Multimodalité

d'après un cours de Laurence Nigay

Philippe.Renevier@unice.fr

http://atelierihm.unice.fr/enseignements/techniques-interaction/

D'après un cours de Laurence Nigay

1 + 1 = ?

- Deux informations simultanées
  - en forment une troisième
  - s'influencent elles-mêmes et se combinent
- *"L'effet McGurk a été mise en évidence pour la première fois en 1976 par McGurk et MacDonald. L'effet consiste en une illusion auditive qui résulte de la combinaison d'un stimulus visuel et d'un stimulus auditif discordants. Le stimulus visuel présente un visage prononcant une syllabe, et le stimulus autitif une autre syllabe. L'illusion dépend du choix des stimuli.*
  Par exemple, un 'bi' auditif combiné avec un 'gi' visuel est perçu par cetaines personnes comme 'di'."
  Source : http://www.ulb.ac.be/philo/phonolab/mcgurk.html
- *Vidéo :* http://www.koreus.com/video/effet-mcgurk.html

Fondation : put that there

- Bolt, Siggraph'80
- Vidéo de démo (CHI 84) : http://www.youtube.com/watch?v=0Pr2KIPQOKE
- Traitement automatique du Langage (Naturel)
- Interface gestuelle

- A marqué les esprits

Définition d'une modalité [Nigay 94]

- Modalité = <dispositif , langage>
- Dispositif = ce qui fourni l'information
  - Un capteur, un service, etc.
- Langage= façon d'interprété l'information
- Relation modalité – interaction – tâche

- Exemples :
  - <micro, langage naturel>
  - <clavier, langage naturel>
  - <clavier, pseudo langage> (ex: ligne de commande)
  - <clavier, direction et déplacement> (ex: jeu)
  - <wiimote, direction et déplacement>
  - Etc.

Composition de modalité

- Problème de fusion de donnée
  - Choix des seuils, etc.
  - Choix de la confiance de l'information
- Relations entre les modalités, propriétés CARE [Nigay 94]
  - Equivalence (l'un ou l'autre)
  - Assignation (à une tâche, à un langage)
  - Complémentarité (les deux)
  - Redondance (l'un et l'autre)

Composition de modalités [Vernier 01, d'après Allen]

Schémas de composition

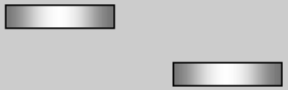
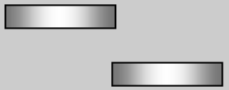
Aspects de composition \ Composition					
Temporelle	Anachronique	Séquentielle	Concomitante	Coïncidente	Parallèle / Simultanée
Spatiale	Disjointe	Adjacente	Intersectée	Imbriquée	Recouvrance
Articulatoire	Indépendance	Fissionnée	Fissionnée + Dupliquée	Partiellement Dupliquée	Dupliquée
Syntaxique	Différente	Complétion	Divergence	Extension	Jumelage
Sémantique	Concurrente	Complé-mentaire	Complémentaire + Redondante	Partiellement Redondante	Totalement Redondante

Tableau 1 : Application des schémas de composition aux cinq aspects proposés.

Enjeu de conception

- La bonne modalité au bon moment
  - La bonne composition
  - Avec l'environnement
- La bonne modalité pour le bon utilisateur

- De nombreux choix à faire ou à inventer

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Ce n'est pas parce qu'une interface est multimodale que les utilisateurs vont utiliser la multimodalité.

Dans QuickSet la multimodalité est utilisée dans 20% du temps d'une session de travail. Les utilisateurs passent d'un mode à l'autre sans raison apparente et restent unimodaux certainement pour des raisons de confort personnel. Cependant les commandes spatiales sont plus fréquemment multimodales ainsi que les informations de taille, de forme des objets, de nombres, de lieux et d'orientations. La richesse sémantique de l'action favorise la multimodalité.

