf(Pair(posicaoX, posicaoY))  
var comida by mutableStateOf(Pair(5,5))
```

Representando a Comida na Tela

Coloque dentro do BoxWithConstraints do quadro um Box vermelho arredondado para representar a comida:

```
//Comida
Box(
    modifier = Modifier
        .offset(
            x = dimensaoPonto*comidaAtual.first,
            y = dimensaoPonto*comidaAtual.second)
        .size(dimensaoPonto)
        .background(Color.Red, Shapes().small))
{ }
```



Colisões do objeto com obstáculos

Tratamento da Colisão com a Comida

- Quando a cabeça da cobra entra em contato com a comida, duas ações são disparadas:
 - A cobra aumenta de tamanho.
 - Uma nova comida é gerada em uma posição aleatória na tela.
- Fixar o tamanho em 4 elementos não é flexível, pois a cobra pode comer a comida diversas vezes durante o jogo.
- Utilizaremos uma variável para armazenar o tamanho atual da cobra, permitindo que ela cresça de forma ilimitada.

Variável na Classe Game

Na classe Game, crie as variáveis tamanho:

```
class Game() {  
    // variáveis  
    var cobra by mutableStateOf(  
        mutableListOf(Pair(7,7),Pair(7,6),Pair(7,5))  
    )  
    var direcaoAtual by mutableStateOf(Pair(0,1))  
    var posicaoX by mutableStateOf(cobra[0].first)  
    var posicaoY by mutableStateOf(cobra[0].second)  
    var novaCabeca by mutableStateOf(Pair(posicaoX,posicaoY))  
    var comida by mutableStateOf(Pair(5,5))  
var tamanho by mutableStateOf(3)
```

Detectando a Colisão com a Comida

A colisão é detectada quando as coordenadas da cabeça da cobra coincidem com as coordenadas da comida:

```
if (novaCabeca == comida) {  
}  
}
```

- Quando esta condição for verdadeira, então, o tamanho deve aumentar.

```
if (novaCabeca==comida)  
    tamanho++  
}  
}
```

Detectando a Colisão

- Para sortear a nova posição, o Kotlin fornece método 'random' que seleciona um número pertencente a um intervalo gerado entre dois valores.
(n1..n2).random()
- Crie um novo Pair para compor a nova localização da comida:

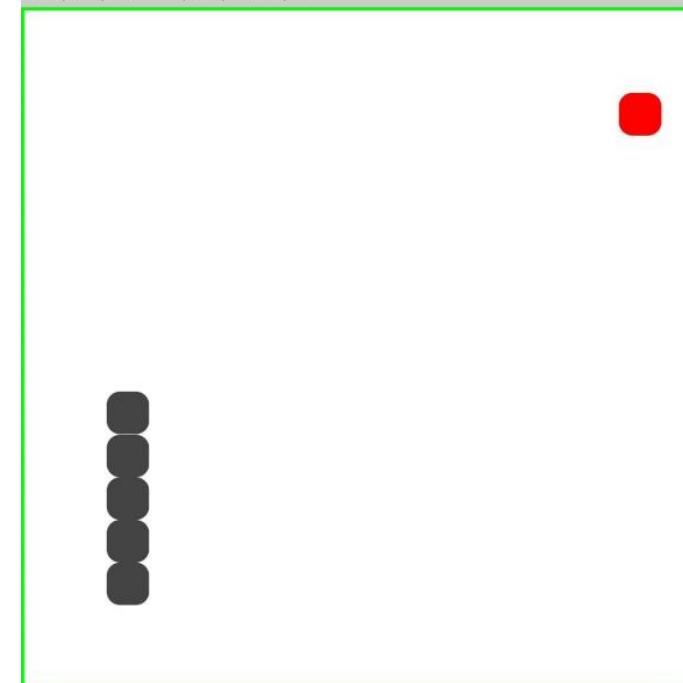
```
if(novaCabeca==comida) {  
    comida=Pair((1..16).random(), (1..16).random())  
    tamanho++  
}
```

Atualizar o Tamanho do Corpo da Cobra

Utilizar o método take para atualizar o tamanho da cobra:

```
if (novaCabeca==comida) {  
    comida=Pair((1..16).random(), (1..16).random())  
    tamanho++  
}  
cobra=cobra.take(tamanho).toMutableList()
```

Jogo da Cobrinha 5 [(2, 13), (2, 12), (2, 11), (2, 10), (2, 9)]



Tratamento da Colisão com o Próprio Corpo

Tratamento da Colisão com o Próprio Corpo

- A colisão com o próprio corpo deve resultar em um game over.
- Vamos criar uma variável auxiliar indicando que o jogo terminou.

