

Emanuele Di Sante  
0000987616  
[emanuele.disante@studio.unibo.it](mailto:emanuele.disante@studio.unibo.it)

Matilde Mariano  
0000970476  
[matilde.mariano@studio.unibo.it](mailto:matilde.mariano@studio.unibo.it)



# TransceiverGO

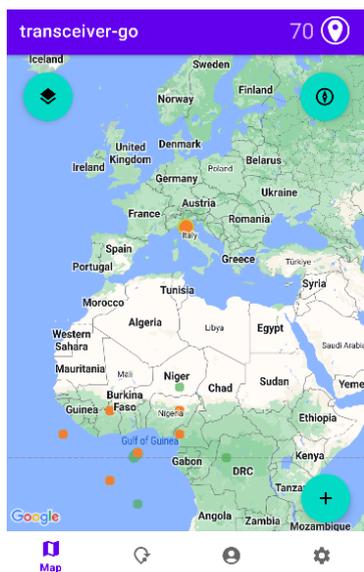
Progetto LAM AA 2022/2023

<b>Panoramica dell'app</b>	<b>3</b>
<b>Raccolta e gestione dati</b>	<b>5</b>
Struttura	5
Classe MeasurementSingleton	6
Classe NetworkSignalStrength	6
Classe NoiseStrength	7
Classe WifiSignalStrength	8
Classe Square	8
Classi Latitude e Longitude	9
Classe ConvertersForSquareDB	9
Interfaccia SquareDAO	9
Classe DatabaseImportExportUtil	10
Altre Classi	10
<b>Interfaccia grafica</b>	<b>12</b>
Struttura	12
FragmentMainMap	13
FragmentGameMap	15
FragmentAccountSettings	17
FragmentSettings	18

# Panoramica dell'app

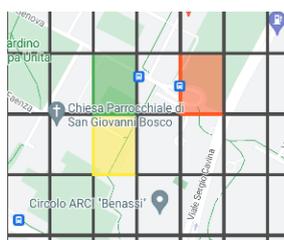
TransceiverGO è una game app sviluppata per il progetto del corso “Laboratorio di applicazioni mobili” AA 2022/2023 dell’università di Bologna.

Lo scopo dell’app è raccogliere dati relativi alla connettività cellulare, connettività WiFi e intensità del rumore, in modo da fare una mappatura di questi valori su un’area geografica. Per fare ciò, oltre a un’interfaccia classica, si ha anche una sezione di gioco che utilizza la gamification per incentivare l’utente a raccogliere dati in maniera uniforme e consistente sull’area in cui si trova.



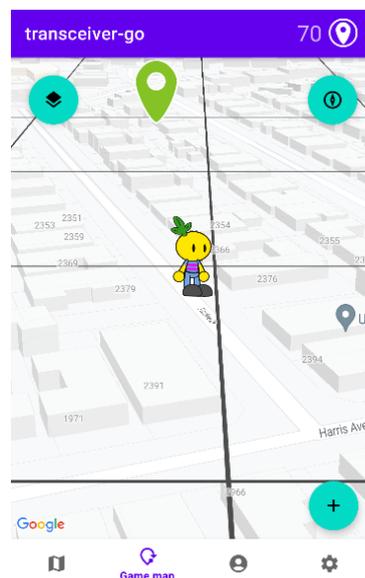
L'applicazione all'avvio presenta una schermata raffigurante una mappa ed una navbar per navigare tra le varie schermate dell'applicazione.

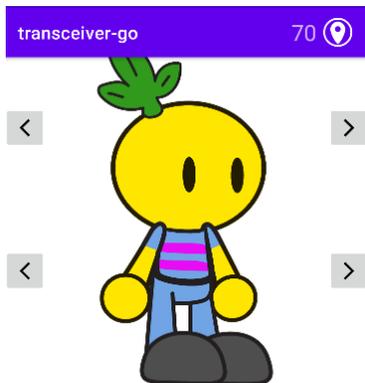
Sulla mappa sono raffigurate le varie misurazioni effettuate in modo da mostrare all'utente la distribuzione dei dati nell'area geografica.



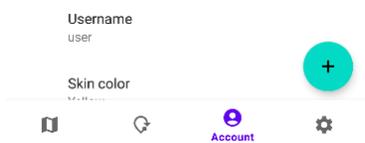
La seconda opzione nella navbar porta alla Game Map, dove l'utente ha un avatar che si muove insieme a lui sulla mappa. Nel gioco vengono generati degli obiettivi da raggiungere (indicati dal marker verde); una volta sul posto l'utente può effettuare delle misurazioni e guadagnare Transceiver Coins (📍).

Grazie all'algoritmo di generazione degli obiettivi l'utente è spinto ad esplorare zone in cui non ci sono misurazioni o in cui le misurazioni sono molto vecchie, in questo modo la raccolta dati tende ad essere più completa ed uniformemente aggiornata.

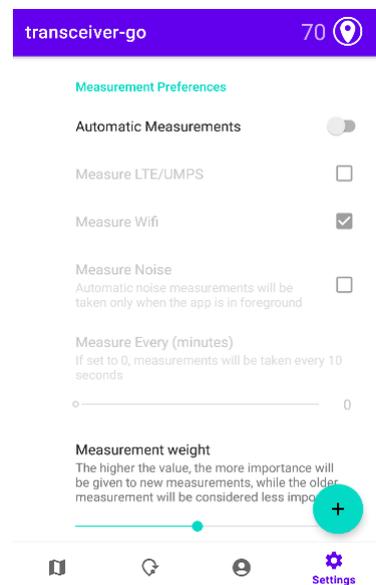




In questa sezione è possibile personalizzare l'avatar di gioco e il nome dell'utente (a cui è collegato il nome del database). Inoltre è possibile acquistare con i Transceiver Coins (👉) dei cosmetici per il proprio personaggio.



