Alma Mater Studiorum – Università di Bologna

scuola di scienze
corso di laurea in informatica

WeWheelRockU:

un'applicazione partecipativa per disabili a Bologna

Relatore:
Prof. Maurizio Gabbrielli

Presentata da:
Manuel Lando

II Sessione
Anno Accademico 2012/2013

Indice

Introduzione

WeWheelRockU è un'idea sviluppata nell'ambito delle applicazioni mobili partecipative. L'obiettivo è quello di fornire al disabile un servizio che possa aiutarlo ad affrontare le difficoltà che facilmente può incontrare all'interno di una città.
É evidente quanto possa essere difficile, per un disabile, riuscire a muoversi autonomamente all'interno di un contesto urbano; questo può essere causato dal fatto che abbiamo dei centri storici non adeguati ma, in molti casi, anche dall'indifferenza generale o dalla disattenzione, sia da parte delle amministrazioni locali, sia da parte delle persone comuni prive di disabilità.
Secondo l'ISTAT i disabili in Italia rappresentano il 5% della popolazione; considerato il fatto che la maggior parte di essi si concentra in città, per una vicinanza maggiore ai servizi, la percentuale nei centri urbani aumenta. Un ulteriore fattore da considerare è il fatto che la disabilità è una condizione strettamente legata all'invecchiamento della persona e questo, tenendo presente che in Italia la media d'età della popolazione è in continuo aumento, porta a stimare per il 2035 un aumento del 65-75% delle persone con disabilità. Il progetto vuole contribuire al miglioramento di questa situazione, cercando di assistere le persone disabili all'interno dei centri urbani e dando loro una maggiore autonomia e sicurezza. Il tentativo è quello di mettere in comunicazione diretta le tre parti sopra citate: il disabile, quale diretto interessato e persona attiva nello sviluppo, l'amministrazione locale quale utilizzatore dell'applicazione ai fini di un miglioramento dei servizi cittadini, e tutte le altre persone che, anche se non in qualità di dirette interessate, potranno contribuire con uno sforzo minimo all'ampliamento delle informazioni a disposizione.
Ciò che preme ricordare è che l'essenza dell'applicazione vuole essere la partecipazione, e il suo obiettivo finale è quello di portare ad una maggiore attenzione, da parte di tutti, nei confronti dei disabili. Non dimentichiamo che questo vale non solo per persone in sedia a rotelle, ma anche per chi ha altre particolari necessità e per chi gira in città con passeggini. Paradossalmente, la meta finale di tale applicazione è quella di arrivare al punto in cui il suo utilizzo non sarà

più necessario.

Il lavoro è stato pensato per la città di Bologna, ma è facilmente applicabile in tutti i contesti urbani italiani.

Il progetto è realizzato in collaborazione con il collega Giuseppe Balistreri. La parte che io tratterò riguarda l'accessibilità della città, in particolare delle strade del centro di Bologna, mentre Giuseppe tratterà l'accessibilità degli edifici pubblici. L'applicazione, nella parte descritta in seguito, consiste di tre sezioni principali: la prima funzionalità è quella di rendering della mappa di Bologna e di tutte le informazioni a disposizione nel database dell'applicazione; particolarità di questa funzionalità è la possibilità di accedere alle mappe in modalità offline; la seconda funzionalità è molto importante e rappresenta la componente partecipativa dell'applicazione, permettendo agli utenti di inserire nella mappa simboli di diverso tipo, utili all'ampliamento del database e alla condivisione; i simboli utilizzati vogliono indicare quelle che possono essere le zone più facilmente attravesabili da un disabile, quelle che presentano alcuni problemi, e le zone difficilmente attraversabili o raggiungibili; infine è presente una funzionalità di navigazione, che utilizza le informazioni a disposizione per il calcolo di percorsi adatti al disabile; quest'ultima rappresenta una soluzione sperimentale al problema.

Dopo una breve introduzione sull'ambiente di sviluppo e sugli strumenti utilizzati, il terzo capitolo passa alla descrizione delle caratteristiche generali dell'applicazione, all'analisi e alla classificazione dei dati da raccogliere, nonchè alle strutture utili allo scopo finale del progetto. Nel capitolo successivo si trattano le funzionalità principali del lavoro, cioè il rendering di mappe offline e la loro consultazione. Nel quinto capitolo si approfondirà l'inserimento di nuovi dati da parte degli utenti, per permettere l'ampliamento delle informazioni utili all'applicazione, mentre il sesto capitolo si concentrerà su una possibile soluzione per lo sviluppo di una funzionalità di navigazione. Nel settimo capitolo si parlerà di un'esperienza di simulazione legata a questo progetto, atta a verificare i possibili miglioramenti portati da tale applicazione, anche in presenza di ostacoli temporanei e con diversi gradi di durata, mentre nella sezione finale si parlerà dei possibili futuri sviluppi del software.