1

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Le pattern parole-pointage n'est pas le plus intéressant.

Depuis le fameux « mets ça là » de Bolt, la multimodalité a été centrée sur le paradigme de l'interaction synergique. Dans ce paradigme la parole est considérée comme mode sémantique dominant et le geste de désignation comme subordonné. En fait cette conception est une survivance du concept clavier/souris (c'est-à-dire de sélection sur une icône ou un menu), bien plus pauvre qu'une interaction qui utiliserait les mouvements gestuels, les expressions faciales ou corporelles, etc. Par exemple des études avec un stylo/voix montrent que la multimodalité est de 14% plus utilisée qu'avec une entrée souris/voix. L'utilisation des déictiques est aussi plus fréquent de 20%.

2

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

La multimodalité ne signifie pas obligatoirement « parallélisme ».

En effet on a constaté que bien souvent le geste précède la parole (99% des cas), même lorsque les deux modes dénotent des informations synchrones comme les déictiques. Le degré d'anticipation dépend de la langue. Il n'y a finalement que 25% des énoncés qui sont véritablement simultanés : synchronie ne signifie pas simultanéité.

3

Figure 1: Quickset multimodal pen/voice system on a hand-held PC

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

La parole n'est pas un mode « de base » dans un système multimodal.

Cela n'est vrai que sur le plan historique. Depuis il y a bien des systèmes qui utilisent la main et le regard par exemple comme modes d'entrée, notamment dans les systèmes militaires. Le problème général de la multimodalité ne se pose donc pas en termes de *commande+sélection,* la commande étant linguistique et la sélection manuelle. Le problème ne se pose pas non plus en terme de *source principale/source secondaire* dans lequel on utiliserait la source secondaire dans le cas où la source principale serait dégradée.

4

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Le langage multimodal ne diffère pas du langage unimodal.

On peut dire seulement que le langage utilisé en contexte multimodal est syntaxiquement moins complexe, que les énoncés sont plus courts et que le débit est moins hésitant. Les ellipses sont plus fréquentes et les constructions linguistiques sont moins ambiguës, car les énoncés sont plus compacts. Il semble que ces propriétés rendent le langage multimodal plus apte à une intégration dans un système homme-machine.

5

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

L'interaction multimodale ne favorise pas la redondance.

On pourrait croire le contraire, mais cela ne va pas dans un sens d'économie du point de vue de l'utilisateur. Celui-ci va donc privilégier la complémentarité. Même dans le cas d'échec puis d'essais de correction, l'usage de la redondance n'augmente pas de façon significative. La redondance n'est pratiquement utilisée que dans le sens d'une recherche de fiabilité

6

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Les erreurs sur un mode ne sont pas compensées par un autre mode.

Il est illusoire de penser que l'on va masquer les insuffisance d'un mode (par exemple les erreurs de reconnaissance de la parole) par un autre mode. En réalité les erreurs se cumulent d'un mode à l'autre. Mais les utilisateurs optimisent l'usage d'un mode au profit de tel autre, après expérience faite de ses performances, ce qui rend somme toute, par effet indirect de l'usage, l'interaction plus robuste. Dans quelques cas cependant, lorsque une double incertitude se produit dans les deux modes d'entrée, il est parfois possible de recouper l'information sur un critère de cohérence sémantique.

7

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Les utilisateurs n'organisent pas « leur » multimodalité de la même manière.

Pour les uns, ce qui est séquentiel, est parallèle chez les autres. Tel mode est dominant chez les uns, et ne l'est pas chez les autres. Tel mode est persistant, etc.

8

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Les modes ne sont pas équivalents.

Leur pouvoir d'expression est différent sans parler de leur pouvoir perceptuel, qui paraît plus évident. Cela signifie que le geste (et inversement la parole) ne peut tout exprimer dans une interaction, il y a des limitations cognitives. Même si parfois on eut rapprocher deux modes, ils n'en différent pas moins par leurs propriétés différentes : précision, latence, etc. Certains modes sont plus inconscients ou passifs que d'autres : la direction du regard par exemple.