Nell'ultima sezione si ha una collezione di impostazioni e feature dell'app, come le misurazioni automatiche, l'import / export delle misurazioni e la calibrazione del microfono.

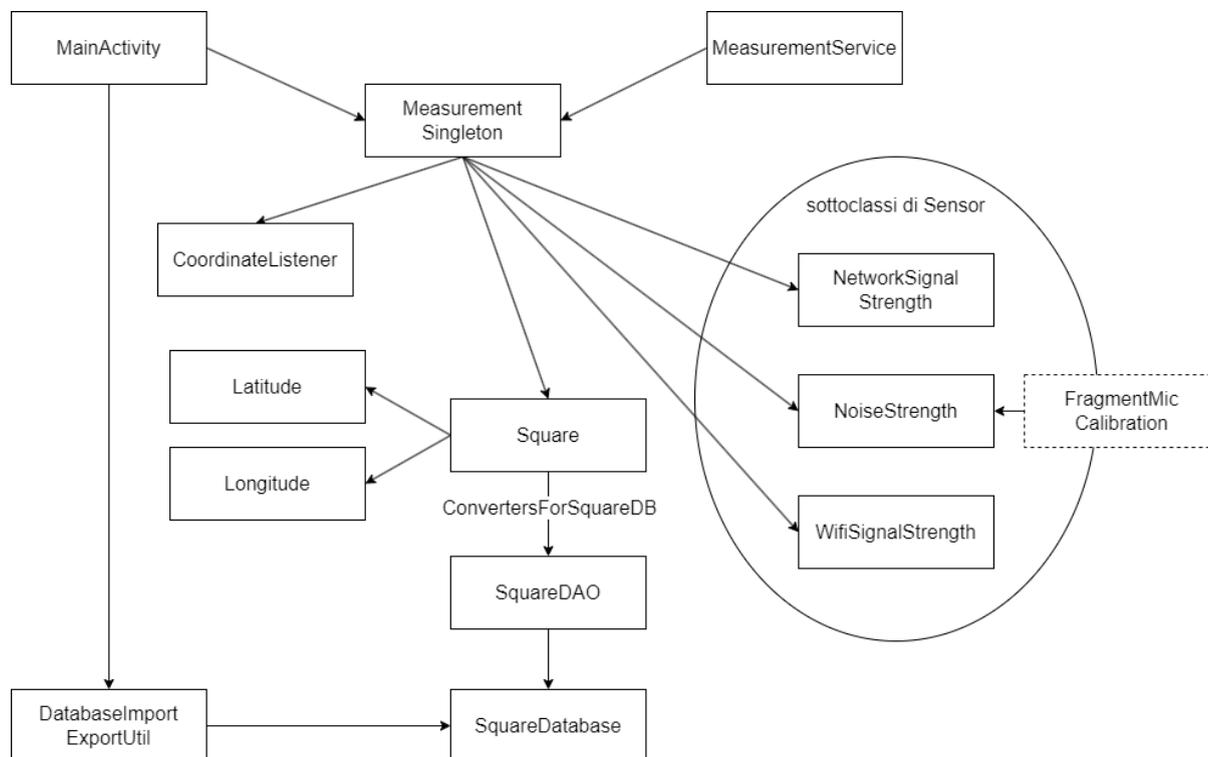


In ogni schermata è sempre presente in primo piano il pulsante delle misurazioni (+). Tramite questo pulsante è possibile accedere all'interfaccia per raccogliere in qualunque momento dati di tutti i tipi supportati (rumore, connettività cellulare e wifi).



# Raccolta e gestione dati

## Struttura



N.B. Il grafico riportato è una rappresentazione informale dei principali rapporti tra le classi relativi ad un determinato ambito; non ha alcuna pretesa di essere una documentazione precisa del codice dell'applicazione.

Il processo di raccolta dati all'interno dell'applicazione è orchestrato principalmente dalla classe `MainActivity` e dal servizio di misurazione `MeasurementService`. Questi componenti inizializzano un'istanza della classe singleton chiamata `MeasurementSingleton`, la quale agisce come punto centrale per coordinare tutte le attività di misurazione e memorizzazione dei dati nel database interno dell'applicazione, gestito da Room.

All'interno del `MeasurementSingleton`, quando una misurazione viene richiesta, vengono richiamati i metodi appropriati delle classi specializzate per ogni tipo di misurazione, ovvero `NetworkSignalStrength`, `NoiseStrength`, e `WifiSignalStrength`. Queste classi specializzate sono figlie di una classe più generica chiamata `Sensor`.

Per esempio, la classe `NetworkSignalStrength` dispone di metodi specifici come `getUmtsSignalStrength` e `getLteSignalStrength`. Analogamente, la classe `NoiseStrength` offre il metodo `startRecording`, mentre `WifiSignalStrength` fornisce `getSignalLevel`.

Dopo aver ottenuto la misurazione, il `MeasurementSingleton` chiama una funzione per inserire il nuovo valore nello `Square` attuale. Se lo `Square` non esiste ancora, viene creato.

Successivamente, il valore viene aggiunto al database degli `Square` utilizzando il `Data Access Object (DAO)` corrispondente.

La classe `Square` gioca un ruolo fondamentale nel gestire le coordinate matematicamente per derivare il centro e l'area del quadrato.

Le sue funzioni di aggiornamento includono anche la ponderazione corretta delle misurazioni in base alla preferenza definita per il loro peso (`measurement_weight`).

Nella sezione seguente vengono esplorati i metodi più importanti di ogni classe in questo processo, così come le scelte implementative effettuate opportunamente motivate.

## Classe `MeasurementSingleton`

Abbiamo deciso di utilizzare una classe singoletto così da limitare l'utilizzo delle risorse dell'applicazione, e mantenere in modo sicuro e coerente le misurazioni effettuate dai due componenti che se ne occupano (`MainActivity` e `MeasurementService`).

