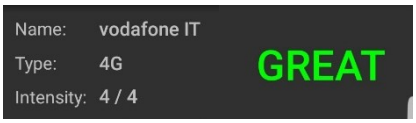
SignalHEX

Mattia Biondi

Corso di Laboratorio di Applicazioni Mobili, A.A. 2017/2018

Introduzione

SignalHEX è un'applicazione per Android scritta in Kotlin e sviluppata nell'IDE Android Studio. Lo scopo dell'app è monitorare l'intensità del segnale ricevuto dal dispositivo di quattro diverse tecnologie: Edge (2G), UMTS (3G), LTE (4G) e Wi-Fi.



L'applicazione è in grado di riconoscere il tipo di rete attualmente in uso e di classificare il segnale secondo cinque livelli di intensità (*none, poor, moderate, good* e *great*). Il tutto viene riportato a schermo in tempo reale, con il livello di intensità colorato secondo una legenda (per valore predefinito dal rosso al verde).

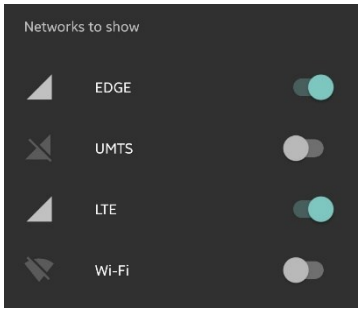
Concedendo all'applicazione l'autorizzazione ad accedere alla posizione del dispositivo, viene visualizzata una mappa fornita da Google Maps nell'area sottostante; all'interno di essa viene mostrata l'attuale posizione del *device* ed i classici comandi d'interazione forniti da Google Maps. Nella parte inferiore dello schermo sono indicate le coordinate corrispondenti all'attuale posizione. L'autorizzazione alla posizione fornisce inoltre la possibilità di interagire con tutti i pulsanti presenti all'interno dell'interfaccia utente e precedentemente "bloccati".

Interfaccia utente

Premendo sul pulsante in alto a destra a forma di freccia (*play*), si avvia la scansione dell'area e sulla mappa viene disegnato un esagono regolare attorno al segnalino di posizione del dispositivo. Muovendosi fisicamente con il *device* e uscendo dall'area disegnata dall'esagono, si presenta a schermo un nuovo esagono adiacente al precedente e di uguale dimensione. L'operazione è ripetuta ogni volta che si entra in una nuova area non precedentemente rilevata, fino al punto di ricoprire l'intera superficie della mappa con una griglia di esagoni regolari. Il colore di ciascuna forma geometrica corrisponde al relativo colore dell'intensità rilevata al momento della creazione di essa secondo la legenda. Nel caso in cui il dispositivo rilevi una variazione d'intensità del segnale all'interno di un'area già rilevata, il colore relativo all'esagono disegnato verrà aggiornato con il colore della nuova intensità.

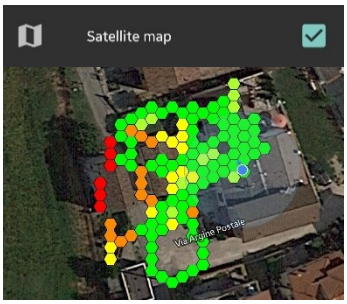
Un pulsante di "*refresh*" è presente sulla barra superiore e fornisce la possibilità di ripristinare la mappa al suo stato originale, rimuovendo tutti i poligoni presenti. Premendo sul pulsante durante il corso di una scansione si interromperà il processo portando lo stato dell'app su *"pause"*.

Sul lato sinistro della barra superiore è presente il pulsante dedicato ad aprire il *drawer* laterale, contenente impostazioni di visualizzazione degli esagoni. È possibile accedere a questo menù anche trascinando il dito verso il centro partendo dal lato sinistro dello schermo.