Extrait du cours de Laurence Nigay

10 mythes (S. Oviatt)

Un système multimodal n'est pas plus efficace qu'un autre.

On croit souvent qu'un système multimodal sera plus efficace qu'un système monomodal, car on pourra faire plusieurs choses en même temps, se reposer en passant d'un mode à l'autre, réduire la charge perceptive et cognitive, économiser le temps de planification, etc. Des expériences ont prouvé le contraire : une commande multimodale est souvent plus longue à exprimer qu'une commande monomodale, car il y a un coût dû à la multimodalité (par exemple la multimodalité produit un débit de parole plus saccadé et des hésitations plus fréquentes).

10

Exemples de scénarios

- Considérons une application sur un téléphone qui propose une fisheye view.

1. Lorsque je touche l'écran et que je vais sur la gauche, cela agrandit la zone de la déformation. Lorsque je touche l'écran et que je vais sur la droite, cela réduit la zone de la déformation.
   dispositif = l'écran tactile (et la génération d'événement)
   langage = associer un geste vers la gauche ou vers la droite à une action.
2. Toujours sur cette même application, je peux utiliser des commandes vocales pour faire quelques actions : dire « zoomer » pour augmenter le grossissement ou dire « reculer » pour diminuer le grossissement.
   dispositif =le micro
   langage = « langage pseudo-naturel » (un sous ensemble limité du langage)
3. Une variante de cette application : lorsque je touche l'écran cela déplace le centre de la zone de déformation.
   dispositif = écran tactile (et la génération d'événement)
   langage = le changement du centre de la déformation…
4. Une composition de modalité : je dis « zoom » et ensuite le touch+déplacement vers la droite ou vers la gauche détermine si on grossit ou réduit, et de combien

Combinatoire pour une modalité : difficulté de conception

- Pour un même couple <dispositif, langage>
  - des réglages et des paramétrages
  - Dans le cas de l'interaction 1, on pourrait par exemple régler la sensibilité de l'écran tactile, la fréquence d'émission d'événement. On pourrait aussi régler la sensibilité du déplacement vers la gauche : 1 pixel de différence ? 10 ? plus ? pour détecter un mouvement vers la gauche ou vers la droite ?
  - Ainsi pour une modalité, la combinatoire des paramètres peut la changer grandement et la rendre utilisable ou inutilisable.
- Derrière un langage d'une modalité, il y a à priori une tâche.
  - L'appariement tâche-dispositif est du domaine de la conception (c.f. CEIHM).
  - C'est-à-dire qu'en changeant le langage d'interaction associé à un dispositif, on en change l'utilisation et l'activité réalisable avec le dispositif.
  - Si l'on compare les interactions 1 et 3, seul le langage change…
- **Ainsi, dans le cadre d'une application, il existe une combinatoire importante entre les dispositifs utilisés et les langages, ce qui amène à des solutions différentes de conception.**
- Pour une interaction donnée, le contexte peut aussi amener à changer les paramètres. Par exemple pour l'interaction 2, si je suis en train de parler à quelqu'un, le système pourrait capter des mots et les interpréter. Il faudrait alors pouvoir désactiver des modalités dans certaines situations, et donc, si on veut pouvoir continuer à utiliser les fonctionnalités, il faudrait associer une autre modalité à la fonctionnalité.
- **Ainsi, il existe aussi une combinatoire entre les dispositifs, les langages et les situations d'usages.**
- **Combiner des modalités pour obtenir de nouvelles interactions**

