6. Gestion des touches et gestes

Réagir aux interactions de l'utilisateur avec les widgets via la gestion des gestes.

Android Studio Flamingo Essentials, Kotlin Edition: chapitre 34

6.1. Quelques définitions

1. Expliquez qu'en jargon *Android*, un toucher est généralisé à la notion de motion :

   a. Un « contact » de la main de l'utilisateur avec l'écran peut impliquer plus d'un toucher : chaque doigt représente un toucher, et l'ensemble des doigts d'un même contact représente une motion

   b. Dans une motion, chaque toucher est nommé un pointeur ; le nombre de pointeurs et leurs caractéristiques peuvent varier durant une même motion

   c. Une même motion génère une multitude d'évènements `onTouch()` : lorsque chaque pointeur est déposé sur l'écran, lorsque chaque pointeur est déplacé sur l'écran, et lorsque chaque pointeur est retiré de l'écran

   d. L'évènement `onTouch()` reçoit deux paramètres : la `View` impliquée, et le `MotionEvent` caractérisant la motion

2. Expliquez qu'il est plus simple de comprendre comment gérer ces évènements via un exemple

6.2. Gestion du `onTouch()`

3. Créez un nouveau projet basé sur le gabarit « Empty Views Activity », nommé **`Motions`** et y activer la liaison de vues

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

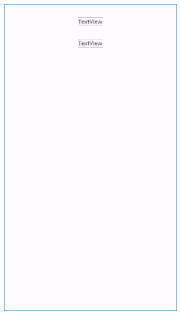
```
buildFeatures {
    viewBinding = true
}
```

b. 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`**.

4. Constituez l'activité comme suit (sans oubliez d'attribuer des contraintes aux widgets) :



   a. Déposez deux `TextView` au haut de l'activité, et attribuez-leurs l'identificateur `textView1` et `textView2`, respectivement

   b. Attribuez `mainLayout` comme identificateur au `ConstraintLayout` de l'activité

5. Soulignez qu'on désire effacer le contenu des `TextView` au démarrage afin qu'il n'affiche rien tant que l'utilisateur n'a pas effectué une opération

   a. Demandez aux étudiants de suggérer une façon d'accomplir cette tâche

   b. Plutôt que d'effacer le contenu des `TextView` dans `onCreate()`, déplacez le contenu de leur attribut `android:text` à l'attribut `tools:text` (celui avec une clé anglaise dans le panneau de propriétés)

   c. Exécutez l'application pour démontrer que les deux `TextView` sont vides

6. Attribuez un gestionnaire de l'évènement `onTouch()` au *layout* de l'activité

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

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

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

        // Installer un gestionnaire onTouch sur l'arrière-plan de l'activité
        binding.mainLayout.setOnTouchListener(object : View.OnTouchListener {
            override fun onTouch(v: View, ev: MotionEvent): Boolean {
                gérerÉvènementTouch(ev)
                return true
            }
        })
    }