2 Strumenti utilizzati

Analizzando le diverse possibilità si è giunti a scegliere come target per la nostra applicazione Android OS. La motivazione di questa scelta è data dal fatto che Android OS è, al momento, il sistema mobile più diffuso al mondo e, in termini economici, il più accessibile. In aggiunta a queste motivazioni, vi è quella dell'utilizzo di una libreria molto utile allo scopo del progetto, ovvero Mapsforge[1], essa permette, attraverso l'override dei principali metodi di Google Maps[2], l'utilizzo di mappe offline provenienti dal progetto collaborativo OpenStreetMap (OSM)[3].

2.1 Android SDK

Android SDK fornisce gli strumenti che servono per creare applicazioni Android. Gli strumenti necessari per poter sviluppare tali applicazioni sono l'IDE Eclipse, Java SE, Android SDK 3.x e ADT[4] Plugin per Eclipse. L'ultimo strumento elencato è di vitale importanza per uno sviluppatore di applicazioni Android. Esso infatti, dà la possibilità di creare degli AVD[5] e definire dei target con precise caratteristiche. Una volta creato un AVD è possibile lanciare l'emulatore Android, incluso in Android SDK, e riprodurre la maggior parte dei gesti che si possono fare con un normale dispositivo. Lo strumento più utilizzato che si trova nella sessione Plattform Tools dell'Android SDK è sicuramente *adb* che permette l'esecuzione di una shell con cui poter effettuare le principali operazioni su file, solitamente disponibili in Android.

[1] Progetto disponibile su http://code.google.com/p/mapsforge/
[2] Google Maps per Android su http://www.google.it/mobile/maps/
[3] Progetto OSM disponibile su http://www.openstreetmap.org
[4] Android Development Toolkit
[5] Android Virtual Device

Un'altra libreria molto importante, inclusa nel progetto, è SQLite[6]; si tratta di uno strumento utile alla creazione e gestione di DBMS[7] relazionali.

2.2 Mapsforge e OpenStreetMap

Nella fase di analisi e di identificazione delle specifiche del progetto si è cercato di trovare uno strumento che potesse fornire delle semplici e veloci funzionalità di personalizzazione e rendering di mappe cittadine. Sin da subito, si è individuato nel progetto OpenStreetMap un ottimo punto di partenza per l'utilizzo di queste mappe; si tratta, infatti, di un progetto di creazione e condivisione di dati cartografici liberi e gratuiti, senza alcun tipo di restrizione legale o tecnica.
Rivolgendo l'attenzione al target finale dell'applicazione, ci si è in seguito indirizzati verso il progetto Mapsforge della Freie Universität di Berlino che si basa su OpenStreetMap e offre la possibilità di renderizzare mappe ad-hoc su dispositivi Android, tramite una libreria libera e open source. Un ulteriore punto a favore di questa scelta è la possibilità di salvare i dati cartografici in locale e, di conseguenza, di poterli consultare anche in modalità offline. Il progetto Mapsforge non si vuole limitare solo ad Android, infatti in futuro, sarà disponibile anche per altre piattaforme. Nei capitoli seguenti vedremo quali sono le potenzialità di tale strumento.

[6] Libreria software disponibile su http://www.sqlite.org/
[7] Data Base Management System

3 WeWheelRockU nel dettaglio

L'applicazione è stata suddivisa in tre principali sessioni, divise per funzionalità; possiamo definirle in questo modo:

- **Consultazione della mappa dell'area urbana di Bologna:** la mappa è caratterizzata da indicazioni speciali per le persone disabili;
- **Inserimento di nuovi dati nella mappa:** si tratta della funzionalità partecipativa che permette l'aggiornamento del database;
- **Calcolo dei percorsi:** la funzionalità, basandosi sul database presente, calcola i percorsi più adatti al tipo di utente che ne fa richiesta.

Occorre ora analizzare nel dettaglio queste funzionalità, per definire le caratteristiche specifiche dell'applicazione in questione. Seguirà, per ogni funzionalità, una sessione d'analisi dei requisiti e dei risultati applicativi che si vogliono raggiungere.

3.1 Consultazione della mappa

Un fattore di fondamentale importanza è la resa molto semplice e chiara della mappa di Bologna, su cui poter lavorare e aggiungere le nostre informazioni in modo chiaro.
La prima caratteristica che si è pensato di inserire nell'applicazione è l'indicazione di tutti i segmenti di strada percorribili da un disabile. Naturalmente, si deve tenere in considerazione il fatto che vi siano diverse caratteristiche che possono portare a dare una valutazione di diverso tipo ad ogni tratto di strada.

