RENDU

TP2 : Les Capteurs

HAI811I - Programmation Mobile

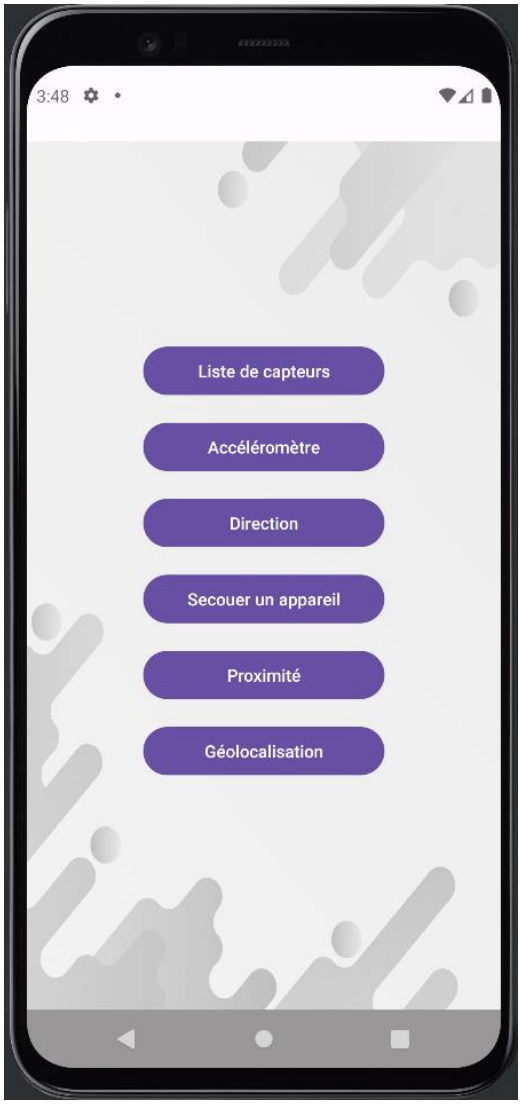
Adam DAIA
Mohammed DAFAOUI
M1 GL

Table de matières

Introduction

Les capteurs sont des composants essentiels dans nos appareils électroniques modernes, permettant une variété de fonctionnalités et d'interactions intuitives. Ce rapport se concentre sur l'exploration et l'exploitation des capacités des capteurs dans un smartphone, en mettant en œuvre différentes applications pour illustrer leurs utilisations pratiques.

Les capteurs les plus couramment utilisés dans les smartphones Android incluent l'accéléromètre, le gyroscope, le magnétomètre, le capteur de proximité, le capteur de luminosité et le capteur de pression. Chacun de ces capteurs joue un rôle important dans la fourniture de fonctionnalités utiles à l'utilisateur, telles que l'orientation de l'écran, la stabilisation de l'image, la mesure de la distance, la mesure de la luminosité ambiante et la surveillance de la santé.



Exercice 1 : Liste de capteurs

L'application se compose de trois boutons permettant d'accéder aux différentes fonctionnalités de l'application :

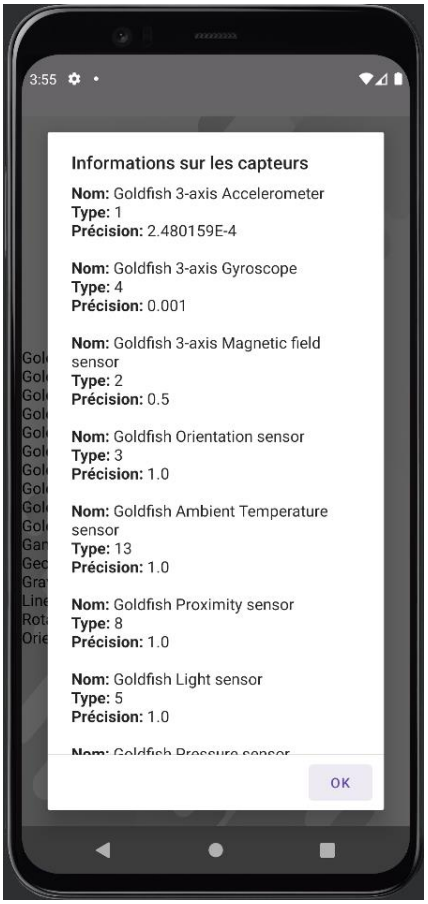
Nous avons utilisé les fonctionnalités fournies par le framework Android, notamment la classe SensorManager, pour aux capteurs du smartphone. Lorsque l'utilisateur appuie sur le bouton "Tous les capteurs", l'application récupère la liste de tous les capteurs disponibles et les affiche à l'écran.