TOUCH
MULTITOUCH
GESTURE

Touch

- « CLICK » (1 touch, sans plus d'information:
  - Attribut « onclick » dans le layout (avec un nom de méthode de l'activité courrante)
  - Interface OnClickListener
    - Méthode void onClick (View v)
    - setOnClickListener sur la View
- TOUCH
  - Surcharge (pour une View) de public boolean onTouchEvent(MotionEvent e)
  - Interface View.OnTouchListener
    - Méthode boolean onTouch(View v, MotionEvent event)
      - » Retourne true si l'événement a été prise en compte
    - setOnTouchListener sur la View

MultiTouch (1/2)

- Information dans le MotionEvent (e)
- e.getActionMasked() / e.getActionIndex ()
  - Les actions :
    - ACTION_CANCEL The current gesture has been aborted.
    - ACTION_DOWN A pressed gesture has started, the motion contains the initial starting location.
    - ACTION_HOVER_ENTER The pointer is not down but has entered the boundaries of a window or view.
    - ACTION_HOVER_EXIT The pointer is not down but has exited the boundaries of a window or view.
    - ACTION_HOVER_MOVE A change happened but the pointer is not down (unlike ACTION_MOVE).
    - ACTION_MOVE A change has happened during a press gesture (between ACTION_DOWN and ACTION_UP).
    - ACTION_OUTSIDE A movement has happened outside of the normal bounds of the UI element.
    - ACTION_POINTER_DOWN A non-primary pointer has gone down.
    - ACTION_POINTER_UP A non-primary pointer has gone up.
    - ACTION_SCROLL The motion event contains relative vertical and/or horizontal scroll offsets.
    - ACTION_UP A pressed gesture has finished, the motion contains the final release location as well as any intermediate
  - e.getActionIndex( ) permet de savoir quel “pointeur” en est la source

MultiTouch (2/2)

- e.getPointerCount() : nombre de pointeurs = nombre de points de contact reconnus
- e.getPointerId(*int index*) : un nombre (0,1,2,…) associé à un pointeur
  - Dans l'ordre d'apparition
  - Ré-indexation en cas de disparition…
  - e.getPointerId(*index*) permet d'avoir l'id du *pointer* qui lui ne changera pas jusqu'à sa disparition
- **findPointerIndex (int pointerId) : permet de retrouver l'index à partir d'un id**
- e.getPointerCoords(*int index*)
- Et les autres méthodes avec un paramètre *pointerIndex ou un pointerId :* *http://developer.android.com/reference/android/view/MotionEvent.html*

Geste

- 2 niveaux de reconnaissance de geste
  - http://developer.android.com/reference/android/view/GestureDetector.html
  - http://developer.android.com/reference/android/view/ScaleGestureDetector.html
- Il faut appeler le (ou les) détector dans onTouch : detecor.onTouchEvent(motionEvent);
  - Attention à la fusion des setOnTouchListener
- Le détector appelle en callback une méthode en correspondance avec le geste détecté

GestureDetector

- Pour les « long touch », « scroll » et « swipe »
- Objet utilisé (délégation)
  - Appel depuis onTouch

```
public boolean onTouch(View v, MotionEvent e) {
      detector.onTouchEvent(e);
      return true;  // l'événement a été consommé
   }
```

- Appel en callback un GestureDetector.OnGestureListener

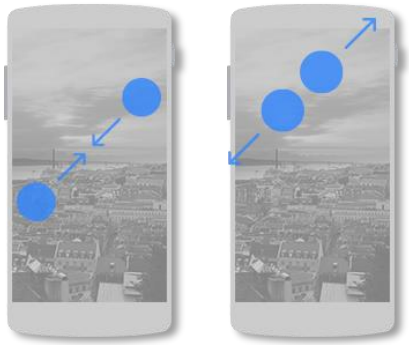
```
new GestureDetector ( Context context,
            GestureDetector.OnGestureListener listener)
```