Questo ha portato a cercare di suddividere i segmenti da segnalare con diversi gradi di giudizio che fondamentalmente si basano sulla larghezza del percorso pedonale, la pendenza, la presenza o meno di piccole barriere architettoniche e il tipo di pavimentazione. Per rendere evidenti queste differenze si è pensato di utilizzare percorsi con diverse colorazioni che indicano il livello di accessibilità degli stessi; questo dovrebbe dare una chiara indicazione all'utente sulla sua possibilità o meno di poter praticare una strada cittadina.
Oltre alla segnalazione delle strade è utile inserire nella mappa alcune informazioni che diano un'indicazione precisa delle caratteristiche del territorio urbano; in molti casi, infatti, il possibile problema per il disabile può essere determinato da un ostacolo presente in una zona molto limitata. Questo ha portato a pensare all'utilizzo di marker posizionati in particolari zone di interesse, in modo da indicare con estrema precisione la posizione di una caratteristica urbana in stretta relazione con le necessità o difficoltà che può incontrare il disabile.
Quello che si è pensato di inserire, come prima versione della soluzione applicativa è un insieme di simboli che indicano passaggi pedonali accessibili, rampe di accesso ai marciapiedi, barriere architettoniche, strettoie e fondi stradali dissestati.
In seguito a di tali premesse si è deciso di inserire nell'applicazione i seguenti simboli:


Passaggio pedonale accessibile o rampa per disabili


Barriera architettonica


Strettoia


Strada dissestata

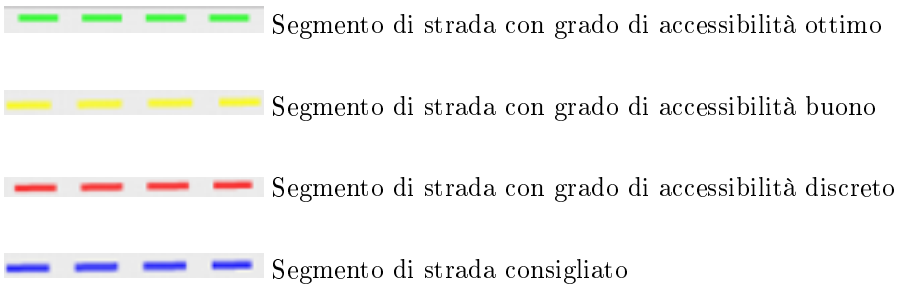

Figure 1.1 - 1.8: simboli utilizzati nell'applicazione

3.2 Funzionalità partecipativa

La funzionalità partecipativa è senza dubbio ciò che rappresenta l'essenza dell'applicazione qui descritta; la condivisione e la partecipazione sono infatti fondamentali per avere un database sempre aggiornato e ricco di informazioni utili. Per tale motivo mi sono posto come obiettivo principale quello di permettere all'utente, con un procedimento molto semplice, l'inserimento dei nuovi dati da poter condividere. Ipoteticamente, chiunque utilizzi l'applicazione, una volta verificata la presenza di una barriera, un particolare ostacolo o un ausilio per disabili, può in pochi istanti inserire un marker. Allo stesso modo, una volta percorso una strada e verificato la sua accessibilità, questa può essere segnalata con qualche tocco. L'interazione infatti è data dalla tecnologia touchscreen, che permette di agire direttamente sulla mappa renderizzata. In modo simile, l'applicazione include anche la possibilità di eliminare tali dati.
Ciò che rende possibile la condivisione dei dati è un database che permette il loro accesso a più client.
Si è optato per un database SQLite, libreria per creazione e gestione di database molto veloce e leggera; il motivo principale di tale scelta è dato dalla natura stessa del progetto, ovvero quella di permettere la consultazione di una mappa,

con le informazioni a disposizione, anche in caso di assenza di una connessione alla rete mobile; è quindi fondamentale la presenza, in locale, dei dati di renderizzazione della mappa e delle informazioni poste al livello superiore di quest'ultima. Naturalmente, per avere un database aggiornato, è opportuno accedere alla rete quando possibile.
Per concludere, la funzionalità partecipativa include anche la possibilità di eliminare i dati salvati, nel caso in cui l'utente voglia modificare le informazioni presenti sulla mappa.

3.3 Criteri di valutazione

Ci soffermiamo ora sulla funzionalità di inserimento dati, per andare a comprendere meglio quali siano i criteri di valutazione dei segmenti di strada che possiamo evidenziare sulla mappa. Come detto in precedenza, si vogliono rappresentare tre tipologie di strada, tramite una suddivisione rispetto al grado di accessibilità della stessa. Ciò a cui puntiamo è dare all'utente il compito di partecipare attivamente allo sviluppo della nostra applicazione mediante l'inserimento di alcuni dati utili alla determinazione del grado. La quarta tipologia di segmento, invece, è priva di questi dati, e può essere utilizzata dall'utente quando non ha la possibilità di avere delle misure precise, ma vuole comunque segnalare il tratto di strada appena percorso.
Posti tali obiettivi, andiamo a vedere ciò che possiamo definire accessibile o non accessibile.
Possimo sicuramente dire che non vi è una definizione univoca di accessibilità, questo è dovuto alle differenti caratteristiche di ogni persona, e ciò che possiamo fare è attenerci alle normative ministeriali in vigore, con particolare riferimento al decreto del 14 giugno 1989, n.236 il quale illustra le prescrizioni tecniche necessarie a garantire l'accessibilita', l'adattabilita' e la visitabilita' degli edifici privati e di edilizia residenziale pubblica sovvenzionata e agevolata, ai fini del superamento e dell'eliminazione delle barriere architettoniche.