De plus, l'application permet également d'afficher des informations détaillées sur chaque capteur dans une boîte de dialogue, en utilisant la classe AlertDialog. Les informations affichées comprennent le nom du capteur, son type et sa précision.

Enfin, l'application simule la désactivation d'un capteur spécifique en filtrant la liste des capteurs disponibles et en marquant le capteur "Goldfish 3-axis Gyroscope" comme indisponible.

"**Tous les capteurs**" : Ce bouton permet d'afficher une liste de tous les capteurs disponibles sur le smartphone.

"**Informations**" : Ce bouton affiche des informations détaillées sur chaque capteur, y compris leur nom, type et précision.



Exercice 2 : Détection de présence/absence de capteurs

Le troisième bouton "**Désactivation du capteur**" : Ce bouton simule la désactivation d'un capteur spécifique, en l'occurrence le capteur "*Goldfish 3-axis Gyroscope*".

Exercice 3 : Accéléromètre

L'application a été spécialement conçue pour diviser l'écran en trois zones distinctes : une zone supérieure, une zone médiane et une zone inférieure. Chacune de ces zones est attribuée à une couleur spécifique : vert pour les valeurs d'accélération inférieures, noir pour les valeurs moyennes et rouge pour les valeurs supérieures détectées par l'accéléromètre du smartphone.

En utilisant les fonctionnalités du framework Android, l'application récupère en temps réel les données d'accélération du smartphone lorsque l'utilisateur effectue des mouvements.

Ces données sont ensuite utilisées dans la méthode *onSensorChanged()* de l'interface *SensorEventListener*. En extrayant les valeurs d'accélération sur les trois axes (x, y, z) à partir de l'événement du capteur, l'application détermine la couleur de chaque zone de l'écran en conséquence. Plus précisément, elle calcule d'abord l'accélération totale en appliquant la formule sqrt(x^2 + y^2 + z^2), puis compare cette valeur à des seuils prédéfinis pour l'évaluation dans quelle plage d'accélération se trouve l 'appareil.

En fonction de cette plage, la couleur des zones correspondantes est modifiée en utilisant la méthode setBackgroundColor(), qui récupère les couleurs à partir du fichier .

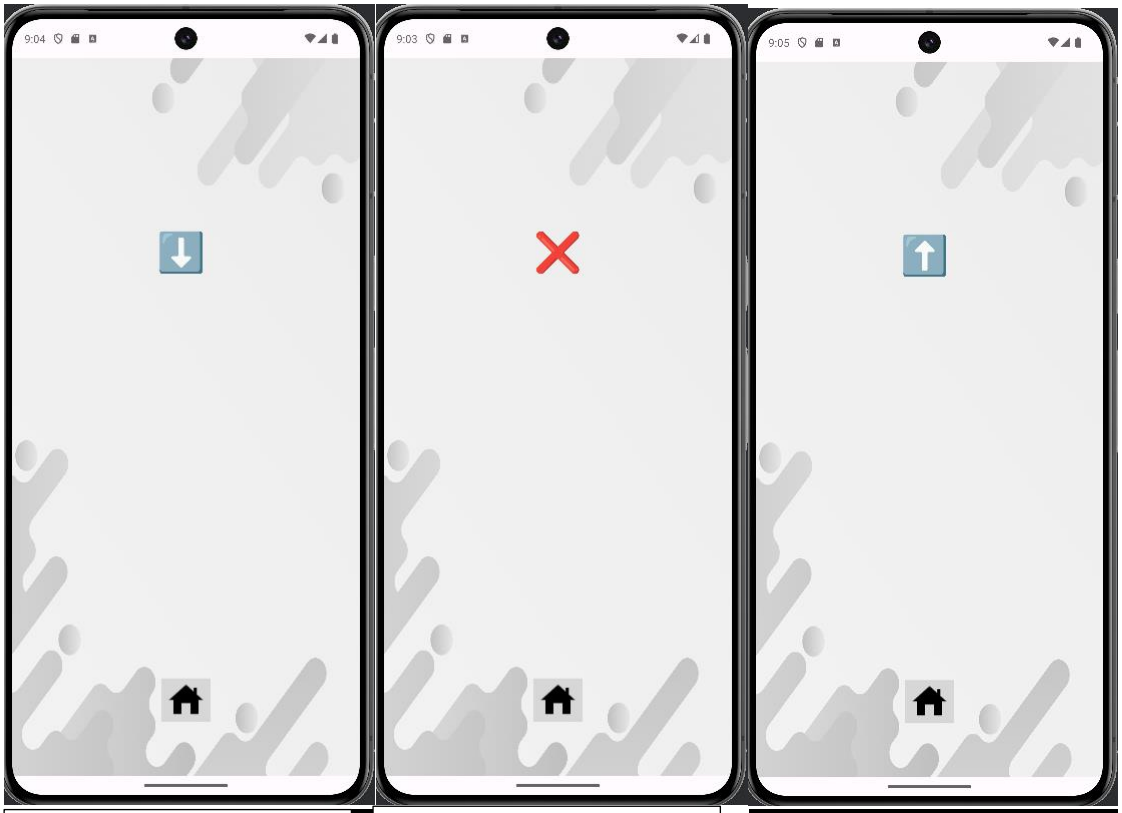
Enfin, la méthode *onAccuracyChanged()* est appelée en cas de changement de précision du capteur, offrant ainsi une gestion complète de l'expérience utilisateur. Cette approche méthodique et technique permet une utilisation immersive de l'accéléromètre, offrant ainsi une interaction dynamique avec l'application tout en offrant une expérience visuelle riche et intuitive pour les utilisateurs.

