Estudo e experimentação do Android

Construção de um bloco de “rascunhos”

Projeto

- Projeto iniciado na aula anterior
  - Nome: Rascunhos
  - Versão: API 22

Vista para a área de desenho

- Incluir na atividade `AreaDesenhoActivity` um *layout* no qual iremos adicionar uma classe derivada da classe `View`
  - XML
    - `FrameLayout` com '*id*' `frAreaDesenho`
  - Kotlin
    - Incluir na atividade a funcionalidade de *View Binding*
    - Adicionar uma nova classe `AreaDesenho` derivada da classe `View`
    - No `onCreate` da atividade incluir

```
binding.frAreaDesenho.addView(AreaDesenho(this))
```

View e onDraw

- Criar um objeto `Paint` com as características de desenho pretendidas, por exemplo

```
val paint = Paint(Paint.DITHER_FLAG).also {
    it.color = Color.BLACK
    it.strokeWidth = 4.0f
    it.style = Paint.Style.FILL_AND_STROKE
}
```

- Fazer o *override* do método `onDraw` e experimentar com

```
canvas?.drawLine(50f,50f,150f,250f,paint)
```

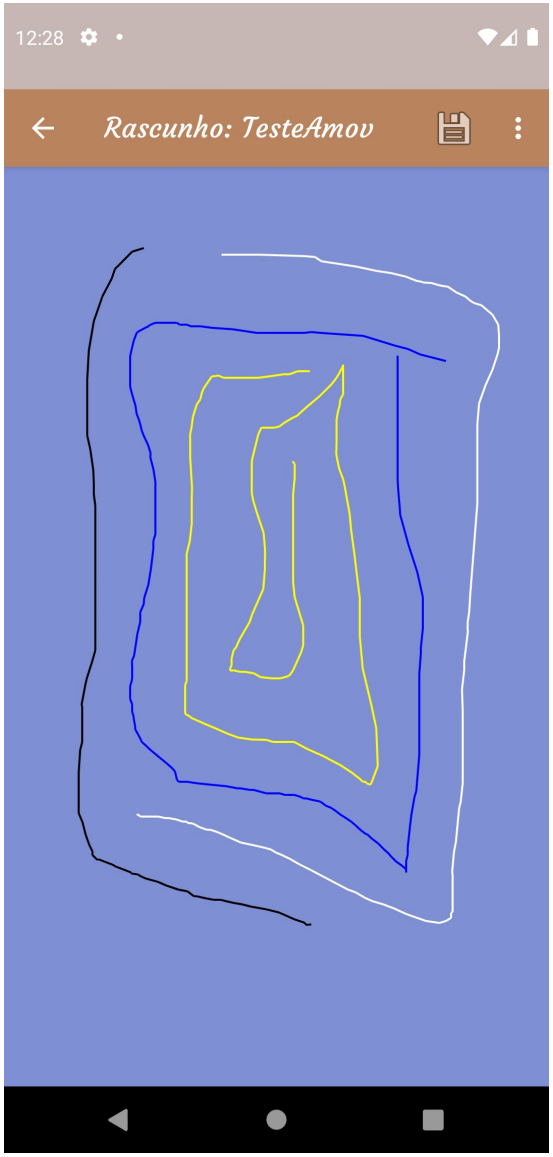
Deteção de gestos

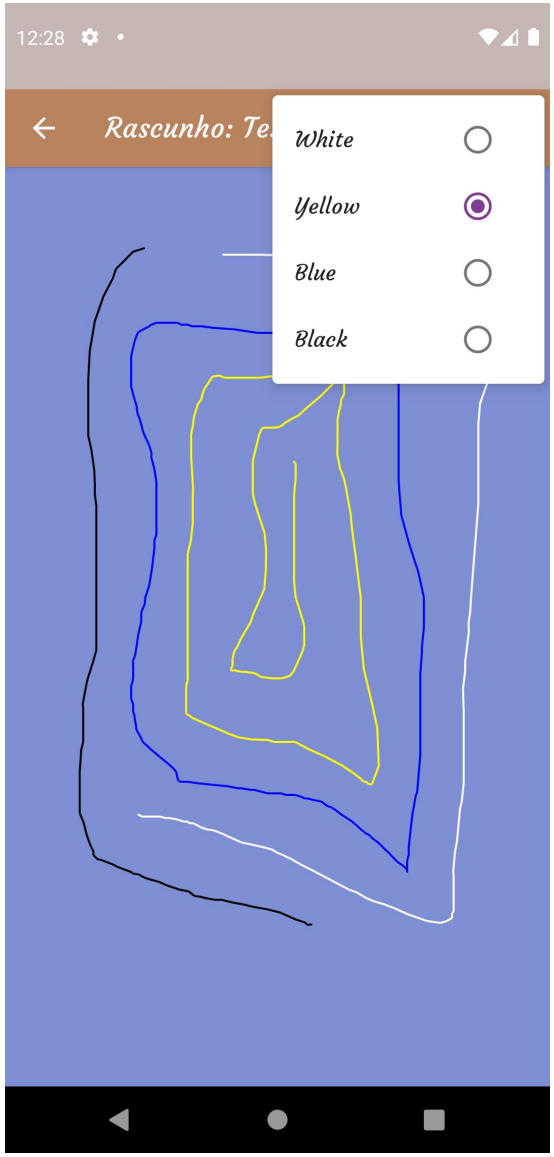
- Na classe `AreaDesenho`…
  - Implementar o interface `android.view.GestureDetector.OnGestureListener`
    - Processar todos os métodos abstratos deste interface, colocando uma linha de log em cada um (objetivo: perceber em que situações são chamadas)
    - **Incluir "`return true`" no método `onDown`**
  - Criar um objeto `GestureDetector` passando como primeiro argumento o contexto recebido e como segundo argumento a própria instância da classe `AreaDesenho`, ou seja, *this*
  - Processar o método `onTouchEvent` redirecionando o processamento do evento para o objeto `GestureDetector` criado

Rascunhos desenhados - Exercícios

- Objetivo
  - 1ª fase
    - Detetar os movimentos/gestos e desenhar linhas correspondentes a esses gestos
  - 2ª fase
    - Cada linha pode ter uma cor diferente
      - A cor pode ser sorteada ou escolhida com a ajuda de um menu

Rascunhos desenhados





Imagens

Fotos da galeria

Capturar foto

FundoImagemActivity

- Criar uma nova atividade comum para ambos os botões relativos utilização de imagens para o fundo do rascunho: `FundoImagemActivity`