    private fun gérerÉvènementTouch(geste: MotionEvent) {
        val nombrePointeurs = geste.pointerCount
        for (i in 0 until nombrePointeurs)
        {
            val x = geste.getX(i)
            val y = geste.getY(i)
            val id = geste.getPointerId(i)
            val action = geste.actionMasked
            val actionIndex = geste.actionIndex
            var actionString: String
            when (action)
            {
                MotionEvent.ACTION_DOWN -> actionString = "DOWN"
                MotionEvent.ACTION_UP -> actionString = "UP"
                MotionEvent.ACTION_POINTER_DOWN -> actionString = "PTR DOWN"
                MotionEvent.ACTION_POINTER_UP -> actionString = "PTR UP"
                MotionEvent.ACTION_MOVE -> actionString = "MOVE"
                else -> actionString = ""
            }
            val statutToucher =
                "Action: $actionString Index: $actionIndex ID: $id X: $x Y: $y"
            if (id == 0)
                binding.textView1.text = statutToucher
            else
                binding.textView2.text = statutToucher
        }
    }
}
```

7. Exécutez l'application pour démontrer son fonctionnement. Pour simuler l'utilisation de deux doigts dans l'émulateur, il faut utiliser le *pinch* à l'aide de la touche du clavier **Ctrl** (**Option** sur un Mac)

   a. Expliquez qu'idéalement, ce code devrait être testé sur un vrai appareil *Android*, où on peut facilement utiliser plusieurs doigts

Évaluation formative 06 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 34 du livre de référence

6.3. Rehausser la solution de l'exercice

8. Expliquez aux étudiants que nous allons maintenant rehausser les fonctionnalités de l'application développée dans le cadre de l'évaluation formative 06 afin de sélectionner un *droid* à déplacer parmi plusieurs *droids* affichés

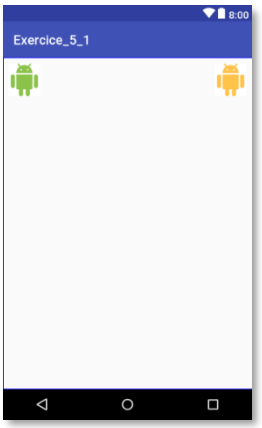
    a. Informez les étudiants qu'ils doivent uniquement vous écouter durant ce travail (et non pas le faire en même temps que vous)

    b. Les modifications apportées à la solution de l'évaluation formative 06 leur seront données lors d'un prochain exercice.

9. Modifiez le *layout* de l'activité comme suit :

    a. Ajouter un second `ImageView` (affichant le *droid*) dans le coin droit, et attribuez-lui des contraintes afin qu'il y soit positionné au démarrage

    b. Donnez une teinte rouge au second *droid* afin qu'on puisse visuellement le distinguer du premier, et renommez-le `id` de chacun en conséquence




```
<ImageView
    android:id="@+id/droidVert"
    android:layout_width="61dp"
    android:layout_height="63dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/android" />

<ImageView
    android:id="@+id/droidOrange"
    android:layout_width="61dp"
    android:layout_height="63dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:tintMode="add"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/android"
    app:tint="#ff0000" />
```

10. Expliquez que, comme dans la solution de l'évaluation sommative 06, on doit distinguer les types d'action dans l'évènement `onTouch()` :

    a. `ACTION_DOWN` : identifier le *droid* à la position du touché et s'en « souvenir »

    b. `ACTION_MOUVE` : déplacer le *droid* ayant été identifié lors du `ACTION_DOWN`

    c. `ACTION_UP` : « oublier » le *droid* ayant été déplacé afin de ne plus le faire

11. Demandez aux étudiants quelle stratégie peut être utilisée pour se « souvenir » du *droid* à manipuler lors des différentes étapes du mouvement

12. Définissez premièrement un attribut membre pour gérer le *droid* sélectionné

    a. Encouragez la participation des étudiants durant la programmation de cette application

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

    private var droidSélectionné: ImageView? = null        // indique le droid à déplacer

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

13. Définissez une fonction membre permettant de savoir si un point donné est sur un *droid* donné

```
// Fonction indiquant si la position donnée est sur le view donné
private fun positionSurView(v: View, x: Float, y: Float): Boolean {
    val droidOrigineX: Float = v.x
    val droidOrigineY: Float = v.y
    val droidOpposéX: Float = droidOrigineX + v.width
    val droidOpposéY: Float = droidOrigineY + v.height
    return (x >= droidOrigineX) && (x <= droidOpposéX) &&
           (y >= droidOrigineY) && (y <= droidOpposéY)
}
```

14. Définissez une autre fonction permettant de centrer un *droid* à une position donnée

```
// Repositionne le view donné afin qu'il soit centré sur la position donnée
private fun centrerViewSurPosition(v: View, x: Float, y: Float) {
    v.x = x - v.width / 2
    v.y = y - v.height / 2
}
```

15. Enfin, modifiez la fonction `gérerOnTouchSurLayout()` de l'évaluation formative 06