### Metodo `takeWifiMeasurement`

Vengono ottenute le coordinate attuali ed il livello del segnale WiFi. Successivamente, il livello del segnale viene registrato e aggiornato nel database. Dopo un breve ritardo di 500 millisecondi, la mappa viene aggiornata se l'applicazione è in foreground e viene mostrato un messaggio `Toast` per informare l'utente che la misurazione WiFi è stata effettuata.

### Metodo `updateWifiMeasurement`

Come ogni operazione sul database nell'applicazione, viene effettuata in un thread apposito anziché nel main thread.

Questa politica è stata scelta per tenere il main thread libero da operazioni computazionalmente lunghe.

Le informazioni sul segnale WiFi vengono aggiornate all'interno di un determinato `Square` nel database, che rappresenta un'area geografica specifica. Se lo `Square` non esiste, ne viene creato uno nuovo.

I restanti metodi della classe agiscono in modo analogo ai primi due già descritti, con l'unico cambiamento degno di nota essere il listener dedicato alla misurazione del rumore, che agisce dopo che la cattura del suono è terminata.

## Classe `NetworkSignalStrength`

Questa classe gestisce la raccolta delle informazioni sul segnale di rete cellulare, in particolare per le tecnologie LTE e UMTS.

### **Metodo startMonitoringSignalStrength**

Questo metodo avvia la registrazione delle variazioni del segnale di rete. Utilizza il listener PhoneStateListener per ricevere gli aggiornamenti sulla potenza del segnale.

### **Metodo getLteSignalStrength**

Restituisce il livello del segnale LTE.

Se la versione SDK è maggiore o uguale a `Build.VERSION_CODES.P`, utilizza `telephonyManager.getSignalStrength()` per ottenere il livello del segnale LTE. Altrimenti viene chiamata una funzione deprecata, dato l'applicazione supporta livelli API fino a 24.

### **Metodo getUmtsSignalStrength**

Restituisce il livello del segnale UMTS. Ha lo stesso comportamento di `getLteSignalStrength`.

### **Listener PhoneStateListener**

Il listener `PhoneStateListener` ascolta le variazioni della potenza del segnale e aggiorna i valori di `lteSignalStrength` e `umtsSignalStrength`.

## **Classe NoiseStrength**

La classe `NoiseStrength` gestisce la registrazione e l'analisi della forza del rumore ambientale.

Fornisce metodi aggiuntivi di calibrazione della scala del rumore, implementati per garantire misurazioni coerenti attraverso diversi dispositivi con qualità di registrazione differenti.

### **Metodo startRecording**

Inizia la registrazione del suono ambientale. Verifica se la registrazione è già in corso o se il permesso `RECORD_AUDIO` è stato concesso. Se non è concesso, richiede il permesso. Avvia un'istanza di `AudioRecord` per catturare l'audio dal microfono e programma un ritardo utilizzando un `Handler` per terminare la registrazione dopo un determinato periodo di tempo. Tale periodo è impostato a 3 secondi per limitare l'uso del microfono del dispositivo e risparmiare batteria.

### **Metodo stopRecording**

Termina la registrazione, calcola la media dell'ampiezza del segnale e riporta su una scala da 0 a 100. Chiama il listener di registrazione `RecordingListener` se è stato impostato.

### **Metodo calculateAverageAmplitude**

Calcola l'ampiezza media del segnale audio registrato.

### **Metodo calibrateTo100Scale**

Scala l'ampiezza del segnale su una scala da 0 (silenzio) a 100 (Applauso). Può superare il rumore dell'applauso.

Gli ulteriori metodi presenti gestiscono la calibrazione del microfono (utilizzando i threshold per il silenzio e per il battito di mani) lavorando in maniera analoga ai metodi descritti sopra.

## Classe `WifiSignalStrength`

La classe `WifiSignalStrength` gestisce la rilevazione del livello e dell'intensità del segnale Wi-Fi.

### Metodo `getSignalStrength`

Restituisce l'intensità del segnale Wi-Fi in dBm. Controlla se il Wi-Fi è abilitato, quindi ottiene le informazioni sulla connessione Wi-Fi e restituisce l'intensità del segnale.

### Metodo `getSignalLevel`

Restituisce il livello del segnale Wi-Fi, calcolato su una scala da 0 a 4.

## Classe `Square`

La classe `Square` rappresenta un quadrato sulla mappa con attributi associati come coordinate, intensità del segnale di rete, intensità del segnale Wi-Fi e intensità del rumore. La classe è progettata per gestire e visualizzare i dati relativi alle misurazioni di segnali in una specifica area geografica rappresentata dal quadrato sulla mappa.

### Attributi Principali

- `SIDE_LENGTH`: Costante statica che rappresenta la lunghezza del lato del quadrato.
- `coordinates`: Identificatore unico del quadrato composto dalle coordinate X e Y.
- `longitude, latitude`: Oggetti che rappresentano la longitudine e la latitudine del quadrato.
- `network, wifi, noise`: Valori che indicano rispettivamente l'intensità del segnale di rete, del segnale Wi-Fi e del rumore.
- `lastNetworkMeasurement, lastWifiMeasurement, lastNoiseMeasurement`: Timestamp dell'ultima misurazione effettuata rispettivamente per il segnale di rete, il segnale Wi-Fi e il rumore.

### Metodi Principali

- `drawTile`: Disegna un quadrato sulla mappa con i colori indicati.
- `drawNoiseTile, drawWifiTile, drawNetworkTile, drawEmptyTile`: Disegnano il quadrato sulla mappa con colori diversi in base ai livelli di intensità.
- `updateNetwork, updateWifi, updateNoise`: Aggiornano i valori di intensità applicando un peso alla nuova misurazione.
- `convertMeasurementWeightToDecimal`: Eseguono operazioni di calibrazione sui valori di intensità.