- Definir constantes `ESCOLHER` e `CAPTURAR` a passar pelo `Intent` para indicar o tipo de imagem pretendida

```
companion object {
    val ESCOLHER = 1
    val CAPTURAR = 2
}
```

- Fazer variar o texto do botão de acordo com o tipo de fundo escolhido
- Apresentar uma pré-visualização da imagem escolhida

Acesso aos ficheiros de imagem

- Escolher imagem da galeria
  - Irá ser solicitada a uma aplicação externa a escolha de uma imagem disponível no dispositivo
  - O acesso à imagem será feito a partir do ficheiro indicado
- Capturar nova imagem (fotografia)
  - Irá ser solicitada a nova imagem a uma aplicação externa
  - Iremos passar referência para um ficheiro onde a imagem deverá ser armazenada

Acesso a ficheiros

- Para minimizar os acessos não autorizados à informação do utilizador, o sistema *Android* exige:
  - Declaração no ficheiro de manifesto da aplicação da necessidade de acesso a determinadas permissões
    - A partir da API 23 passou também a ser obrigatório pedir as permissões em *runtime*, com apresentação de uma janela com pedido de autorização explícito por parte do utilizador
  - Implementação de determinados protocolos para acesso a informações ou para dar permissão de acesso a informações internas

Permissões

- Incluir permissões no ficheiro de manifesto
  - `<uses-permission android:name="...." />`

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

  - Para este exercício não será necessário pedir permissões para utilizar a câmara uma vez que vamos recorrer à aplicação *Camera* do sistema
- Para sistemas com API23 ou posterior é necessário pedir as permissões em *runtime*

```
if (ContextCompat.checkSelfPermission(this,
android.Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED ||
    ContextCompat.checkSelfPermission(this,
android.Manifest.permission.WRITE_EXTERNAL_STORAGE)!= PackageManager.PERMISSION_GRANTED) {

    ActivityCompat.requestPermissions(this,
            arrayOf(android.Manifest.permission.READ_EXTERNAL_STORAGE,
                    android.Manifest.permission.WRITE_EXTERNAL_STORAGE), 1234)
}
```