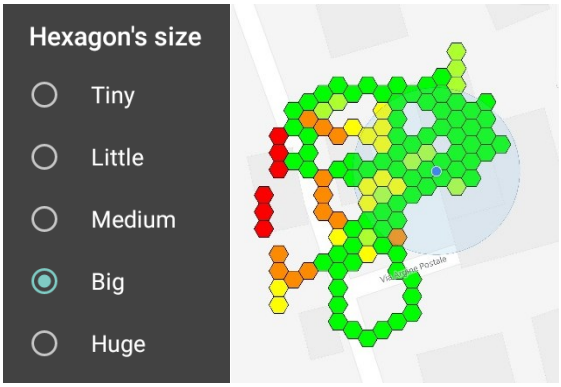
Il primo gruppo di impostazioni è composto da quattro *switch*, uno per ogni tecnologia supportata dall'applicazione. La funzione di questi ultimi è quella di rendere visibili/invisibili gli esagoni sulla mappa rappresentanti la tecnologia corrispondente. In questo modo è possibile formare fino a $2^4 = 16$ visualizzazioni diverse della stessa mappa. Questa funzione torna particolarmente utile quando il telefono passa da una tecnologia all'altra, in quanto l'applicazione non si interrompe e continua a formare esagoni allineati, rendendo impossibile distinguere le varie tecnologie.

Il secondo gruppo di impostazioni è composta da un solo elemento, che modifica il tipo di mappa visualizzata permettendo di scegliere tra “satellite” e “normale”. La visualizzazione “satellite” è utile quando si rileva l’intensità in un luogo aperto (ad esempio in piena campagna), dove le strade non sono mappate da Google Maps, mostrando quindi un’area di un unico colore in modalità “normale”, rendendo impossibile per l’utente capire dove si trovi o a che livello di zoom.

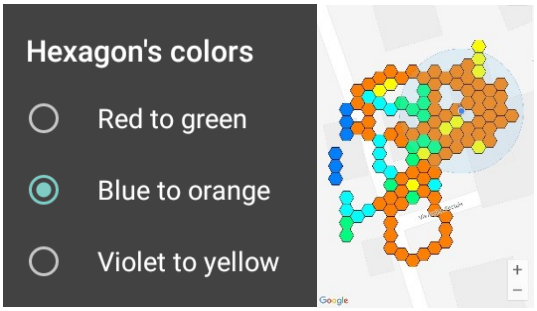


Il terzo gruppo di impostazioni gestisce le impostazioni di visualizzazione degli esagoni sulla mappa, permettendo di modificarne dimensione, colore e trasparenza.

Una maggiore dimensione degli esagoni diminuisce la precisione con cui viene mappata un’area, ma torna utile nel caso si debba mappare una vasta superficie ed il segnale cambi poco frequentemente (ad esempio nel caso si stia rilevando l’intensità di una rete cellulare).



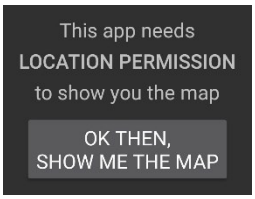
La scelta del colore va a modificare la scala cromatica delle intensità, cambiando sia il colore della scritta in alto, sia il colore di tutti gli esagoni.



L’ultima opzione permette di modificare la trasparenza dei colori degli esagoni, utile quando dopo aver mappato completamente un’intera area, si voglia visualizzare la mappa sottostante ai poligoni. Diverse percentuali di trasparenza favoriscono o diminuiscono la possibilità di vedere attraverso gli esagoni.

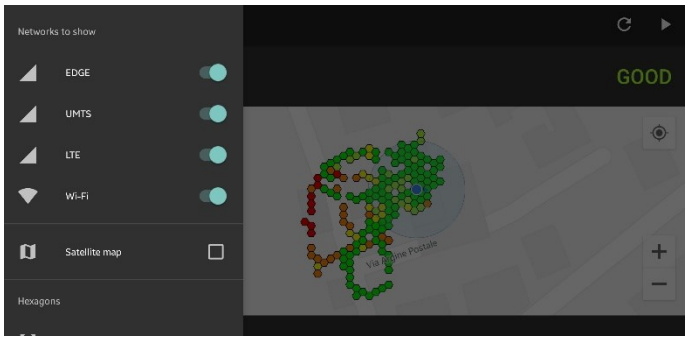
Caratteristiche e requisiti

L’applicazione funziona su tutti i dispositivi aventi almeno Android 6.0 e richiede quattro permessi per funzionare: il permesso di accedere allo stato della rete cellulare e a quello della rete Wi-Fi, l’accesso grossolano alla posizione e l’accesso preciso alla posizione; solamente quest’ultimo ha bisogno del permesso esplicito dell’utente per essere ottenuto.