Per gestire la ripetizione delle misurazioni dello stesso tipo nello stesso Square è stato deciso di non salvare ogni singolo valore nel database, per evitare di consumare troppa memoria del dispositivo e di avere operazioni troppo costose di recupero dei dati. Invece è stato scelto di fare una media sbilanciata tra il valore delle misurazioni precedenti e la nuova, con il fattore di sbilanciamento (`measurementWeight`) è un valore tra 0 e 1 scelto dall'utente.

In pratica la formula assume questo aspetto:

```
measurement(0)      = newMeasurement
measurement(n + 1) = newMeasurement * measurementWeight + measurement(n) * (1 -
                    measurementWeight)
```

## Classi Latitude e Longitude

La classe `Latitude` rappresenta la latitudine e fornisce metodi per eseguire operazioni matematiche su questo valore.

La classe `Longitude` svolge una funzione analoga alla classe `Latitude` ma per la longitudine.

Le classi sono progettate per garantire che i valori di latitudine e longitudine siano sempre validi e rientrino nei range previsti per evitare problemi durante la rappresentazione su una mappa.

## Classe ConvertersForSquareDB

La classe `ConvertersForSquareDB` contiene metodi annotati con `@TypeConverter` utilizzati per convertire oggetti delle classi `Longitude` e `Latitude` in valori primitivi (`double`) e viceversa.

Questi metodi sono necessari quando si utilizzano tali oggetti all'interno di un database Room in Android.

## Interfaccia SquareDAO

L'interfaccia `SquareDAO` definisce le operazioni di accesso ai dati (DAO) per la classe `Square`. Sono presenti le operazioni di base quali quelle di inserimento, aggiornamento, eliminazione e query per l'entità `Square`.

Alcune di queste operazioni non sono utilizzate nel prodotto finale, ma sono state lasciate in quanto potrebbero esserlo in una futura versione dell'applicazione.

Le operazioni del DAO vengono eseguite in modo asincrono per evitare il blocco dell'interfaccia utente durante l'accesso al database.

## Classe DatabaseImportExportUtil

Questa classe fornisce un insieme di utility per la gestione e la manipolazione dei database nell'applicazione a livello filesystem, consentendo operazioni come l'esportazione, l'importazione, il cambio nome e l'eliminazione dei database.

### Metodo shareDatabase

Esporta il database e lo condivide con un'app scelta dall'utente. Il metodo avvia un'attività per condividere il file del database tramite un intent.

### Metodo changeDatabaseName

Chiude il vecchio database, rinomina il file del database nel sistema di archiviazione interno dell'app e riapre il database con il nuovo nome.

### Metodi di Importazione (importFileToDatabaseDirectory, handleImportFileResult, importSelectedFile)

- `importFileToDatabaseDirectory`  
Crea un intent per aprire un selettore di documenti ed importare un file nella directory dei database dell'app.
- `handleImportFileResult`  
Gestisce il risultato dell'attività di selezione del documento. Estrae l'URI del documento selezionato e chiama il metodo `importSelectedFile`.
- `importSelectedFile`  
Importa il file selezionato nella directory dei database dell'app, sovrascrivendo il file esistente se presente. Infine copia il contenuto del file selezionato nel nuovo file del database.

### Metodo deleteDatabase

Chiude il database e ne elimina il file dal sistema di archiviazione interno dell'app.

## Altre Classi

La classe `SquareDatabase` rappresenta il database Room per la gestione dei dati relativi agli oggetti `Square`.

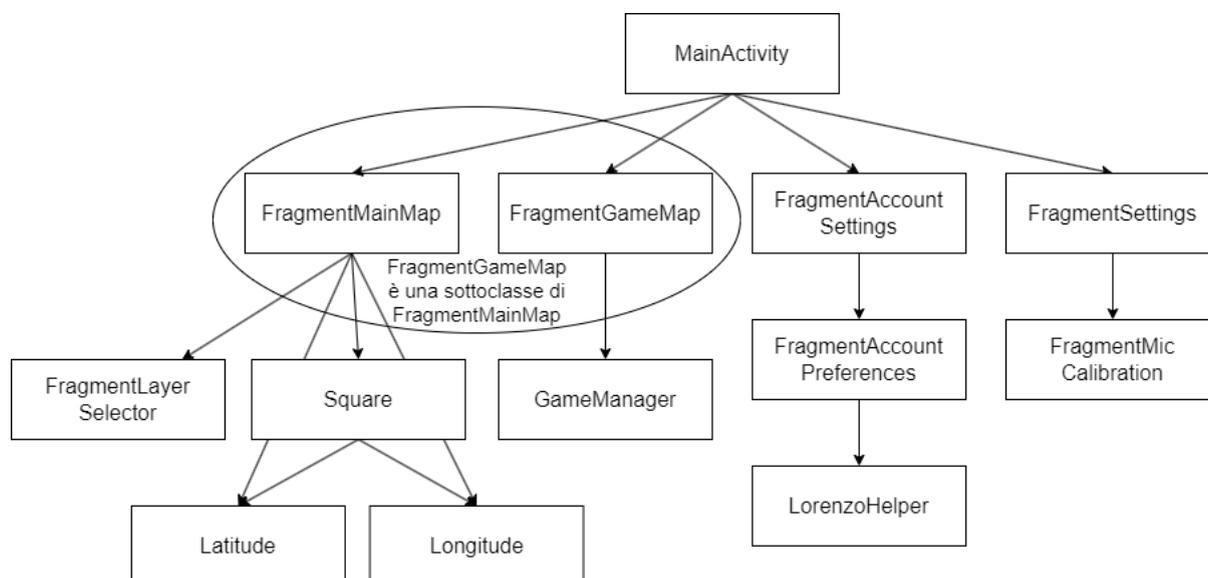
La classe `SquareMigration` estende la classe `Migration` di Room e implementa il metodo `migrate` che viene chiamato quando si effettua una migrazione del database.