Permissões

- Após o `requestPermissions`, quando o utilizador responde ao pedido de permissão, é chamado o método `onRequestPermissionsResult`

```
override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
) {
    super.onRequestPermissionsResult(
                    requestCode, permissions, grantResults)
    if (requestCode == 1234) {
        // ... (p.ex: verificar se as permissões foram dadas
    }
}
```

Selecionar imagem da galeria

- Pedir ao sistema que lance uma aplicação que permita escolher uma imagem

```
val intent = Intent(Intent.ACTION_PICK)
intent.setType("image/*")
startActivityForResult(intent, 10)
// 10 -> id do pedido startActivityForResult
```

Selecionar imagem da galeria

- Processar o método `onActivityResult` para receber o resultado

```
override fun onActivityResult(requestCode: Int, resultCode: Int, info: Intent?)
{
    if (requestCode == 10 && resultCode == Activity.RESULT_OK && info != null) {
        val uri = info.data?.apply {
            val cursor = contentResolver.query(this,
                    arrayOf(MediaStore.Images.ImageColumns.DATA), null, null, null)
            if (cursor !=null && cursor.moveToFirst())
                imagePath = cursor.getString(0)
        }
        return
    }
    super.onActivityResult(requestCode, resultCode, info)
}
```

- *Nota: a imagem é obtida através de um componente `ContentProvider`, consultado através de um objeto `ContentResolver`*
  - *Os Content Providers já foram introduzidos nas aulas teóricas e serão posteriormente aprofundados nas aulas teóricas*

registerForActivityResult

- Embora o método anterior, `startActivityForResult`, continue a ser válido, é recomendada a utilização do novo método introduzido com o *AndroidX* que permite uma gestão mais eficiente do resultado
- Para o exemplo anterior deve-se fazer:
  - Lançar a atividade:

```
val intent = Intent(Intent.ACTION_PICK)
intent.setType("image/*")
startActivityForResult.launch(intent)
```

  - Gerir o lançamento e processar o resultado:

```
var startActivityForResult = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult() ) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            val resultIntent = result.data
            val uri = resultIntent?.data?.apply {
                val cursor = contentResolver.query(this,
                    arrayOf(MediaStore.Images.ImageColumns.DATA), null, null, null)
                if (cursor !=null && cursor.moveToFirst())
                    imagePath = cursor.getString(0)
            }

        }
    }
```

Mostrar imagem

- Dependendo do objeto/situação...
  - `ImageView`
    - `ImageView.setImageBitmap`
  - `View`
    - `View.setBackground(Drawable)`
    - `[kotlin] View.background = <drawable>`
  - No âmbito do `onDraw`
    - `Canvas.drawBitmap`
  - ...

- Usar classes e funções adequadas para criar *bitmaps* e/ou *drawables*
  - `Bitmap, BitmapFactory`
  - `BitmapDrawable, bitmap.toDrawable(...)`
  - `getDrawable(...), ResourcesCompat.getDrawable(...)`

Mostrar imagem

- Há que ter cuidados com o espaço de memória necessário para processar as fotos
  - https://developer.android.com/topic/performance/graphics/load-bitmap
  - https://developer.android.com/training/camera/photobasics#TaskScalePhoto

```
fun setPic(view: View, path: String) {
    val targetW = view.width
    val targetH = view.height
    if (targetH < 1 || targetW < 1)
        return
    val bmpOptions = BitmapFactory.Options()
    bmpOptions.inJustDecodeBounds = true
    BitmapFactory.decodeFile(path, bmpOptions)
    val photoW = bmpOptions.outWidth
    val photoH = bmpOptions.outHeight
    val scale = max(1,min(photoW / targetW, photoH / targetH))
    bmpOptions.inSampleSize = scale
    bmpOptions.inJustDecodeBounds = false
    val bitmap = BitmapFactory.decodeFile(path, bmpOptions)
    when {
        view is ImageView -> (view as ImageView).setImageBitmap(bitmap)
        //else -> view.background = bitmap.toDrawable(view.resources)
        else -> view.background = BitmapDrawable(view.resources,bitmap)
    }
}
```