L’accesso alla mappa e alle sue opzioni di visualizzazione è ottenuto solamente dopo aver concesso i relativi permessi. Se i permessi vengono rimossi dopo essere stati concessi, l’applicazione ritorna allo stato iniziale bloccando ogni impostazione e permettendo all’utente di riconcedere i permessi.

L’applicazione supporta la modalità “*landscape*”, permettendo all’utente di mantenere lo stato esatto dopo la rotazione del dispositivo. Ogni modifica effettuata viene inoltre mantenuta in memoria anche dopo il termine dell’applicazione, fornendo all’utente lo stesso profilo di impostazioni e le aree mappate durante l’utilizzo precedente dell’app.



Non ci sono requisiti espliciti per l’ambiente di esecuzione, l’applicazione funziona anche in totale assenza di segnale in quanto è uno “stato” riconosciuto dell’intensità (*none*).

Progettazione e scelte implementative
Per l’ambiente di sviluppo è stato fondamentale l’utilizzo di un device reale per velocizzare i test in quanto l’esecuzione dell’app su ADV è rasente all’inutilità.

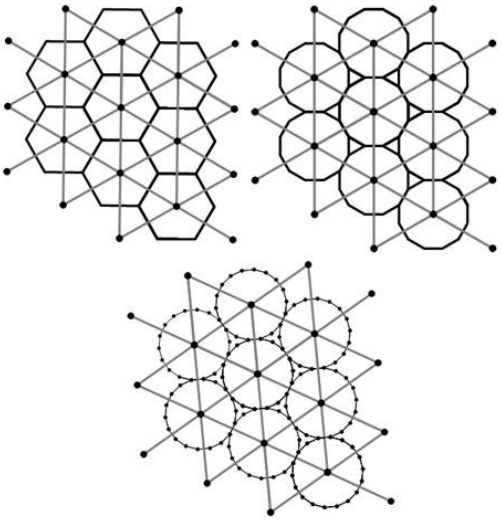
La decisione di utilizzare Kotlin come linguaggio per sviluppare l’applicazione è data dal fatto che questo linguaggio è destinato a prendere il posto del Java nella programmazione di app per Android nel giro di qualche anno. Dovendo imparare ad utilizzare uno dei due linguaggi, ho preferito scegliere il più utile per il futuro.

L’interfaccia utente è stata resa il più intuitiva e semplice possibile, senza limitare le opzioni di personalizzazione. Essendo il focus principale sulla mappa, ho scelto una *Navigation Drawer Activity* per raggruppare tutte le principali impostazioni nella comoda barra laterale.

Nel caso in cui l’utente non voglia concedere il permesso alla posizione all’applicazione, essa continuerà a funzionare limitatamente, permettendo di conoscere in tempo reale la qualità del segnale. Questo permette di non incentrare l’intero funzionamento dell’app sui permessi richiesti.

Il requisito di un API alto è dato dalla funzione *getLevel()* della classe *SignalStrength* che permette di ottenere un intero da 0 a 4 rappresentante l’intensità generale del segnale. La funzione viene chiamata ogni volta che l’intensità di segnale cambia, permettendo di avere sempre l’ultimo valore corretto in tempo reale.

La scelta di utilizzare gli esagoni come forma geometrica per interpolare le aree è derivata dalla necessità di ricoprire l’intera superficie senza lasciare spazi vuoti all’interno della mappa. Il cerchio è la forma geometrica più indicata per rappresentare un’area di egual raggio partendo dalla posizione del dispositivo, ma causa inevitabilmente “buchi” a meno che non vengano sovrapposti. Sovrapponendo i cerchi si sarebbero “nascoste” le rilevazioni precedenti, creando intuitivamente imprecisioni oltre ad una cattiva resa visiva. Per questo la scelta è ricaduta sull’esagono, ovvero il poligono con più lati con il quale è possibile creare una griglia senza buchi.



Il nome dell’applicazione deriva dall’unione della parola “*signal*” con l’abbreviazione di “*hexagon*”.

L'applicazione non ha tasti "salva" o "carica", in quanto avrebbero reso più meccanico il funzionamento. L'intera mappa si salva in memoria assieme alle impostazioni di visualizzazione ogni volta che viene chiamata la *onPause()* e viene ricaricato il tutto durante *onResume()*. Questa scelta implementativa rende fluido l'utilizzo dell'app e l'utente non ha bisogno di caricare o salvare manualmente i progressi ogni volta.