Questa migrazione è stata implementata durante le prime fasi di costruzione del database e contiene migrazioni non più utilizzate, tuttavia importanti a scopo didattico per lo sviluppo di migrazioni funzionanti con Room/SQLite.

In sintesi, la struttura dell'applicazione garantisce un approccio organizzato e modulare per la raccolta, gestione e memorizzazione dei dati di misurazione. L'utilizzo del pattern singleton facilita la coordinazione di queste attività da diverse parti dell'applicazione, garantendo coerenza e facilità di manutenzione.

# Interfaccia grafica

## Struttura



N.B. Il grafico riportato è una rappresentazione informale dei principali rapporti tra le classi relativi ad un determinato ambito; non ha alcuna pretesa di essere una documentazione precisa del codice dell'applicazione.

L'intera applicazione è gestita tramite una sola activity chiamata MainActivity.

MainActivity, grazie all'ausilio di un menù di navigazione (`bottom_nav_menu`), attiva i fragment che contengono la porzione dell'interfaccia con cui l'utente desidera interagire.

Questi quattro fragment sono:

- FragmentMainMap: contenente la mappa dove vengono rappresentati i dati raccolti.
- FragmentGameMap: contenente un piccolo gioco per incoraggiare l'utente a raccogliere dati in maniera uniforme.
- FragmentAccountSettings: sezione dove è possibile modificare il proprio nome utente ed il proprio avatar, nonché comprare vari accessori per quest'ultimo.
- FragmentSettings: contenente le impostazioni dell'applicazione.



In basso a destra si trova un FloatingActionButton, se toccato si espande in tre altri bottoni che permettono in qualsiasi momento di effettuare una misurazione dei parametri desiderati.

## FragmentMainMap

FragmentMainMap contiene una mappa fornita da Google Maps SDK for Android e due pulsanti per agevolare la navigazione.

Il pulsante più a destra (indicato con ) è chiamato orientationButton e serve a riportare l'orientamento della mappa verso nord nel caso sia stata ruotata.

Per fare ciò viene reso invisibile il pulsante già presente nella mappa con questa funzione, quando orientationButton viene premuto viene simulato un tocco su tale pulsante nascosto.

Inoltre la rotazione dell'icona di orientationButton indica sempre in che direzione si trova il nord.

Il pulsante a sinistra (indicato con ) invece è layerButton ed un tocco su di esso sostituisce FragmentMainMap con FragmentLayerSelector. Quest'ultimo fragment permette di scegliere la tipologia di dati da visualizzare sulla mappa, scegliere da quali database leggere i dati ed eliminare i database che non si desidera più mantenere.

In particolare, per visualizzare la lista di database, per ognuno di essi viene creata una CheckBoxPreference popolata sia con una serie di attributi già precedentemente definiti in un file XML, sia con degli attributi che dipendono dalla singola istanza:

```
for (String s : dbList) {
    if (!s.endsWith("-wal") && !s.endsWith("-shm")) {
        Preference pref = new CheckBoxPreference(requireContext(), attr);
        pref.setKey(s);
        pref.setTitle(s);
        category.addPreference(pref);
    }
}
```

Dove dbList è la lista di nomi dei database, category è la PreferenceCategory dentro la quale collocare le CheckBoxPreference ed attr è il modello da cui vengono presi gli attributi.

attr è definito come:

```
XmlPullParser parser = getResources().getXml(R.xml.layer_preference_list);
try {
    parser.next();
    parser.nextTag();
} catch (Exception e) {
```



```

    e.printStackTrace();
}
AttributeSet attr = Xml.asAttributeSet(parser);

```

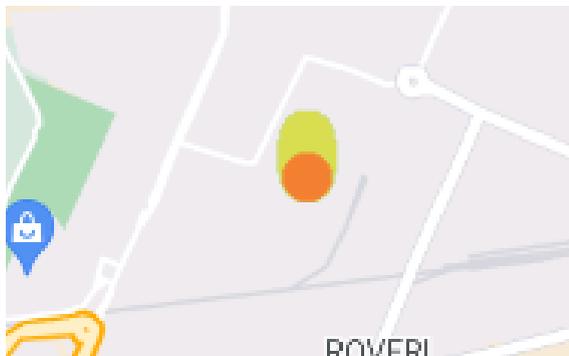
E layer\_preference\_list contiene:

```

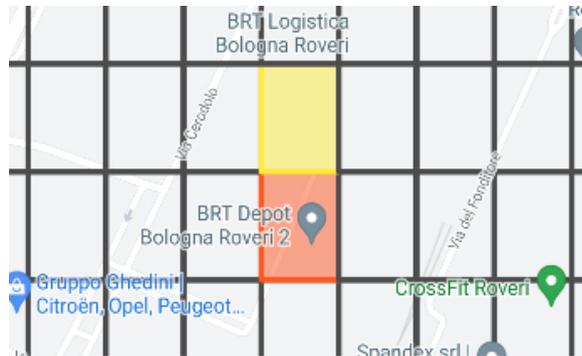
<CheckBoxPreference
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:key="defaultKey"
    android:title="defaultKey"
    app:defaultValue="false"/>

```

La mappa ha due modalità di visualizzazione delle informazioni: tramite heatmap, per dare un'idea approssimativa della distribuzione dei valori nell'area con bassi livelli di zoom, e tramite griglia, dove invece quando si utilizza uno zoom elevato vengono mostrate in maniera più precisa le aree in cui l'app divide il territorio ed i relativi valori.