Nel contesto urbano, un elemento a cui dobbiamo fare attenzione è il marciapiede e le sue caratteristiche, le misure indicate come adatte ad una struttura accessibile ai disabili sono le seguenti:

- **Larghezza del marciapiede:** una giusta misura per un modello pedonale è 150 cm; questo dovrebbe permettere il pasaggio contemporaneo di due carrozzine.
- **Dislivello fra zone adiacenti:** il dislivello, in termini d'altezza, non dovrebbe superare i 2,5 cm.
- **Pendenza:** nei tratti inclinati del marciapiede o percorso esterno, la pendenza non deve superare il 5%.
- **Pendenza laterale:** deve essere al massimo dell'1,5%.
- **Pavimentazione:** dovrebbe essere costruita in materiale non sdrucciolevole e ben livellata, si dovrebbe inoltre evitare l'utilizzo di grate con fessure troppo larghe, per evitare che ruote, bastoni o altri ausili si possano incastrare.

Altro elemento da tenere in considerazione in un contesto urbano sono le rampe, che in generale dovrebbero avere le seguenti caratteristiche:

- **Dislivello considerato accessibile:** un dislivello superiore ai 3,20 m ottenuto esclusivamente mediante rampe non è considerato accessibile.
- **Larghezza delle rampe:** la larghezza ottimale è di 150 cm, per permettere il passaggio di due persone, mentre quella minima è di 90 cm.
- **Ripiano orizzontale:** qualora vi sia una successione di rampe, è necessario che vi sia alla fine di ogni rampa un pianerottolo con dimensioni minime di 1,50 x 1,50 m.
- **Pendenza delle rampe:** non deve superare l'8% (8 cm di alzata ogni metro di lunghezza presa in piano).

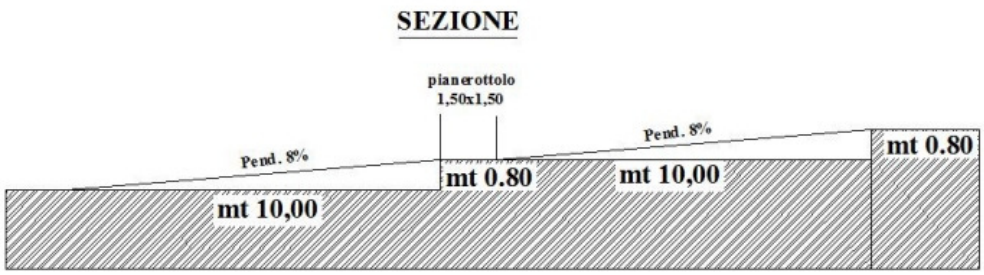

Figura 3.1: indicazione delle misure per le rampe

Oltre a questi elementi è importante che i passaggi pedonali siano adeguatamente attrezzati per l'attraversamento da parte dei non vedenti e che, in caso di strade molto larghe, vi siano apposite aiuole salvagente.
Alla luce di quanto detto in precedenza e delle misure appena viste si è deciso di definire per ogni caratteristica dei range di accettabilità. Ciò che ci interessa maggiormente è concentrarsi sui marciapiedi o i percorsi pedonali esterni in generale. Ecco le suddivisioni e i valori scelti per le valutazioni a livello applicativo:

Accessibilità con grado ottimo	**min**	**max**
Larghezza (cm)	150	-
Dislivello zone adiacenti (cm)	-	2
Pendenza laterale (%)	-	0.5
Pendenza (%)	-	4

Accessibilità con grado buono	**min**	**max**
Larghezza (cm)	91	149
Dislivello zone adiacenti (cm)	2.1	2.5
Pendenza laterale (%)	0.6	1.2
Pendenza (%)	4.1	6

Accessibilità con grado discreto	**min**	**max**
Larghezza (cm)	70	90
Dislivello zone adiacenti (cm)	2.6	3.5
Pendenza laterale (%)	1.3	2
Pendenza (%)	6.1	8.5

Tabelle 3.1 - 3.3: range di accettabilità per ogni tipo di segmento stradale

Con i dati presenti nelle tabelle sarà possibile, al momento dell'inserimento di un nuovo segmento stradale, determinare il grado di accessibilità della strada. In generale, per ogni insieme di caratteristiche, ci si baserà sul dato ritenuto peggiore per quanto riguarda l'accessibilità.