Difficoltà e soluzioni

Durante lo sviluppo ho avuto difficoltà a reperire il corretto livello d'intensità del segnale mobile utilizzando lo stesso codice su più telefoni. Avendo la possibilità di testare l'applicazione su quattro dispositivi Android diversi, ho notato come la funzione *getAllCellInfo()* ritornasse "*null*" sul device più vecchio. La soluzione consigliata è quella di utilizzare un metodo più vecchio e deprecato, *getCellLocation()*, che inoltre ritorna "*null*" nel caso ci sia un'unica cella attiva con segnale LTE. Non avendo dunque modo di utilizzare questi metodi, ho deciso di imporre un SDK minimo per l'esecuzione dell'applicazione, avendo così modo di utilizzare la funzione *getLevel()* che risolve ogni problema implementativo e rende il reperimento della qualità del segnale più preciso e veloce.

La seconda principale difficoltà è stata l'utilizzo del *Heatmap Utility* di Google Maps. Quella che ad un primo impatto sembrava la miglior soluzione possibile per questa applicazione, si è invece dimostrata, una volta implementata, come incompatibile su più fronti. Studiando il codice delle API si nota che la mappa di calore di Google si basi sulla densità di valori in un punto e non sulla loro intensità. Esiste la possibilità di utilizzare coordinate pesate, ma il loro peso viene agglomerato a tutte le rilevazioni adiacenti, creando falsi valori a determinati livelli di *zoom*. Inoltre, non c'è un limite massimo del valore d'intensità, e quest'ultimo influisce sulla trasparenza dei colori con cui si descrive la *heatmap*. Questo funzionamento porta all'invisibilità di aree in realtà rilevate, in quanto codificando una grande quantità di valori in un'area ed un minor numero in un'altra, la seconda non ha modo di "farsi vedere" in quanto la sua densità non raggiunge quella della prima. Questi problemi mi hanno portato a scegliere un metodo più classico per rappresentare i dati ottenuti.

Aspetti rilevanti per lo sviluppo

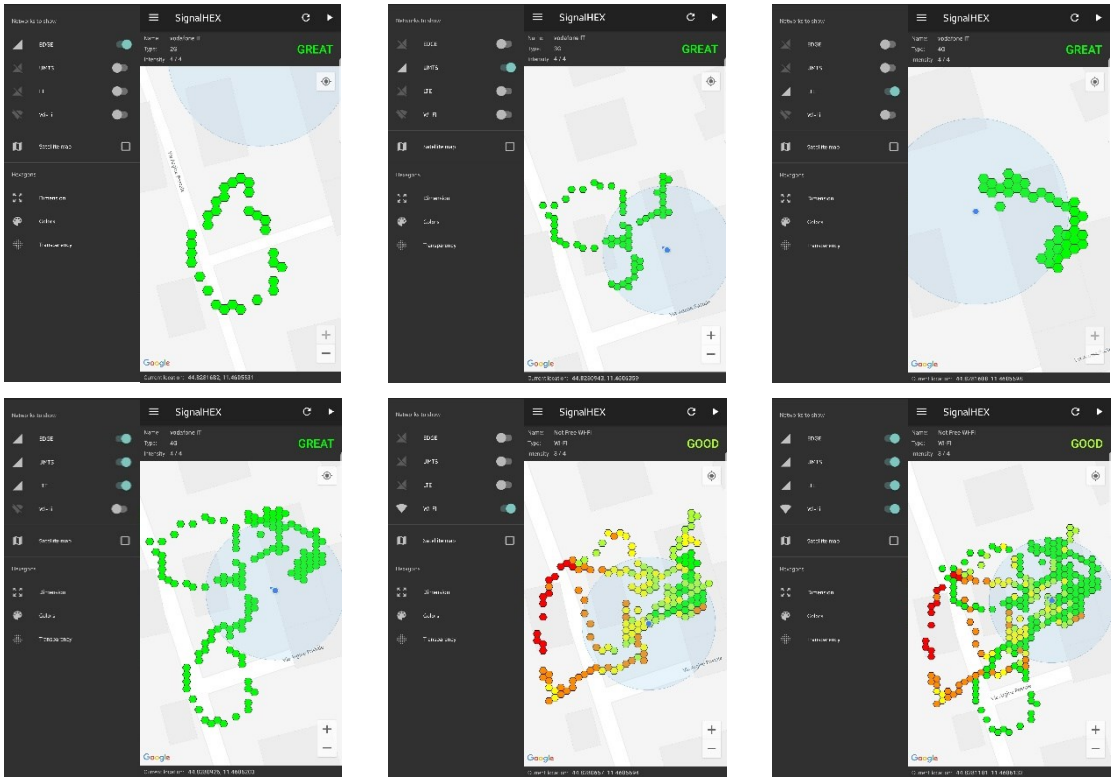
L'utente deve poter ruotare il dispositivo senza perdere i dati presenti sullo schermo, dunque ad ogni rotazione del device si devono salvare programmaticamente i poligoni e le impostazioni per poter recuperare il tutto durante la ricreazione dell'*Activity*.