Heatmap



Griglia

Il disegno delle heatmap è gestito dal metodo `retrieveSquaresAndDrawHeatmap` mentre quello della griglia da `retrieveAndDrawSquares`.

In `retrieveSquaresAndDrawHeatmap` per la rappresentazione dei dati sulla mappa viene utilizzata Google Maps Android Heatmap Utility, la quale però ha due grossi problemi che sembrerebbero renderla inutilizzabile:

- La gestione dei colori è imprecisa e molte configurazioni (specialmente quando si hanno pochi dati) sono rappresentate in maniera molto confusa.
- Diminuendo lo zoom i valori dei dati adiacenti tra loro vengono sommati; questo significa che un'ampia area riempita da dati che mostrano bassa intensità, se vista sufficientemente da lontano, viene mostrata come una piccola area con alta intensità.

Per ovviare a queste problematiche la utility viene utilizzata in maniera poco ortodossa, creando tre heatmap monocolori per ognuna delle tre fasce di intensità (bassa, media ed alta) e sovrapponendole tra loro.

Ogni heatmap comprende i valori della propria fascia e di quelle inferiori, in questo modo l'area su cui sono state fatte le misurazioni viene coperta in maniera uniforme e senza buchi. Il raggio dell'area coperta da un singolo punto dell'heatmap invece viene calcolato a runtime basandosi sul livello di zoom, in modo da non lasciare mai spazi vuoti tra due misurazioni adiacenti (ciò è fatto dal metodo `calculateProperHeatmapRadiusBasedOnZoom`).

Ciò però genera un ulteriore problema: con livelli di zoom sufficientemente alti (e di conseguenza raggi sufficientemente ampi) il rendering delle heatmap consuma troppe risorse e causa un crash dell'applicazione.

Questa limitazione in realtà non causa grosse difficoltà in quanto oltre un certo livello di zoom la rappresentazione dei dati viene affidata alla griglia.

`retrieveAndDrawSquares` invece per la rappresentazione dei dati si poggia sui metodi `drawNetworkTile`, `drawNoiseTile`, `drawWifiTile` e `drawEmptyTile` di `Square`. L'unica criticità che questa funzione ha è che con bassi livelli di zoom la quantità di `Square` da renderizzare diventa imponente e causa lunghe attese sul UI Thread, ma anche in questo caso il problema può essere ignorato in quanto con bassi livelli di zoom subentrano le heatmap.

In molti punti del codice (per esempio nel recupero dei dati dai database o nel disegno degli `Square`) si può osservare che è stato necessario dividere la terra in due emisferi e trattarli separatamente.

Questo, a fronte di qualche funzione in più, ha permesso di evitare la complicata gestione del punto in cui la longitudine passa da -180 a +180 e viceversa.

Le funzioni influenzate da questa scelta sono facilmente identificabili in quanto contengono quasi sempre all'interno del loro nome l'emisfero di riferimento (che, in una visione europocentrica del mondo, sono chiamati Western ed Eastern Hemisphere), per esempio si hanno `truncateLongitudeToWesternHemisphere` oppure `truncateLongitudeToEasternHemisphere`.

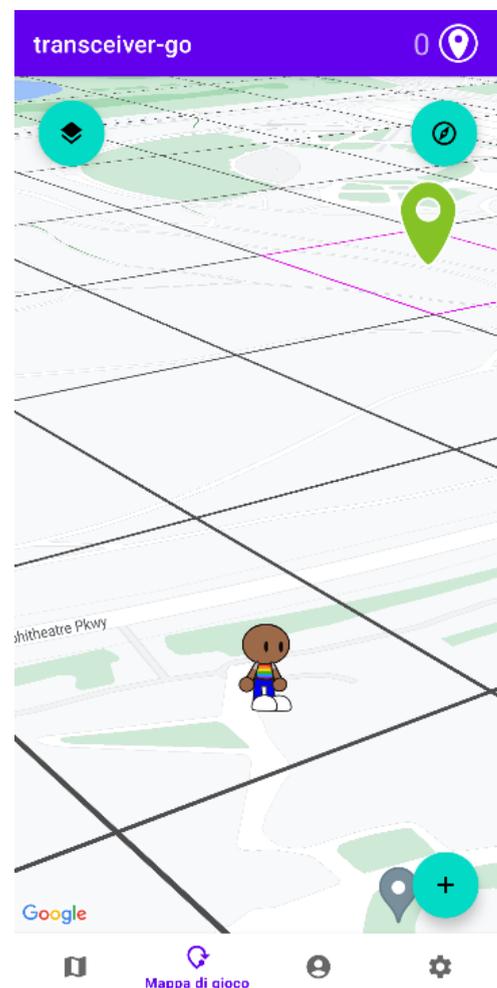
## FragmentGameMap

`FragmentGameMap` è una classe figlia di `FragmentMainMap`, di conseguenza eredita ed estende le sue funzionalità dando loro uno scopo molto diverso.

Infatti lo scopo dell'interfaccia di questo fragment è utilizzare la gamification per motivare l'utente a raccogliere in modo uniforme e completo i dati sulla zona in cui si trova, ispirandosi nel funzionamento a giochi come *Pokemon-GO* e *Randonautica*.

Rispetto alla classe genitore a schermo si trovano due elementi in più: un avatar che rappresenta la posizione del giocatore ed un marker che rappresenta il luogo in cui andare per ottenere la ricompensa.

La posizione del giocatore viene aggiornata in tempo reale grazie ad un `CoordinateListener`, il quale fa in modo che l'avatar si muova da una posizione all'altra tramite una serie di animazioni.



La posizione del marker invece viene generata dal metodo `generateNewTarget` della classe `GameManager`, prendendo un singolo `Square` casualmente tra i 10 attualmente disegnati a schermo con il valore più vecchio.