```
private fun gérerOnTouchSurLayout(geste: MotionEvent) {
    val x = geste.x
    val y = geste.y
    val action = geste.actionMasked

    when (action) {
        MotionEvent.ACTION_DOWN -> {
            if (positionSurView(binding.droidVert, x, y))
                droidSélectionné = binding.droidVert
            else if (positionSurView(binding.droidOrange, x, y))
                droidSélectionné = binding.droidOrange
            else
                droidSélectionné = null
        }

        MotionEvent.ACTION_UP -> {
            // On relâche le droid sélectionné (s'il y en avait un)
            droidSélectionné = null

            // On efface l'affichage de la position du toucher, puisque
            // l'utilisateur ne touche plus l'écran
            binding.coordonneesTextView.text = null
        }

        MotionEvent.ACTION_MOVE -> {
            // On doit travailler avec une variable locale car le droidSélectionné
            // peut être sujet à un conflit d'accès dû au multi tâche
            val sélection = droidSélectionné

            // Si le droid fut sélectionnée dans ACTION_DOWN, on le repositionne
            // à la position actuelle du toucher
            if (sélection != null) {
                centrerViewSurPosition(sélection, x, y)

                // Sans oublier d'afficher la position du toucher
                binding.coordonneesTextView.text = "X = " + x.toInt() + ", Y = " + y.toInt()
            }
        }

        else -> {}
    }
```

16. Si des fonction `onSaveInstanceState` et `onRestoreInstanceState` ont été supplantées lors de l'évaluation formative, les supprimer du projet afin d'éviter d'avoir à les mettre à jour

17. Exécutez l'application pour démontrer son fonctionnement

6.4. Reconnaissance de gestes

18. Expliquez que la gestion des évènements `MotionEvent` ne nous permet pas facilement d'identifier un mouvement particulier (p.ex. le *finch* ou le *pinch*)

19. L'API de *Android* nous offre cependant une infrastructure de détection des gestes les plus courants exploités dans les applications

   a. ***Scroll*** : pour le défilement d'une liste ou d'une image

   b. ***Tap*** : l'équivalant d'utiliser un bouton, mais sans le bouton

   c. ***Finch*** (nommée *swipe* dans *iOS*) : pour un changement d'activité ou de page

   d. ***Pinch*** : pour modifier le facteur d'agrandissement

   e. ***Rotate*** : pour tourner un widget à l'écran

20. De plus, *Android Studio* offre des outils nous permettant de définir et programmer nos propres gestes (c'est cependant assez complexe)

21. Expliquez que la reconnaissance de gestes exclue généralement la gestion des évènements `MotionEvent` via `onTouch()` : soit on gère nous-même les évènements `MotionEvent`, soit on exploite l'infrastructure de reconnaissance

   a. C'est possible de faire les deux dans une même activité, mais c'est compliqué

22. Expliquez que, contrairement à *Xcode*, *Android Studio* n'offre pas de widget facilitant l'exploitation des gestes dans une application : tout doit être fait par programmation

23. Enfin, expliquez que plusieurs widgets de *Android Studio* gèrent eux-mêmes la détection des gestes leur étant significatifs. Par exemple :

   a. Le `ScrollView` permet de défiler son contenu lorsque celui-ci est plus grand que la taille du `ScrollView`

   b. Le `MapView` permet défiler et modifier le facteur d'agrandissement de la carte

24. Conséquemment, par manque de temps nous n'allons pas étudier en détails la programmation de détection des gestes dans une application

   a. Les étudiants ne seront pas évalués sur l'implantation de reconnaissance de gestes

   b. Si nous avons besoin de reconnaître un geste particulier plus tard dans le trimestre, nous verrons comment le faire pour ce besoin particulier