```
// variáveis
var cobra by mutableStateOf(
    mutableListOf(Pair(7,7), Pair(7,6), Pair(7,5))
)
var direcaoAtual by mutableStateOf(Pair(0,1))
var posicaoX by mutableStateOf(cobra[0].first)
var posicaoY by mutableStateOf(cobra[0].second)
var novaCabeca by mutableStateOf(Pair(posicaoX, posicaoY))
var comida by mutableStateOf(Pair(5,5))
var tamanho by mutableStateOf(3)
var gameOver by mutableStateOf(false)
```

Detecção da Colisão com o Próprio Corpo

- A detecção de colisão é realizada verificando se a posição atual da cabeça da cobra está contida no conjunto de posições já ocupadas pelo corpo.
- Utiliza-se o método contains, que retorna verdadeiro caso um valor esteja presente em uma lista.

```
    novaCabeca = Pair(posicaoX, posicaoY)
    if(! (cobra.contains(novaCabeca))) {
        cobra.add(0, novaCabeca)
        if (novaCabeca == comida) {
            comida = Pair((1..16).random(), (1..16).random())
            tamanho++
        }
        cobra = cobra.take(tamanho).toMutableList()
    }
```

Reiniciar o Jogo

- Ao chamar essa função, também informaremos ao jogador que o jogo foi pausado.

```
if (! (cobra.contains(novaCabeca))) {  
    cobra.add(0, novaCabeca)  
    if (novaCabeca == comida) {  
        comida = Pair((1..16).random(), (1..16).random())  
        tamanho++  
    }  
    cobra = cobra.take(tamanho).toMutableList()  
} else {  
    gameOver=true  
    reset()  
}
```

Reiniciar o Jogo

- Criar uma função chamada fora da função roda que restaura todas as variáveis aos seus valores iniciais.

```
suspend fun roda(){...}  
//reset  
fun reset(){  
    tamанho = 3  
    cobra = mutableListOf(Pair(7,7),Pair(7,6),Pair(7,5))  
    direcaoAtual = Pair(0,1)  
    novaCabeca = Pair(7,7)  
    posicaoX = (novaCabeca.first)  
    posicaoY = (novaCabeca.second)  
}
```

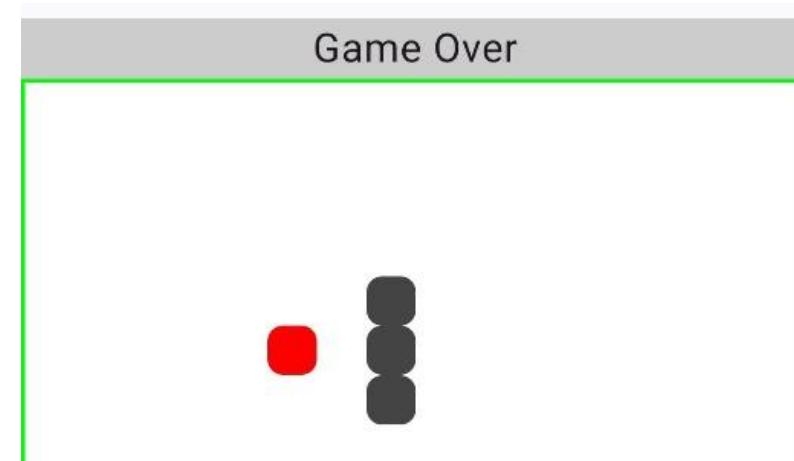
Indicar na nossa Activity que o jogo terminou

Vamos utilizar o espaço do texto superior onde estamos monitorando a lista com as coordenadas da cobra para mostrar que o jogo acabou:

```
// tela
BoxWithConstraints(){
    val dimensaoPonto = maxWidth/16
    val tamBotao = Modifier.size(64.dp)
    var corpoCobra = game.cobra
    var comidaAtual=game.comida
    Column(
        modifier = Modifier.background(Color.LightGray),
        verticalArrangement = Arrangement.Top,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        texto = "Jogo da Cobrinha ${corpoCobra.size} ${corpoCobra} "
        Text(text = texto)
    }
}
```

→