L'algoritmo utilizzato risulta quindi:

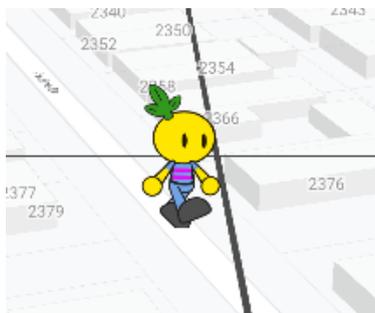
```
public Square generateNewTarget(List<Square> loadedSquares){
    Collections.shuffle(loadedSquares); // randomizes the order of
        LoadedSquares, bringing more entropy to the generation

    loadedSquares.sort(tipoDiDatoVisualizzato);

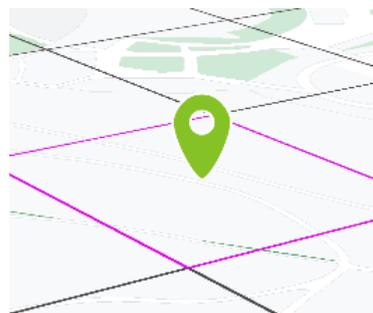
    int bound = Math.min(poolSize, loadedSquares.size());
    Random rand = new Random();
    int indexOfTarget = rand.nextInt(bound); // generates a random number
        between 0 and bound - 1

    currentTarget = loadedSquares.get(indexOfTarget);
    return currentTarget;
}
```

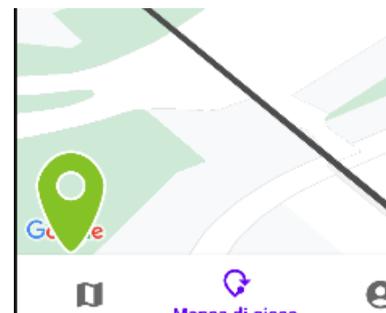
Sia l'avatar che il marker ad una prima analisi paiono essere rappresentati al meglio da un Marker di Maps SDK, ma questo impedirebbe all'avatar di essere personalizzabile dall'utente e di avere un'animazione di camminata, mentre impedirebbe al marker di rimanere sempre visibile a schermo per indicare al giocatore in che direzione andare.



Frame dell'animazione di camminata



Marker che punta l'oggetto sulla mappa



Marker che rimane sul bordo quando l'oggetto esce dallo schermo

Per questi motivi è stato necessario utilizzare al posto dei marker delle `ImageView` che, grazie a `map.getProjection().toScreenLocation(posizioneDelloGggetto)`, vengono posizionate sul punto dello schermo a cui corrisponde la loro posizione sulla mappa. Però il metodo `toScreenLocation` andando molto facilmente in overflow si è dimostrata poco affidabile, di conseguenza è stato necessario costruire il metodo `checkForPositionOnScreenOverflow` che si occupa di controllare la validità dei valori da essa ritornati.

Il marker inoltre è interattivo: se il giocatore si trova nel suo stesso `Square` interagendovi può eseguire delle misurazioni, riscuotere la ricompensa e far generare una nuova posizione del puntatore; se invece è troppo lontano toccandolo riceve un messaggio di errore e può decidere di rigenerare la posizione nel caso la precedente sia inaccessibile.

Per quanto riguarda le interazioni con la mappa le principali differenze sono:

- la camera non è più perpendicolare alla mappa, ma è sempre il più inclinata possibile verso l'orizzonte, in modo da rendere la visuale di gioco il più simile possibile a ciò che vede il giocatore nella realtà permettendogli meglio di orientarsi.
- `orientationButton` (📍) punta sempre verso nord, ma premerlo questa volta fa posizionare la camera sull'avatar del giocatore, orientandola nella direzione in cui esso sta guardando.

## FragmentAccountSettings

`FragmentAccountSetting` è una schermata contenente tutte le opzioni di personalizzazione del proprio account, questo comprende il nome utente e l'aspetto dell'avatar.

Il fragment si divide in due parti: nella parte superiore si trova una specie di camerino/negoziato, dove si possono cambiare i vestiti ed il cappello al proprio avatar ed acquistarne altri con le monete guadagnate giocando in `FragmentGameMap`; nella parte inferiore invece si hanno due campi dove inserire il proprio nome utente ed il colore della pelle dell'avatar.

In questo fragment, così come in `FragmentGameMap`, l'avatar del giocatore non è composto da una sola `ImageView`, bensì da tre.

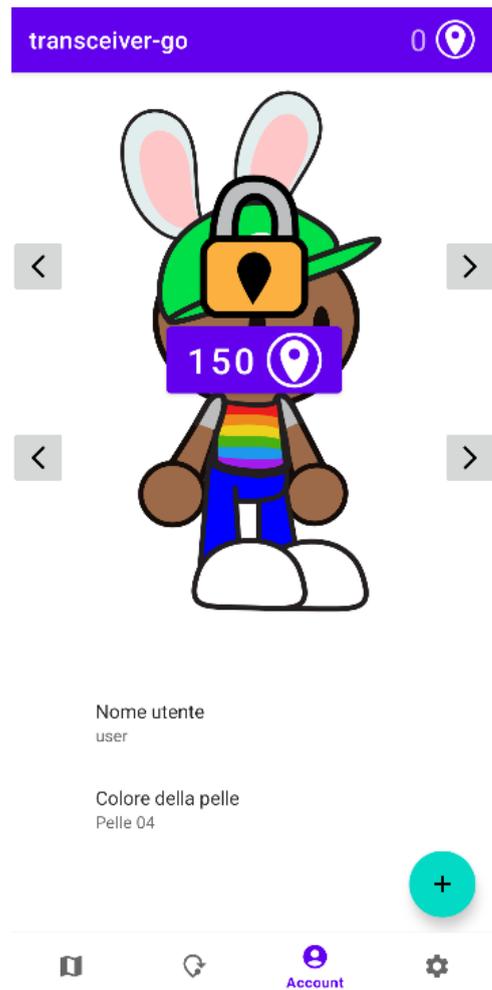
Questo permette di personalizzare in maniera abbastanza espressiva il proprio personaggio, assegnandogli separatamente un colore della pelle, un copricapo e dei vestiti.