GestureDetector.OnGestureListener

- boolean onDown (MotionEvent e)
  - Précéde tous les autres events
  - Doit être consommé (return true) pour que onFling ou onScroll soit appelé
- boolean onFling (MotionEvent initial, MotionEvent current, float velocityX, float velocityY)
  - Détection d'un mouvement ≈ touch « dragged » après avoir relevé le doigt
  - Tests sur velocityX (seuil) et
    Test sur current.getX() – initial.getX( ) (seuil et signe)
    Pour savoir s'il y a un swipe horizontal
  - Idem pour Y
- onLongPress (MotionEvent e)
- void onShowPress (MotionEvent e)
  - Après un onDown mais pas bougé depuis. Pour retour d'information
- boolean onSingleTapUp (MotionEvent e)
  - Le doigt se relève… et ne touche plus l'écran
- boolean onScroll (MotionEvent initial, MotionEvent current, float distanceX, float distanceY)
  - Pour les scrolls, comme fling, mais le doigt touche encore l'écran

- Dans la suite « normale » :
  - On appuie sur l'écran avec le doigt : onTouch / onDown
  - On déplace le doigt : suite de onTouch / onScroll
  - On relève le doigt : onTouch / onFling

ScaleDetector



- Pour les zooms avec un pitch
  - C.f. image de http://developer.android.com/design/patterns/gestures.html
- Objet utilisé en délégation
- Appel en callback un ScaleGestureDetector.OnScaleGestureListener

```
new ScaleGestureDetector (
Context context,
ScaleGestureDetector.OnScaleGestureListener listener)
```

ScaleGestureDetector.OnScaleGestureListener

- 3 méthodes
  - boolean onScale(ScaleGestureDetector detector)
  - Boolean onScaleBegin(ScaleGestureDetector detector)
  - onScaleEnd(ScaleGestureDetector detector)
- On obtient les informations par rappel au détecteur
  - Par exemple : detector. getScaleFactor() pour avoir le facteur d'échelle

Combiner plusieurs détecteurs

- 1 seul onTouch qui appel tous les détecteurs
  - Directement
  - Ou indirectement en appelant des onTouch pour un fonctionnement isolé

Commande vocale,

Text to Speech,

Lecture de son

AUDIO / VOCALS

Reconnaissance Vocale (service Google)

- Google propose une « activité » pour la reconnaissance vocale.
  - RecognizerIntent.ACTION_RECOGNIZE_SPEECH
  - http://developer.android.com/reference/android/speech/RecognizerIntent.html
  - Pas d'autorisation particulière
- On lance donc cette activité depuis l'activité de l'application, dans l'attente d'un résultat.
  - Pour le paramétrage, on utilise un Intent
- On reçoit le résultat dans un autre Intent

Vérifier la disponibilité de l'activité « recognition »

```
PackageManager pm = getPackageManager();
List<ResolveInfo> activities =
pm.queryIntentActivities( new Intent(
RecognizerIntent.ACTION_RECOGNIZE_SPEECH ),
0);

if (activities.size() != 0) {
     // disponible…
} else {
     // non disponible…
}
```

Lancer la reconnaissance vocale

```
// on est dans une activité
// lancement de l'intent avec le nom de la classe... (constante)
Intent intent = new Intent(
       RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
             RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);

// titre de la boite de dialogue
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Fisheye - Speech recognition");

// VOICE_CODE pour retrouver l'appel pour le résultat
// nombre choisi par le développeur
startActivityForResult(intent, VOICE_CODE);
```

Réception du résultat

- **Pour test unitaire : isoler dans une méthode le traitement du retours dans une méthode**

```
protected void onActivityResult(int requestcode, int resultcode,
Intent data) {
        // test si le retour vient de la reconnaissance vocale
        // et si l'activité à bien fonctionnée
  if (requestcode == VOICE_CODE && resultcode == RESULT_OK) {
        ArrayList<String> matches =
  data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        processVoiceInput(matches);
        }
  else {
        // autres retours d'activité…
  }
}
```