3.4 Funzionalità di navigazione

Ciò che ci si aspetta dalla funzionalità di navigazione è la possibilità, sulla base dei dati presenti nel database, di calcolare dei percorsi adatti a particolari richieste da parte dell'utente. Chi utilizza tale funzionalità dovrebbe poter, naturalmente, inserire un indirizzo di partenza ed uno di arrivo, aggiungendo anche la possibilità di indicare particolari condizioni del percorso come la larghezza, la pendenza praticabile, il dislivello massimo attraversabile, la

disponibilità ad affrontare tratti di strada privi di marciapiede ed altre caratteristiche. Quello a cui si è pensato, a livello applicativo, è quello di dare la possibilità, attraverso l'iserimento dei dati in un form, di effettuare la richiesta di calcolo.
Questa funzionalità non è stata ancora integrata nella nostra applicazione, ma è stato svolto uno studio, creato un modello e sviluppata, a parte, una possibile soluzione implementativa. Il motivo di questa scelta è la limitazione data dalla libreria Mapsforge; la possibilità di accedere a certi tipi di dato, infatti, non è ancora possibile, ma in fase di sviluppo. Per questo si è cercato di intraprendere un lavoro di integrazione dei dati utili alla navigazione utilizzando i dati grezzi provenienti dal progetto OpenStreetMap e tale funzionalità è ancora ad uno stato sperimentale. Vedremo in seguito, nel capitolo dedicato all'implementazione delle funzionalità, quali sono i dettagli e le soluzioni implementative a cui si è pensato.

4 Il rendering della mappa e degli Overlay

Questo capitolo si addentra nelle specifiche di implementazione delle funzionalità basilari dell'applicazione, ovvero l'elaborazione dei dati necessari alla visualizzazione, il rendering della mappa e la creazione e gestione di un database che permetta la corretta visualizzazione degli overlay renderizzati sulla mappa; in particolare vedremo quali sono gli strumenti utilizzati e le parti più interessanti del codice creato.

4.1 La mappa

Per il rendering della mappa è stata inizialmente individuata un'area su cui si voleva lavorare. Una volta scelta l'area geografica di lavoro, meglio conosciuta come bounding box, si sono ottenuti, attraverso il sito del progetto OpenStreetMap, i dati grezzi della mappa in questione; questi dati non sono altro che informazioni di tipo geografico descritte tramite un linguaggio di markup. Una volta ottenuto il file in formato OSM è stato possibile, grazie all'applicazione Osmosis[8] di OpenStreetMap e al plugin di Mapsforge Map-Writer[9], creare il file *Bologna.map*. Questo file serve a garantire il funzionamento offline dell'applicazione Mapsforge e di conseguenza è necessario il suo salvataggio nel percorso */mnt/sdcard/* di ogni device Android che la deve eseguire. Va detto che il file creato sarebbe sufficiente ad ottenere una resa tradizionale (OSM) della mappa in questione, tuttavia, visto lo scopo finale dell'applicazione, si è optato per una resa molto semplice e lineare, per permettere una facile consultazione delle informazioni aggiunte in seguito sulla mappa. Per questo si è creato il file *renderTheme.xml* con delle particolari regole di rendering:

[8] Applicazione disponibile su http://wiki.openstreetmap.org/wiki/Osmosis
[9] Plugin disponibile su http://code.google.com/p/mapsforge/wiki/GettingStartedMapWriter

```
<?xml version="1.0" encoding="UTF-8"?>
<rendertheme xmlns="http://mapsforge.org/renderTheme" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://mapsforge.org/renderTheme renderTheme.xsd" version="2" map-background="#FFEE99">
<!-- ways -->
    <rule e="way" k="*" v="*">

        <!-- building -->
        <rule e="way" k="building" v="*">
            <area fill="#d9ccbe" stroke="#d3c6b9" stroke-width="0.2" />

            <rule e="way" k="*" v="*" zoom-min="17">
                <caption k="name" font-style="bold" font-size="10" fill="#ffffff" stroke="#d0c9c3" stroke-width="2.0" />
                <caption k="addr:housenumber" font-style="bold" font-size="10" fill="#8c837b" stroke="#ffffff" stroke-width="2.0" />
            </rule>
        </rule>
        <rule e="way" k="highway" v="*">
            <!-- no tunnel -->
            <rule e="way" k="tunnel" v="~|no|false">
                <!-- no area -->
                <rule e="way" k="area" v="~|no|false">
                    <!-- highway cores -->
                    <rule e="way" k="highway" v="service">
                        <line stroke="#ffffff" stroke-width="0.85" />
                        <rule e="way" k="*" v="*" zoom-min="15">
                            <pathText k="name" font-style="bold" font-size="10" stroke="#ffffff" stroke-width="2.0" />
                        </rule>
                    </rule>
                    <rule e="way" k="highway" v="road">
                        <line stroke="#ffffff" stroke-width="1.25" />
                        <rule e="way" k="*" v="*" zoom-min="15">
                            <pathText k="name" font-style="bold" font-size="10" stroke="#ffffff" stroke-width="2.0" />
                        </rule>
                    </rule>
                    <rule e="way" k="highway" v="pedestrian">
                        <line stroke="#ffffff" stroke-width="1.35" />
                        <rule e="way" k="*" v="*" zoom-min="15">
                            <pathText k="name" font-style="bold" font-size="10" stroke="#ffffff" stroke-width="2.0" />
                        </rule>
                    </rule>

                    [...OMISSIS...]

                </rule>
            </rule>
        </rule>
    </rule>
</rendertheme>
```