La composizione delle immagini per la parte di gioco è affidata alla classe `LorenzoHelper`, in particolare a `buildLorenzoIdleFromPreferences` e `buildLorenzoWalkFromPreferences`.

I vestiti ed i cappelli sono salvati in `array.xml` ed i pulsanti ai lati cicliano tra di essi caricando di volta in volta nella `ImageView` relativa il capo selezionato.

La gestione del colore della pelle invece è più complessa in quanto non è possibile fare un array di colori in `array.xml`; di conseguenza in quel file è salvato un array contenente i nomi dei colori (dichiarati in `colors.xml`) ed il metodo `nameToColor` di `LorenzoHelper` li rimappa al corrispettivo colore.

```
public static int nameToColor(Context context, String name) {
    try {
        return context.getColor(context.getResources().getIdentifier(
            name, "color", context.getPackageName()));
    } catch (Exception e) {
```



```
        return 0;
    }
}
```

Poi sempre i metodi di LorenzoHelper `lorenzoIdleSkinBuilder` e `lorenzoWalkSkinBuilder` applicano un filtro all'immagine della pelle rendendola del colore desiderato.

Gli articoli acquistati e la configurazione attuale dell'aspetto dell'avatar vengono salvati nelle `DefaultSharedPreferences`.

La stessa cosa vale per il nome utente, il quale però ha una criticità: siccome è utilizzato anche come nome del database dove vengono salvati i dati personali, se esiste già un database importato con lo stesso nome si perdono tutti i dati.

Di conseguenza prima di rinominare il database viene controllato se ne esistono altri con quel nome e nel caso il nome viene ripristinato a quello precedente generando un messaggio di errore.

## FragmentSettings

FragmentSettings è un fragment contenente le restanti impostazioni dell'applicazione.

È diviso in tre sottocategorie:

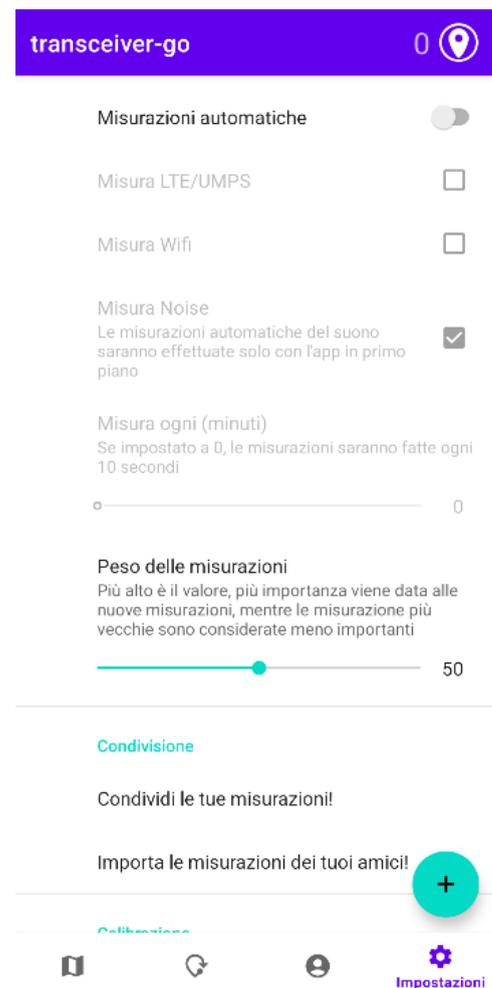
- Preferenze delle misurazioni
- Condivisione
- Calibrazione

In Preferenze delle misurazioni si ha un'opzione per avviare il service che si occupa di prendere misurazioni in background, il quale all'attivazione effettua tutti i controlli necessari affinché il service possa correttamente prendere le misurazioni (permessi e ottimizzazione della batteria).

Legata a questa opzione si ha la possibilità di selezionare quali misurazioni vengono effettuate e con che frequenza.

Successivamente vi è uno slider per configurare il peso delle misurazioni, che altera l'importanza delle nuove misurazioni rispetto a quelle già prese, come descritto nella sezione Raccolta e gestione dati.

In Condivisione vi sono due opzioni che lanciano due intent, rispettivamente per l'importazione ed esportazione del database dump personale



dell'utente, così da poter condividere i propri progressi con altri utenti.

In Calibrazione vi è un'unica opzione che carica il fragment `FragmentMicCalibration`.

Questo fragment permette all'utente di accedere alle funzioni di calibrazione del microfono provviste nell'applicazione.

## Calibrazione del microfono

Come funziona:

La calibrazione del microfono richiede solo pochi secondi e serve ad assicurare che le tue misurazioni del rumore siano simili a quelle di tutti gli altri (e non dipendenti dalla sensibilità del microfono)! Come spiegato più in basso devi solamente impostare due threshold: uno corrispondente al silenzio ed uno corrispondente al battito delle tue mani.

Posizionati in un ambiente silenzioso, premi il pulsante ed attendi tre secondi in silenzio. Il rumore di fondo del microfono viene calcolato in dBm facendo la media del rumore percepito in quei tre secondi.

0 dBm

IMPOSTA THRESHOLD DEL SILENZIO

Posizionati con le mani a circa 30cm dal telefono, premi il pulsante e sbatti le mani almeno una volta nell'arco di 3 secondi. Il relativo Threshold viene calcolato in dBm ed è basato sul massimo rumore percepito in quei tre secondi.

100 dBm

IMPOSTA THRESHOLD DELL'APPLAUSO