Exercice 4 : Direction

L'activité Direction utilise le capteur d'accélération linéaire du smartphone pour détecter les mouvements de l'utilisateur. L'écran affiche une flèche indiquant la direction du mouvement détecté en temps réel. Lorsque l'utilisateur effectue un mouvement significatif, tel qu'un mouvement vers la gauche, la droite, le haut, le bas, l'avant ou l'arrière, la flèche change de direction en conséquence.

Cette fonctionnalité est réalisée en écoutant les changements du capteur d'accélération linéaire à l'aide d'un SensorEventListener. Lorsqu'un mouvement significatif est détecté, la direction du mouvement est calculée en fonction des valeurs d'accélération sur les trois axes (x, y, z) du capteur. Ensuite, la flèche affichée à l'écran est mise à jour pour indiquer la nouvelle direction.

En plus de détecter les mouvements, l'activité gère également la fiabilité des données du capteur en vérifiant périodiquement la précision du capteur à l'aide de la méthode onAccuracy-Changed(). Cette approche permet une expérience utilisateur fluide et fiable lors de l'utilisation de l'application



Mouvement vers le bas

Aucun Mouvement

Mouvement vers le haut

Mouvement vers la droite

Mouvement vers la gauche

Exercice 5 : Secouer un appareil

Pour allumer le flash en secouant l'appareil et l'éteindre également en le secouant, nous avons suivi les étapes suivantes dans Android Studio :