Tamanho do ecrã

- Para obter as dimensões do ecrã:

```
val windowManager = getSystemService(WINDOW_SERVICE) as WindowManager
val display = windowManager.defaultDisplay
val metrics = DisplayMetrics()
display.getMetrics(metrics)
var targetW = metrics.widthPixels
var targetH = metrics.heightPixels
```

- A *view* possui um método `onSizeChanged` que pode ser usado para ter acesso às dimensões
  - Quando este evento for recebido deve-se garantir que se faz o refrescamento dos elementos necessários

Tirar uma foto

- O modo mais fácil é recorrer a uma aplicação que já exista para esse fim
  - Lançar essa aplicação através de `startActivityForResult`

```
val folder = getExternalFilesDir(Environment.DIRECTORY_PICTURES)?.absolutePath
imagePath = folder +"/image.img"
val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE).apply {
    val fileUri = Uri.fromFile(File(imagePath))
    putExtra(MediaStore.EXTRA_OUTPUT, fileUri)
}
startActivityForResult(intent, 20)
```

  - Se o `resultCode` no método `onActivityResult` for de sucesso então a imagem está disponível para utilização
  - Mais informação e alternativas
    - http://developer.android.com/training/camera/photobasics.html
- O código apresentado poderá (deverá) ser adaptado para o novo modelo usado no *AndroidX* (`registerForActivityResult`)

Proteções em novas versões

- Para sistemas com API24 e posterior deixou de ser possível expor diretamente caminhos para ficheiros próprios
- Soluções:
  - Desligar a verificação de divulgação de ficheiros internos e o acesso controlado ao sistema de ficheiros da aplicação
    - Este método deve ser evitado e já deixou de ser suportado, por exemplo, na API 30
  - Usar os mecanismos/protocolos adequados: `FileProvider`
    - https://developer.android.com/training/camera/photobasics.html#TaskPath
    - https://developer.android.com/training/data-storage#scoped-storage

Proteções em novas versões

- Desligar a verificação de divulgação de ficheiros internos e o acesso controlado ao sistema de ficheiros da aplicação
  - No onCreate

```
if (Build.VERSION.SDK_INT >= 24) {
    try {
        val m: Method = StrictMode::class.java.getMethod("disableDeathOnFileUriExposure")
        m.invoke(null)
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
```

  - No ficheiro de manifesto

```
<application
    android:requestLegacyExternalStorage="true"
    …
```

FileProvider

- Para respeitar o protocolo de partilha de ficheiros da aplicação com outras aplicações deve-se usar os mecanismos do `FileProvider` para registar os locais e ficheiros a que outras aplicações podem ter acesso
  - Criar recurso *xml* com os detalhes dos recursos a que se está a dar acesso
  - Adicionar à aplicação (ficheiro de manifesto) o *provider* `androidx.core.content.FileProvider`
  - Obter um *URI* para o ficheiro com o auxílio do método `FileProvider.getUriForFile(…)`
    - Este URI será divulgado às aplicações que necessitam aceder aos recursos/ficheiros em causa

FileProvider

res/xml/file_paths.xml

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-files-path name="my_images" path="Pictures" />
</paths>
```

AndroidManifest.xml

```
...
<application ...>
    ...
    <provider
        android:name="androidx.core.content.FileProvider"
        android:authorities="pt.isec.amov.prepa4_a.android.fileprovider"
        android:exported="false" android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/file_paths" />
    </provider>
    ...
</application>
```

FileProvider

- Obtenção do URI seguro
  - Ex:

```
...
val folder = getExternalFilesDir(Environment.DIRECTORY_PICTURES)?.absolutePath
imagePath = folder +"/image.img"
fileUri = FileProvider.getUriForFile( context,
            "pt.isec.amov.prepa4_a.android.fileprovider", File(imagePath))
...
```

Histórico de rascunhos - Exercício

- Permitir consultar um histórico de rascunhos
  1. Guardar os rascunhos
     - Para simplificar poderá usar um `ArrayList` de objetos armazenado no objeto `Application` ou usar um "`object`" do *Kotlin*
     - Adicionar opções para gravar na atividade `AreaDesenho`
  2. Listar o histórico
  3. Permitir a consulta dos rascunhos
  4. Permitir a edição dos rascunhos

- Usar `ListView` para a lista de rascunhos
  - Ver exemplo das aulas teóricas