Nella fase seguente, attraverso il software di sviluppo Android SDK ed Eclipse Juno si è creato un virtual device, nello specifico un emulatore del tablet Asus Nexus 8 con sistema operativo Android 2.3.3 come target. All'interno del pacchetto di sviluppo di Android è possibile utilizzare alcuni tool, molto utili allo sviluppo di applicazioni e alla fase di testing. Grazie al tool *adb*, che si trova in */android-sdk-linux/plattform-tools* è possibile eseguire la shell del virtual device appena avviato dall'AVD Manager dell'SDK; con questo tool è possibile gestirne il file system e tramite il comando *./adb push* si è caricato in memoria il file *Bologna.map* e *renderTheme.xml*, posizionandoli alla radice del file system della sdcard, opportunamente creata e dimensionata, in precedenza, con l'ausilio dell'AVD Manager.

Passando all'implementazione del codice, non possiamo non soffermarci su un elemento fondamentale per la comprensione di un'applicazione Android: l'Activity; tale classe è uno degli elementi centrali di ogni applicazione e solitamente ne rappresenta una singola schermata. Possono essere definite diverse Activity ed ognuna è responsabile del salvataggio del proprio stato; questo per garantire un riutilizzo successivo, come parte del ciclo di vita dell'applicazione stessa. Generalmente una Activity rappresenta un'azione specifica ed essendo una delle componenti principali nel ciclo di vita di ogni applicazione Android, la determinazione del loro modo di interagire è fondamentale.
Inizialmente è stata creata l'Activity principale, dalla quale si diramano tutte le altre Activity che vanno a rappresentare le diverse funzionalità applicative, in poche parole la prima classe rappresenta il menu principale e permette la scelta dell'azione da svolgere. Questa è parte del codice creato per la fase iniziale:

```
public class MenuActivity extends MapActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_activity1);
        final Button aprimap = (Button) findViewById(R.id.button1);
        [...OMISSIS...]
        aprimap.setOnClickListener(new OnClickListener(){
                public void onClick(View v) {
                        if(v == aprimap)
                                startMap();
                        }
                });
        }

        private void startMap(){
                Intent launchMap = new Intent(this, Activity2.class);
                startActivity(launchMap); }
                [...OMISSIS...]
        }
```

In questa parte di codice è possibile vedere come è stato implementato il lancio dell'Activity di visualizzazione della mappa; la selezione della funzionalità avviene tramite un bottone che ne permette l'avvio. Il layout dei bottoni e

delle stringhe presenti nelle activity sono opportunamente definiti in file XML specifici.

Come detto in precedenza, la libreria Mapsforge permette di sviluppare applicazioni contenenti mappe geografiche, questo è reso possibile attraverso l'override dei principali metodi della classe MapView di Google Maps API[10]. Vediamo come è stato possibile, nel nostro codice, avviare il rendering e specificare quali sono i file da utilizzare a tale scopo:

```
@Override protected void onCreate(Bundle savedInstanceState) {
    //rendering mappa
    super.onCreate(savedInstanceState);
    MapView mapView = new MapView(this);
    mapView.setClickable(true);
    mapView.setBuiltInZoomControls(true);
    //'mnt/sdcard/Bologna.map' e 'mnt/sdcard/renderTheme.xml'
    mapView.setMapFile(new File(Environment.getExternalStorageDirectory().getPath(), "Bologna.map"));
    File xmlF = new File(Environment.getExternalStorageDirectory().getPath(), "renderTheme.xml");
    try { mapView.setRenderTheme(xmlF); }
    catch (FileNotFoundException e) {
            catch block e.printStackTrace();
    }
    setContentView(mapView);
```

Ciò che è sicuramente fondamentale per il rendering è la presenza del file *Bologna.map,* per quanto riguarda il file XML creato per dare alcune regole di rendering, in caso di eccezione a causa di mancata lettura, viene eseguito un rendering di default con le caratteristiche tipiche del progetto OpenStreetMap.