- Ajouter la permission pour accéder à la caméra et au flash dans le *Manifest.*

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-permission android:name="android.permission.FLASHLIGHT"/>
```

- Créer un *SensorEventListener* pour détecter les mouvements de secousse.
- Lorsque la vitesse de mouvement dépasse le seuil *SHAKE_THRESHOLD*, la fonction *toggleFlashlight()* est appelée pour allumer ou éteindre le flash.

Avant toutes secousses

Après une 1ère secousse

Après une 2ème secousse

Exercice 6 Proximité :

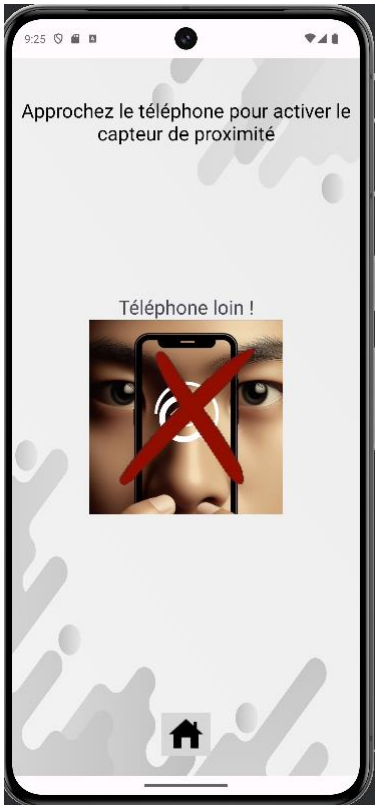
L'activité `ProximityActivity` utilise le capteur de proximité du smartphone pour détecter la proximité d'un objet par rapport au téléphone. Lorsque l'activité démarre, elle initialise le `SensorManager` et récupère le capteur de proximité à l'aide de `getDefaultSensor(Sensor.TYPE_PROXIMITY)`.

Ensuite, elle référence une `ImageView` et un `TextView` pour afficher respectivement l'image représentant l'état de la proximité et le texte descriptif. De plus, un `ImageButton` est utilisé pour permettre à l'utilisateur de revenir en arrière et de quitter l'activité.

Lorsque l'activité est en cours, elle enregistre un `SensorEventListener` pour écouter les changements de proximité à l'aide de `registerListener(this, proximitySensor, SensorManager.SENSOR_DELAY_NORMAL)`. Cette écoute se fait avec la méthode `onSensorChanged(SensorEvent event)`.

À chaque changement de proximité détecté, la méthode `onSensorChanged` est appelée. Si le changement concerne le capteur de proximité, l'activité met à jour l'image affichée à l'écran en fonction de la proximité détectée. Si l'objet est proche, l'image est définie sur une image représentant la proximité, sinon elle est définie sur une image indiquant que l'objet est loin. De plus, le texte descriptif est également mis à jour pour informer l'utilisateur de l'état de la proximité.

Enfin, lors de la mise en pause de l'activité, le `SensorEventListener` est désenregistré pour économiser les ressources du smartphone à l'aide de `unregisterListener(this)` dans la méthode `onPause()`



Avant rapprochement du téléphone

Après rapprochement du téléphone

Exercice 7 : Géolocalisation

L'activité `LocationActivity` utilise les services de localisation de Google Play pour obtenir la position géographique de l'appareil. Au démarrage de l'activité, elle initialise l'interface utilisateur en référençant un `TextView` pour afficher les coordonnées géographiques et un `ImageButton` pour permettre à l'utilisateur de revenir en arrière.

Ensuite, elle vérifie les autorisations de géolocalisation en appelant `checkLocationPermissions()`. Si les autorisations ne sont pas accordées, elle demande à l'utilisateur de les accorder. Une fois les autorisations accordées, elle crée une requête de localisation à l'aide de `createLocationRequest()` pour obtenir des mises à jour de la position géographique.

La méthode `createLocationRequest()` crée une requête de localisation avec une priorité élevée et vérifie à nouveau les autorisations. Si les autorisations sont accordées, elle demande des mises à jour de la localisation à l'aide de `requestLocationUpdates()`. Si les autorisations ne sont pas accordées, elle demande à nouveau à l'utilisateur de les accorder.

Lorsque des mises à jour de localisation sont disponibles, elles sont capturées par `onLocationResult()` de l'objet `locationCallback`. Cette méthode met à jour l'interface utilisateur avec les nouvelles coordonnées géographiques et utilise les services de géocodage inversé pour obtenir l'adresse correspondante.

En cas de refus des autorisations de géolocalisation par l'utilisateur, la méthode `onRequestPermissionsResult()` est appelée pour gérer la réponse de l'utilisateur. Si les autorisations sont accordées, elle appelle `createLocationRequest()` pour obtenir la position géographique. Sinon, elle affiche un message d'erreur.

Enfin, lorsque l'activité est détruite, la méthode `onDestroy()` est appelée pour arrêter les mises à jour de localisation à l'aide de `removeLocationUpdates()` afin d'économiser les ressources du smartphone

Conclusion :

En résumé, les capteurs disponibles sous Android Studio offrent une palette riche de fonctionnalités pour les développeurs d'applications mobiles. Ils permettent de recueillir une variété de données telles que l'accélération, l'orientation, la luminosité et la proximité, offrant ainsi des possibilités infinies pour la création d'applications interactives et intuitives.

L'association de plusieurs capteurs peut également permettre d'obtenir des données plus précises et des fonctionnalités plus complètes, ouvrant ainsi la voie à des applications encore plus innovantes.

Cependant, il est essentiel de prendre en considération l'impact de l'utilisation des capteurs sur la durée de vie de la batterie et les performances du dispositif mobile. Pour cette raison, les développeurs doivent optimiser leur code et limiter la collecte de données aux seules informations nécessaires, garantissant ainsi une expérience utilisateur fluide et une utilisation efficace des ressources du smartphone.