7. Reconnaissance de gestes communs

Reconnaître les gestes prédéfinis de *Android*.

Android Studio Flamingo Essentials, Kotlin Edition: chapitre 35

7.1. Introduction

1. Expliquez qu'*Android* est apte à reconnaître une série de gestes prédéfinis, dits *gestes communs* :

    a. Un « geste » débute dès qu'un premier doigt touche l'écran, et se termine lorsque le dernier doigt a quitté l'écran

    b. Gestes communs : touché (*tap*), double touchés (*double tap*), pression prolongée (*long press*), défilement horizontal ou vertical (*scroll*), balayage horizontal ou vertical (*flinch*)

2. *Android* facilite aussi de reconnaître des gestes plus complexes, tels que la rotation ou le pincement (*pinch*)

    a. Mais l'implantation de la gestion de ces gestes est plus complexe

    b. Soulignez que le chapitre 36 n'est pas couvert (manque de temps)

3. Enfin, *Android* nous permet de définir nos propres gestes (p.ex. reconnaître un cercle ou un carré tracé avec le doigt)

    a. C'est cependant assez ardu, et ça implique dessiner le geste sous forme d'image dans un fichier à intégrer dans l'application

    b. Encore une fois, soulignez que le chapitre 28 explique tout ça en détails, mais qu'on n'a pas le temps de couvrir cette matière

7.2. Reconnaissance des gestes communs

4. Expliquez que la détection d'un geste commun est une étape supplémentaire à la gestion d'un touché avec `OnTouchListener`

    a. Le `MotionEvent` reçu lors du `onTouch` peut être refilé à un objet de type **`GestureDetectorCompat`** qui va reconnaître un geste commun et exécuter une fonction membre surchargée selon le geste reconnu

5. Énumérez les étapes menant à la détection de gestes communs

    1) Rehausser la classe d'activité avec l'interface `GestureDetector.OnGestureListener` ainsi que les fonctions membres pertinentes parmi `onFling()`, `onDown()`, `onScroll()`, `onShowPress()`, `onSingleTapUp()` et `onLongPress()`

    2) Instancier la classe **`GestureDetectorCompat`** et l'associée à l'activité

    3) Si besoin est, invoquer la fonction membre **`setOnDoubleTapListener()`** de l'instance de `GestureDetectorCompat` pour gérer les touchés doubles

    4) Implanter un `OnTouchListener` sur le widget d'intérêt et, dans sa fonction `onTouch()`, invoquer à son tour la fonction `onTouch()` du `GestureDetectorCompat`

6. Expliquez qu'il est plus simple d'expliquer tout ça via un exemple

7. Créez un nouveau projet basé sur le gabarit « Empty Views Activity », nommé **`Gestes`** et consistant en un seul `TextView` centré dans l'activité et dont le `id` est `textView`

    a. Puisqu'on doit assigner un *listener* au `ConstraintLayout` de l'activité, attribuez-lui `mainLayout` comme `id`

b. Ne pas oublier d'activer la liaison de vues

i. Dans *app » Gradle Scripts » build.gradle.kts (Module:apps)* ajoutez :

```
buildFeatures {
    viewBinding = true
}
```

ii. Modifiez `MainActivity.kt` :

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding

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

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        ...
    }
```

et accédez aux widgets de l'activité via **`binding`**.

8. Indiquez que l'activité doit implanter les interfaces `GestureDetector.OnGestureListener` et `GestureDetector.OnDoubleTapListener`, et implantez dans la classe les diverses fonctions membres requises par ces deux interfaces

```
class MainActivity : AppCompatActivity(),
    GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {
    private lateinit var binding: ActivityMainBinding

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

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }

    override fun onDown(event: MotionEvent): Boolean {
        binding.textView.text = "onDown"
        return true
    }

    override fun onFling(event1: MotionEvent, event2: MotionEvent,
                         velocityX: Float, velocityY: Float): Boolean {
        binding.textView.text = "onFling"
        return true
    }
```

