Cours 9

Gestion d'événements de toucher

- Sur appareils mobiles, le dispositif principal d'interaction est le doigt de l'utilisateur.
- Celui-ci a beaucoup de différences avec une sourie:
  - il n'y a pas de pointeur à l'écran (généralement pas de "*hover*");
  - il n'y a pas plusieurs boutons permettant de faire des actions différentes;
  - il a tendance à cacher ce qu'il interagit avec;
  - la surface du toucher a très peu de précision.

- D’ailleurs, les mêmes actions peuvent être complétées de façon totalement opposée:
  - pour lire plus bas, on va pousser le doigt vers le haut pour déplacer le contenu, tandis que l’on aurait plutôt utilisé la roulette de la sourie vers le bas.
- Ceci est dû au fait que le doigt effectue une manipulation directe de l’objet à l’écran.
- Par exemple, pour tourner une page, on fait un “*swipe*” plutôt qu’appuyer sur un bouton “*next page*”.

- Non seulement l’utilisation de boutons visuels est plus rare, mais elle est souvent déconseillée puisque la surface d’affichage est limitée, et un bouton sur appareil mobile doit prendre suffisamment de place physiquement pour qu’il soit de la taille d’un doigt.
- On préfère utiliser des gestes et des menus contextuels.

- Les gestes ajoutent un autre niveau d'interaction:
  - un léger mouvement du doigt peut souvent indiquer une opération totalement différente;
  - l'absence de mouvement est également importante (par exemple, pour distinguer un "*long press*" d'un "*drag*");
  - on peut utiliser plusieurs doigts pour changer la signification d'un mouvement (un "*drag*" devient plutôt un "*pinch*" s'il est opposé par un autre "*drag*").

- Les gestes sont généralement gérés par la classe **`GestureDetector`** qui s'occupe de détecter tous les gestes communs d'une façon indépendante de l'application. Vous n'avez pas besoin, par exemple, de savoir combien de temps est nécessaire pour un "*long press*" et avec quelle vitesse un "*drag*" devient un "*swipe*".

- Pour ce cours, nous ne toucherons pas aux gestes, mais nous nous concentrerons plutôt sur les détails des touchers individuels.
- Un toucher est généralement représenté par 4 événements, peu importe la plateforme:
  - le début du toucher, lorsque le doigt touche l'écran;
  - le déplacement du toucher, tandis que le doigt reste en contact avec l'écran;
  - la fin du toucher, lorsque le doigt quitte l'écran;
  - et un événement d'annulation du toucher (par exemple, si une fenêtre apparaît par dessus un surface avec un toucher actif).

- Sur Android, il y a plusieurs façon de recevoir ces événements.
- Par exemple, on peut s'enregistrer pour recevoir ces événements sur une vue à l'aide d'un **`View.OnTouchListener`**, comme vous avez déjà fait avec **`View.OnClickListener`** et **`View.OnLongClickListener`**.

- Puisque nous apprenons présentement à faire des vues personnalisées, nous allons plutôt le faire en définissant la méthode **`onTouchEvent`** sur nos vues:

```
public class TouchableView extends View {

    public TouchableView(Context context) {
        this(context, null);
    }

    public TouchableView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        return super.onTouchEvent(event);
    }
}
```

- Contrairement aux autres méthodes que nous avons vues, l'implémentation de base de **`onTouchEvent`** de la classe **`View`** est importante lorsque le toucher n'a pas provoqué de réaction, et nous allons généralement toujours terminer notre implémentation en retournant son résultat.

- Normalement, l'activité elle-même reçoit initialement l'événement, puis le passe à sa vue principale, qui le propage à ses enfants, jusqu'à ce qu'une vue retourne **`true`** pour signaler que l'événement lui était destiné. Par exemple, si votre vue ne remplit pas entièrement l'espace qui lui est alloué, il est possible que d'autres vues soient visible derrière la vôtre, alors vous devriez uniquement retourner **`true`** si vous avez confirmé que le toucher vous appartient.