Traitement du résultat

```
protected void processVoiceInput(ArrayList<String> matches) {
if (nb_rep > 0)
        {
        // matches contient les réponses potentielles
        // il faut regarder dans ces réponses s'il y a
        // une valeur attendue…
        // ici sur les 3 1ère réponses
        int nb_test = Math.min(3, nb_rep);

        for(int i = 0; i < nb_test; i++) {
                if (matches.get(i). equalsIgnoreCase("commande")) {
                        // faire l'action associée à la commande
                        break;
                        }
                // etc.
                }
        }
}
```

Text To Speech

- Lecture de texte
- Mode asynchrone
  - lecture sur un autre thread que le thread principal
- Téléchargement de voix lors de la 1ère utilisation

Initialisation de TTS

```
// this est l'activité en cours, c'est surtout un Context
tts = new TextToSpeech(this, new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            // traitement en cas de succes
            // par exemple activation d'un bouton lecture
            play.setEnabled(true) ;
            play.setOnClickListener(NomDeLaClasseActivity.this);

            // NomDeLaClasseActivity.this : référence à l'activité,
            //car ici this = le onInitListener

            // déterminer la langue
            tts.setLanguage(Locale.FRANCE);

            // pour savoir quand un texte est fini d'être lu
            tts.setOnUtteranceProgressListener(ttsListener);
            }
        }
});
```

Lire un texte (vers sdk < 21)

```
// version sdk < 21
HashMap<String, String> params = new HashMap<String,
String>();
// pour recevoir les événements de fin de lecture
params.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,
"id"+System.currentTimeMillis());
// écrase une éventuelle lecture en cours
// possibilité d'ajouter : TextToSpeech.QUEUE_ADD
tts.speak(txt,  TextToSpeech.QUEUE_FLUSH, params);
```

- Version sdk < 21 : speak(String text, int queueMode, HashMap<String, String> params)
- Version sdk 21 : speak(CharSequence text, int queueMode, Bundle params, String utteranceId)

Listener sur le TTS

```
// class abstraite…
UtteranceProgressListener ttsListener = new UtteranceProgressListener() {
        @Override
        public void onStart(String utteranceId) {
           // ...
        }

        @Override
        public void onDone(String utteranceId) {
           // ...
         }

        @Override
        public void onError(String utteranceId) {
           // ...
        }
    };

// exécuté potentiellement sur un thread différent du thread principal
// donc => lactivité.runOnUiThread( ... ) pour des mises à jours graphiques
```

MediaPlayer

- http://developer.android.com/guide/topics/media/mediaplayer.html
- http://developer.android.com/reference/android/media/MediaPlayer.html

- Exemple avec un Service

- *D'autres possibilités (play list…)*

Les capteurs disponibles

Le gps

SENSORS

Accelerometers

- Interface SensorEventListener
- Connecter
  - À lancer dans onCreate ou onStart

```
SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Sensor mAccelero = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sm.registerListener(the_listener, mAccelero, SensorManager.SENSOR_DELAY_UI);
```

- Plusieurs capteurs
  - http://developer.android.com/reference/android/hardware/Sensor.html
  - Lumière, pression, … et `Sensor.TYPE_ACCELEROMETER`
  - Présence non garantie (dépend du dispositif)
- Déconnecter

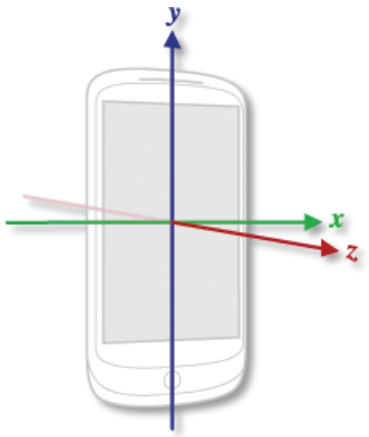
```
mSensorManager.unregisterListener(the_listener);
```