```
if (game.eOver) {
    texto = "Game Over"
    jogoRodando = false
} else {
    texto = "Jogo da Cobrinha ${corpoCobra.size} ${corpoCobra} "
}
Text(text = texto)
```



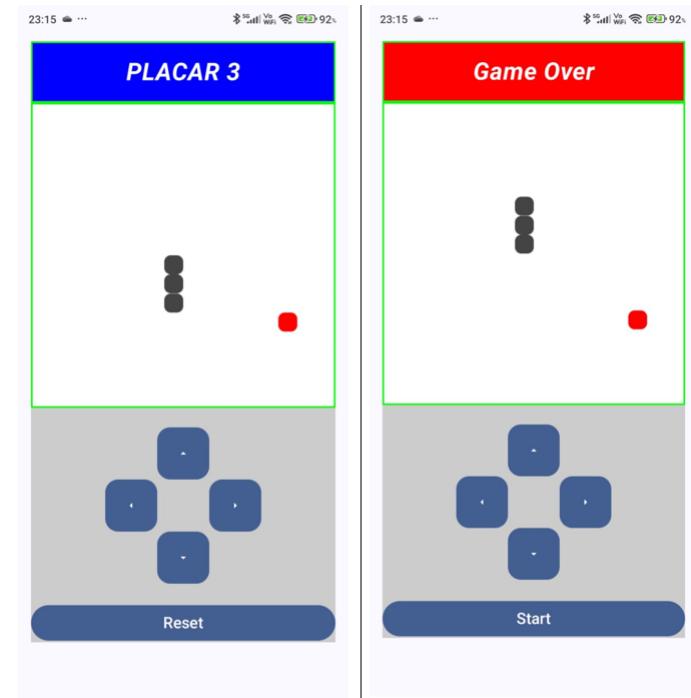
Adicionar placar

Definir o Box do Placar

- O box deve refletir as mudanças feitas no algoritmo.

```
Box(  
    Modifier  
        .fillMaxWidth() // Preenche a largura máxima disponível  
        .background(corFundo) // Define a cor de fundo  
        .border(2.dp, Color.Green) // Borda verde de 2dp  
        .padding(20.dp), // Margem de 20dp em todas as direções  
    contentAlignment = Alignment.Center // Centraliza o conteúdo dentro da caixa  
) {  
    Text(text = texto,  
        color = corTexto,  
        textAlign = TextAlign.Center,  
        fontSize = 30.sp,  
        fontWeight = FontWeight.Bold,  
        fontStyle = FontStyle.Italic)  
}
```

variáveis



Publicação de Aplicativos na Google Play Store

Objetivo do Processo

- Garantir a qualidade e a segurança dos aplicativos.
- Proporcionar uma experiência segura para os usuários.
- Conferir maior credibilidade aos aplicativos.

Gerando o Aplicativo Instalável (APK)

Tipos de APK:

- APK simples
- Signed APK (APK assinado)

Assinatura Digital:

- Garante autenticidade e integridade do aplicativo.
- Diretório META-INF verifica a assinatura e integridade.

Criação de APK:

- Menu Build → Build Bundles / APKs... → Create APK
- Menu Build → Generate Signed App Bundle / APK → Escolher entre APK e AAB

Android App Bundle (AAB):

- É um formato mais moderno.
- Ocupa menos espaço.
- Melhora o desempenho e a eficiência.
- Atualizações mais rápidas e econômicas.
- Requer conversão para APK antes da instalação.

Conta de Desenvolvedor

Conta de Desenvolvedor:

- Necessidade de conta de desenvolvedor.
- Taxa para criar a conta.

Anexar a Chave:

- Importância de guardar as informações da chave.
- Criar nova chave se necessário.

Versões do App:

- Chave privada digital para assinatura.
- Verificação de atualizações legítimas.

Preços e Distribuição:

- Modelos de monetização variados.
- Exploração de diferentes modelos de negócio.

Interatividade

Qual método do Kotlin é utilizado para selecionar um número pertencente a um intervalo gerado entre dois valores?

- a) range
- b) select
- c) random
- d) choose
- e) pick

Resposta

Qual método do Kotlin é utilizado para selecionar um número pertencente a um intervalo gerado entre dois valores?

- a) range
- b) select
- c) random
- d) choose
- e) pick

ATÉ A PRÓXIMA!