[10] API di Google per creare mappe, disponibile su https://developers.google.com/maps/?hl=it

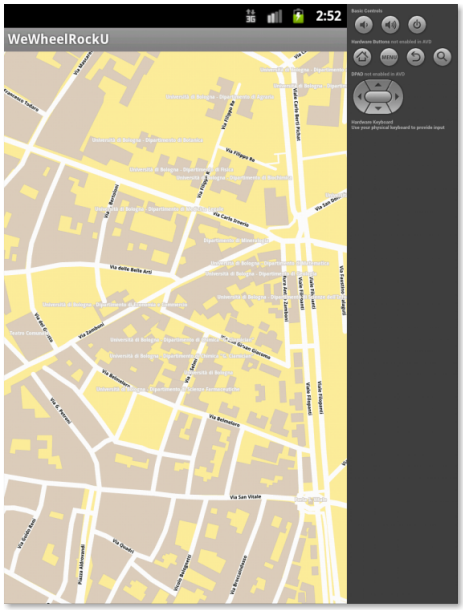

Figura 4.1: rendering della mappa

4.2 Il database

Per meglio introdurre la parte riguardante il rendering degli overlay è utile parlare del database, di come è stato creato e come è possibile gestirlo. Innanzitutto bisogna prendere in cosiderazione il tipo di dato che si vuole salvare, tipicamente dei Geopoint e un nome di riferimento; il Geopoint rappresenta un punto sulla superficie terrestre, ogni punto sulla superficie è univoco e la sua rappresentazione è data da due valori: latitudine e longitudine. Tali valori, nel tipo GeoPoint, sono salvati assieme sotto forma di microgradi (*microdegrees*) e sono senza dubbio fondamentali per la nostra funzionalità di overlay.
In primo luogo è stato necessario, mediante l'interfaccia Basecolumns[11], creare un'estensione di quest'ultima, in modo da avere delle tabelle personalizzate nelle quali poter salvare i punti geografici rappresentanti i diversi tipi di marker inseriti sulla mappa.

[11] http://developer.android.com/reference/android/provider/BaseColumns.html

```
public interface MarkerTable_Pass extends BaseColumns {
    String PASSAGGI = "markers_pass";
    String CODICE1 = "codice1";
    String CODICE = "codice";
    String NOME = "nome";
    String[] COLUMNS = new String[] { _ID , CODICE, CODICE1, NOME };
}
```

Quello che più ci interessa sono le stringhe *CODICE* e *CODICE1*, esse rappresenteranno i nostri geopoint, e la tabella nel database, che in questo caso rappresenta i marker utilizzati per indicare passaggi pedonali accessibili e rampe dedicate ai disabili, avrà tante righe quanti sono gli elementi di questo tipo all'interno del database. Le interfacce per le altre tipologie di marker sono create allo stesso modo mentre, per quel che riguarda i segmenti di strada, abbiamo una coppia di geopoint che indica il collegamento da un incrocio ad un altro. L'inizializzazione del database e delle tabelle è fatta nel modo seguente:

```
private static final String DATABASE_NAME = "devAPP.db";
private static final int SCHEMA_VERSION = 1;
public DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, SCHEMA_VERSION);
}
@Override public void onCreate(SQLiteDatabase db) {
    [...OMISSIS...]
    String sql = "CREATE TABLE {0} ({1} INTEGER PRIMARY KEY AUTOINCREMENT,
    {2} TEXT NOT NULL,{3} TEXT NOT NULL, {4} TEXT NOT NULL);";

    db.execSQL(MessageFormat.format(sql, MarkerTable_Pass.PASSAGGI, MarkerTable_Pass._ID,
    MarkerTable_Pass.NOME, MarkerTable_Pass.CODICE, MarkerTable_Pass.CODICE1));
    [...OMISSIS...]
}
```

Il database *devAPP.db* viene salvato nel file system del device target e collocato in */data/data/nome_progetto/databases/*. Vediamo ora come è stato possibile implementare l'inserimento dei dati nel database; l'interazione da parte dell'utente, per effettuare l'inserimento dei dati, verrà trattata nel capitolo dedicato all'implementazione della funzionalità partecipativa. Per effettuare

gli inserimenti nel database è stata utilizzata la classe ContentValues (metodo *put*) ed in seguito il metodo *insert* della classe SQLiteDatabase.

```
public void inserisciGeopoint(SQLiteDatabase db, String nome, double codice, double codice1, int table) {
        switch(table) {
            case 1:
                    ContentValues v = new ContentValues();
                    v.put(MarkerTable_Pass.CODICE, codice);
                    v.put(MarkerTable_Pass.CODICE1, codice1);
                    v.put(MarkerTable_Pass.NOME, nome);
                    db.insert(MarkerTable_Pass.PASSAGGI, null, v);
                    break;
            case 2:
                    [...OMISSIS...]
        }
}
```