  - si on sort de l'application / onStop - onPause
  - À reconnecter si on revient / onResume

onSensorChanged (1/2)

- Listener à séparer de l'application (test)
- public void onSensorChanged(SensorEvent se)
- Tenir compte de l'orientation

```
WindowManager wm = (WindowManager)
getSystemService(Context.WINDOW_SERVICE);
Display display = wm.getDefaultDisplay();
display.getRotation();
```



- On obtient des accélérations…
  - selon les 3 axes avec la gravité incluse
  - Il faut un gyroscope (pour Sensor.*TYPE_MAGNETIC_FIELD*) pour « extraire » la gravité
    - Absent de la tablette…

TIM 2015-2016

onSensorChanged (2/2)

```
@Override
public void onSensorChanged(SensorEvent sensorEvt) {
    if (sensorEvt.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
        // traitement des valeurs
        // dans le tableau sensorEvent.values
        // qui contient dans l'ordre
        // les accélérations selon x, y, z
    }
}
```

Accéléromètre et Gyroscope (1/3)

Etape 1 : déclaration

```
// Il faudra implémenter l'interface :
// SensorEventListener

// variable nécessaire pour s'abonner
      private SensorManager mSensorManager;
      private Sensor mAccelerometer;
```

Etape 2 : retrouver le capteur fourni par le système

```
// ce code est extrait d'une Activity
// Là c'est pour savoir les changements d'orientation du téléphone
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
// pour obtenir la « gravité »
mAccelerometer=mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
// pour obtenir le champ magnétique
mMagnetic = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Accéléromètre et Gyroscope (2/3)

Etape 3 : s'abonner par exemple quand l'application est réactivée

```
protected void onResume() {
        super.onResume();
        // on s'abonne en précisant le listener, le capteur
        // et le type de fréquence de réception des événements
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_UI);
        mSensorManager.registerListener(this, mMagnetic, SensorManager.SENSOR_DELAY_FASTEST);
}
```

Etape 3 bis : on doit se désabonner quand on quitte l'application

```
// ce code est extrait d'une Activity
protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
}

@Override
protected void onStop() {
        super.onStop();
        mSensorManager.unregisterListener(this);

}
```

Accéléromètre et Gyroscope (3/3)

Etape 4 : on implémente `SensorEventListener`

```
public void onAccuracyChanged(Sensor sensor, int accuracy) { }

float [] gravity = new float[3];
float [] geomagetic = new float[3];

public void onSensorChanged(SensorEvent event) {
                // pour obtenir l'orientation, il faut avoir des matrices de rotation et d'inclinaison
                // pour les avoir, il faut la gravité et le champ magnétique
                if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
                                for(int i=0; i<3; i++){  geomagetic[i] =  event.values[i];   }
                                }

                if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                                for(int i=0; i<3; i++){  gravity[i] =  event.values[i];   }
                                }


                if ((gravity[2] == 0) || (geomagetic[0] == 0)) { // on n'a pas les infos pour calculer les matrices
                                return;
                                }

                // calcul des matrices
                float[] RotationMatrix = new float[9];
                float[] I = new float[9];
                SensorManager.getRotationMatrix(RotationMatrix, I, gravity, geomagetic);

                // calcul de l'orientatoin
                float [] values = new float[3];
                SensorManager.getOrientation(RotationMatrix, values);

                // il reste à faire le traitement de l'orientation
                }
```

Sensors de la tablette