Durante la messa in pausa, tutti i servizi che richiedono dati dal dispositivo devono fermarsi per evitare un *battery drain*. Devono riprendere solamente alla ripresa del focus da parte dell'app.

La rete a cui è collegato il dispositivo può cambiare da un momento all'altro, alternando in alcuni casi due tecnologie più volte al secondo. L'applicazione deve essere in grado di non interrompere la rilevazione ed ottenere sempre l'intensità del segnale della rete in uso.

Durante la creazione degli esagoni bisogna tener conto del fatto che andranno posizionati in un preciso ordine e non devono sovrapporsi l'un l'altro, dunque ho dovuto utilizzare un'apposita funzione per calcolare i punti esatti del poligono da disegnare quando si entra in una nuova area.
A causa di imprecisioni del GPS, la posizione può "saltare" da un punto all'altro, lasciando un varco più grande della dimensione di un esagono tra le coordinate attuali ed il poligono più vicino. Bisogna dunque anche "prevedere" dove disegnare l'esagono in modo da poter continuare la creazione della griglia con il giusto ordinamento.

Casi d'uso ed esempi di funzionamento tipici
L'utilizzo dell'app è principalmente destinato ad essere eseguito in ambienti esterni per via della maggior precisione del GPS. In ambienti chiusi il segnale di posizione può risultare anche di molto sbagliato, segnando valori in coordinate inesatte sulla mappa.



Possibilità di estensione
Si può estendere l'applicazione implementando la possibilità di salvare uno *snapshot* della mappa in vari formati *(.jpeg, .png, .pdf)*, esportare le impostazioni in un file esterno o estrarre le coordinate rilevate in un documento di testo, permettendo all'utente di scegliere dove salvare il tutto tra memoria interna, scheda SD, piattaforme cloud dei principali *provider*.

Si potrebbe inoltre dare la possibilità all'utente di scegliere tra varie forme geometriche da utilizzare per codificare i valori sulla mappa.

Il numero di livelli d'intensità può essere reso personalizzabile così come i singoli colori corrispondenti ad ogni livello.

La gestione del salvataggio/caricamento può essere di molto ottimizzata.

I poligoni possono essere resi cliccabili, in modo da avere informazioni su di essi selezionandoli.

Conclusioni
Durante gli anni delle scuole superiori ho avuto la possibilità di approcciare la programmazione di applicazioni Android in modo basilare, ma nonostante ciò sono stato attratto da questo ramo fin dal primo momento. Avendo lavorato per due anni nella riparazione di dispositivi mobili, il corso mi ha permesso di completare l'apprendimento dei device Android conoscendone ogni aspetto hardware e software. Essendo da anni appassionato di questo mondo ed avendo intenzione di proseguire questo tipo di programmazione, ho deciso di utilizzare il linguaggio Kotlin per lo sviluppo dell'applicazione in quanto si prospetta essere il futuro per Android. La documentazione ufficiale non è ancora pienamente aggiornata per il nuovo linguaggio, ho dunque avuto la necessità di tradurre autonomamente porzioni di codice Java. Nonostante sia la prima app sviluppata autonomamente mi ritengo pienamente soddisfatto del risultato, soprattutto a fronte delle varie difficoltà affrontate e che come ho spiegato precedentemente sono riuscito comunque a risolvere.

Per la creazione degli esagoni e le loro relative funzioni ho fatto riferimento a questa guida di Red Blob Games, adattata al linguaggio Kotlin.

L'intero codice dell'applicazione è disponibile su *GitHub* a questo indirizzo in formato open source.