```
    override fun onLongPress(event: MotionEvent) {
        binding.textView.text = "onLongPress"
    }

    override fun onScroll(e1: MotionEvent, e2: MotionEvent,
                          distanceX: Float, distanceY: Float): Boolean {
        binding.textView.text = "onScroll"
        return true
    }

    override fun onShowPress(event: MotionEvent) {
        binding.textView.text = "onShowPress"
    }

    override fun onSingleTapUp(event: MotionEvent): Boolean {
        binding.textView.text = "onSingleTapUp"
        return true
    }

    override fun onDoubleTap(event: MotionEvent): Boolean {
        binding.textView.text = "onDoubleTap"
        return true
    }

    override fun onDoubleTapEvent(event: MotionEvent): Boolean {
        binding.textView.text = "onDoubleTapEvent"
        return true
    }

    override fun onSingleTapConfirmed(event: MotionEvent): Boolean {
        binding.textView.text = "onSingleTapConfirmed"
        return true
    }
}
```

9. Expliquez la subtile différence entre `onDoubleTap` et `onDoubleTapEvent` : lorsque le *layout* est touché deux fois de suite, les deux fonctions sont exécutées

   a. `onDoubleTap` est exécutée une seule fois après le second toucher

   b. `onDoubleTapEvent` est exécuté six fois : `ACTION_DOWN`, `ACTION_MOVE` puis `ACTION_UP`, répétés lors du second toucher

10. Enfin, intégrer un détecteur de gestes à la classe

```
class MainActivity : AppCompatActivity(),
    GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {
    private lateinit var binding: ActivityMainBinding
    private var détecteurGestes: GestureDetectorCompat? = null

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

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        this.détecteurGestes = GestureDetectorCompat(this, this)
        this.détecteurGestes?.setOnDoubleTapListener(this);
    }
    ...
```

11. Enfin, installez un *listener* sur le *layout* de l'activité afin de passer les `MotionEvent` au détecteur

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

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    this.détecteurGestes = GestureDetectorCompat(this, this)
    this.détecteurGestes?.setOnDoubleTapListener(this);

    binding.mainLayout.setOnTouchListener(object : View.OnTouchListener {
        override fun onTouch(v: View, ev: MotionEvent): Boolean {
            gérerOnTouchSurLayout(ev);
            return true
        }
    })
}

private fun gérerOnTouchSurLayout(geste: MotionEvent) {
    this.détecteurGestes?.onTouchEvent(geste);
}
...
```

    a. Expliquez que l'auteur propose une implantation alternative : il n'installe pas un `OnTouchListener` sur le *layout* de l'activité, mais supplante plutôt la fonction `onTouchEvent` dans la classe de l'activité ; nous utilisons ici un *listener* car nous sommes familiers avec cette façon de faire

    b. Nous avons besoin du `OnTouchListener` car nous l'utilisons pour permettre le déplacement de *droids*

    c. Cependant, notre `OnTouchListener` « gobe » les `MotionEvent` transmis à l'activité, donc le `GestureDetector` ne les voit pas

    d. Conséquemment, notre `OnTouchListener` doit « passer » au détecteur les `MotionEvent` qu'il reçoit

12. Exécutez l'application pour démontrer que ça fonctionne

13. Rappelez que le prochain chapitre (chapitre 36) couvre la reconnaissance de gestes plus sophistiqués, tels que la *rotation* et le *pincement*, ainsi que la programmation de nouveaux gestes

    a. Mais ces sujets ne sont pas matière au cours (pas le temps)

Évaluation formative 7.1 Indiquez aux étudiants qu'ils peuvent récupérer l'énoncé de l'évaluation (en format PDF) sur le portail éducatif du collège. Ils devraient pouvoir solutionner l'exercice sans aide en se référant au chapitre 35 du livre de référence