```
name = android.hardware.wifi / FeatureInfo{41bc4110 android.hardware.wifi fl=0x0}
name = android.hardware.location.network / FeatureInfo{41bc4188 android.hardware.location.network fl=0x0}
name = android.hardware.bluetooth_le / FeatureInfo{41bc4218 android.hardware.bluetooth_le fl=0x0}
name = android.hardware.location / FeatureInfo{41bc42a0 android.hardware.location fl=0x0}
name = android.software.input_methods / FeatureInfo{41bc4320 android.software.input_methods fl=0x0}
name = android.hardware.sensor.gyroscope / FeatureInfo{41bc43a8 android.hardware.sensor.gyroscope fl=0x0}
name = android.hardware.screen.landscape / FeatureInfo{41bc4438 android.hardware.screen.landscape fl=0x0}
name = android.hardware.screen.portrait / FeatureInfo{41bc44c8 android.hardware.screen.portrait fl=0x0}
name = android.hardware.usb.accessory / FeatureInfo{41bc4558 android.hardware.usb.accessory fl=0x0}
name = android.hardware.camera.any / FeatureInfo{41bc45e0 android.hardware.camera.any fl=0x0}
name = android.hardware.touchscreen.multitouch.distinct / FeatureInfo{41bc4668 android.hardware.touchscreen.multitouch.distinct fl=0x0}
name = android.hardware.bluetooth / FeatureInfo{41bc4718 android.hardware.bluetooth fl=0x0}
name = android.hardware.microphone / FeatureInfo{41bc4798 android.hardware.microphone fl=0x0}
name = android.software.live_wallpaper / FeatureInfo{41bc4820 android.software.live_wallpaper fl=0x0}
name = android.software.app_widgets / FeatureInfo{41bc48b0 android.software.app_widgets fl=0x0}
name = android.software.sip / FeatureInfo{41bc4938 android.software.sip fl=0x0}
name = android.hardware.touchscreen.multitouch.jazzhand / FeatureInfo{41bc49b0 android.hardware.touchscreen.multitouch.jazzhand fl=0x0}
name = android.hardware.usb.host / FeatureInfo{41bc4a60 android.hardware.usb.host fl=0x0}
name = android.hardware.touchscreen.multitouch / FeatureInfo{41bc4ae0 android.hardware.touchscreen.multitouch fl=0x0}
name = android.hardware.faketouch / FeatureInfo{41bc4b80 android.hardware.faketouch fl=0x0}
name = android.hardware.camera / FeatureInfo{41bc4c00 android.hardware.camera fl=0x0}
name = android.software.home_screen / FeatureInfo{41bc4c80 android.software.home_screen fl=0x0}
name = android.software.sip.voip / FeatureInfo{41bc4d08 android.software.sip.voip fl=0x0}
name = android.hardware.location.gps / FeatureInfo{41bc4d88 android.hardware.location.gps fl=0x0}
name = android.software.device_admin / FeatureInfo{41bc4e10 android.software.device_admin fl=0x0}
name = android.hardware.camera.front / FeatureInfo{41bc4e98 android.hardware.camera.front fl=0x0}
name = android.hardware.touchscreen / FeatureInfo{41bc4f20 android.hardware.touchscreen fl=0x0}
name = android.hardware.sensor.accelerometer / FeatureInfo{41bc4fa8 android.hardware.sensor.accelerometer fl=0x0}
name = null / FeatureInfo{41bc5040 glEsVers=2.0 fl=0x0}
```

GPS

- Description complète à http://developer.android.com/guide/topics/location/strategies.html
- Principe d'abonnement / callback
  - Attention à la modularité (pour test…)
  - « Mock » / via adb (erf) / avec une application qui simule le gps (fake gps)

Pour utiliser le gps

- Permission :

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

- Méthode getSystemService de contexte

```
LocationManager locationManager = (LocationManager)
uneActivite.getSystemService(Context.LOCATION_SERVICE);
```

- Abonnement aux événements

```
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0,
0, aLocationListener);
```

- Interface logicielle LocationListener

```
public void onLocationChanged(Location location)
public void onStatusChanged(String provider, int status, Bundle extras)
public void onProviderEnabled(String provider)
public void onProviderDisabled(String provider)
```

Location

- http://developer.android.com/reference/android/location/Location.html
- Contient latitude, longitude, etc.
- Notez la méthode distanceTo…