- L'objet **`MotionEvent`** reçu contient toute les informations de l'événement actuel ainsi que les informations de tout autre toucher actif.
- On détermine quel événement est reçu grâce à la propriété "*action*" de l'événement.
- Celle-ci contient également l'indice du toucher ayant lancé l'événement, alors on utilise généralement la méthode **`getActionMasked()`** pour obtenir sa valeur.

```
switch (event.getActionMasked()) {


}
```

- Nous avons vu qu'il y a généralement 4 événements de toucher, qui sont retournés sur Android en tant que les actions suivantes:
  - **`ACTION_DOWN`** - lorsque le doigt touche l'écran;
  - **`ACTION_MOVE`** - lorsque le doigt se déplace sur l'écran;
  - **`ACTION_UP`** - lorsque le doigt quitte l'écran;
  - **`ACTION_CANCEL`** - lorsque le toucher se fait annuler.

- Pour raisons historiques, Android ajoute deux autres actions pour gérer le "*multi-touch*":
  - **`ACTION_POINTER_DOWN`** - un doigt additionnel vient de toucher l'écran;
  - **`ACTION_POINTER_UP`** - un doigt vient de quitter l' écran en y laissant un ou plusieurs autres doigts.
- Nous devons donc corriger la définition de deux actions précédentes:
  - **`ACTION_DOWN`** - lorsque le premier doigt touche l' écran;
  - **`ACTION_UP`** - lorsque le dernier doigt quitte l'écran.

- Faites attention que la dénomination de "*pointer*" dans les actions précédentes de veut pas dire qu'un toucher est considéré un pointeur uniquement lorsqu'il y a plusieurs doigts impliqués. Tous les touchers sont des pointeurs, même s'ils viennent d'une action **`ACTION_DOWN`**. Ces actions ne représentent que l'ordre des touchers, et il est possible qu'un **`ACTION_DOWN`** se termine avec **`ACTION_POINTER_UP`**.
- Vous remarquerez aussi qu'il n'y a pas de distinction pour les événements **`ACTION_MOVE`** et **`ACTION_CANCEL`**.

- Si vous n’avez pas besoin de faire la distinction, vous pouvez traiter les événements de “multi-touch” avec les événements réguliers:

```
switch (event.getActionMasked()) {

    case MotionEvent.ACTION_DOWN:
    case MotionEvent.ACTION_POINTER_DOWN:
        // début de toucher
        break;

    case MotionEvent.ACTION_MOVE:
        // déplacement
        break;

    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
        // fin de toucher
        break;

    case MotionEvent.ACTION_CANCEL:
        // annulation
        break;
}
```

- Chaque événement contiendra un tableau avec la liste de tous les touchers actifs, et un indice indiquant lequel a provoqué l'événement.
- Vous pouvez obtenir cet indice à l'aide de **`getActionIndex()`**.
- Faites attention! L'indice n'est valide que pour la position dans le tableau de l' événement actuel. Si vous voulez identifier un toucher d'un événement à l'autre, vous devez obtenir son identifiant à l'aide de **`getPointerId()`**.

```
int pointerIndex = event.getActionIndex();
int pointerId = event.getPointerId(pointerIndex);
```

- Vous pouvez maintenant obtenir la position en X et Y du toucher à l'aide de getX et getY.

```
float x = event.getX(pointerIndex);
float y = event.getY(pointerIndex);
```

- Vous remarquerez que ces valeurs sont des **`float`** puisqu'elles sont des approximations selon la taille et la pression du doigt.

- Lorsque plusieurs actions de mouvements sont enchaînées rapidement, Android les regroupe en une seule action. Dans ce cas, les positions obtenus de X et Y sont les positions les plus récentes lorsque l'on n'a pas besoin de savoir la trajectoire.
- Si on a besoin de savoir tous les points de la trajectoire depuis la dernière action, on peut utiliser:

```
for (int i = 0; i < event.getHistorySize(); ++i) {

    float x = event.getHistoricalX(pointerIndex, i);
    float y = event.getHistoricalY(pointerIndex, i);
}
```

- Attention! Généralement vous obtiendrez toujours tous les événements de vos touchers, mais puisque c’est la responsabilité de la vue parent de vous distribuer les événements, il peut y avoir des situations particulières que vous n’obtiendrez pas toutes les informations.