Il valore *table* indica il tipo di marker che si vuole inserire; allo stesso modo è stato creato il metodo *inserisciLink* per aggiungere nel database i segmenti di strada da evidenziare.
Un ulteriore metodo importante per il database rappresenta l'interrogazione dello stesso; tale metodo verifica il tipo di richiesta e ritorna il valore richiesto. Per permettere lo scorrimento delle tabelle si utilizza un cursore posizionato appositamente.

```
public Cursor getPoint(int marktype) {
        switch (marktype) {
                case 1: return(getReadableDatabase().query(MarkerTable_Pass.PASSAGGI,
                        MarkerTable_Pass.COLUMNS, null, null, null, null, null));
                break;
                case 2: return(getReadableDatabase().query(MarkerTable.BARRIERE,
                        MarkerTable.COLUMNS, null, null, null, null, null));
                break;
                [...OMISSIS...]
        }
}
```

4.3 Gli overlay

Gli overlay rappresentano tutte le informazioni che vogliamo aggiungere sulla nostra mappa. Come detto in precedenza sono di diverso tipo e sono rappresentati da differenti tabelle nel database. La procedura, ripetuta per tutti i tipi di marker, è quella di interrogare il database, ottenere i dati e salvarli in alcune liste che vengono in seguito date in input ai metodi della libreria Mapsforge per il rendering. Allo stesso modo, oltre al metodo *getPoint*, è stato creato anche il metodo *getLink* che permette di ottenere la coppia di geopoint che serve al rendering dei segmenti utilizzati per tracciare le strade accessibili. Nell'Activity di visualizzazione della mappa avremo quindi, in seguito alla fase di rendering della mappa, il seguente codice:

```
DatabaseHelper databaseHelper = new DatabaseHelper(this);
//PASSAGGI PEDONALI
Cursor c1 = databaseHelper.getPoint(1);
try {
      while (c1.moveToNext()) {
          /*Stampa geopoints nel Log*/
          Log.d("devAPP_passaggi", c1.getLong(0) + " " + c1.getDouble(1) + " " +
          c1.getDouble(2) + " " + c1.getString(3));
          /*salva geopoint*/
          Gp = new GeoPoint(c1.getDouble(1), c1.getDouble(2));
          Passaggi_geoPointsArray.add(c1.getDouble(1));
          Passaggi_geoPointsArray.add(c1.getDouble(2));
      }
}
finally {
    c1.close();
}
```

Una volta ottenuta una lista di elementi di tipo double, questi vengono opportunamente convertiti al tipo Geopoint ed inseriti in una lista apposita. In seguito si utilizza il metodo di rendering di Mapsforge per poter finalmente visualizzare i nostri overlay.

```
ListOverlay listOverlay = new ListOverlay();
List<OverlayItem> overlayItems = listOverlay.getOverlayItems();
[...OMISSIS...]
Iterator<Marker> itr = markers_passaggi.iterator();
while(itr.hasNext()) {
        Marker element = itr.next();
        overlayItems.add(element);
}
[...OMISSIS...]
//Overlays rendering
mapView.getOverlays().add(listOverlay);
```

L'applicazione contiene di default alcuni marker e segmenti, per ogni tipo di overlay; vediamo in seguito un'immagine che illustra, attraverso il virtual device, il risultato ottenuto.

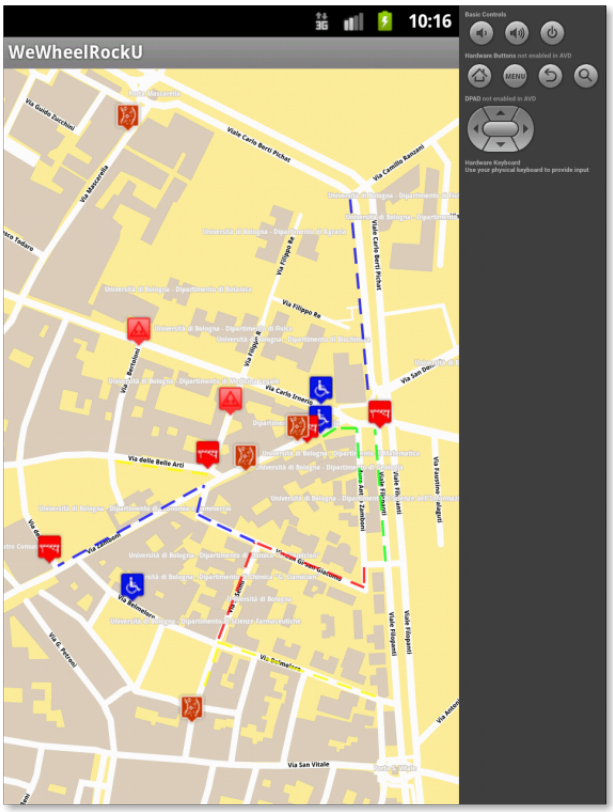

Figura 4.2: rendering degli overlay

5 L'inserimento di nuovi dati

Questo capitolo descrive nel dettaglio la funzionalità partecipativa e la sua implementazione; vediamo nello specifico quali sono state le scelte implementative che permettono l'interazione dell'utente con l'applicazione, al fine di effettuare le operazioni di inserimento dei dati da condividere con gli altri utenti.

5.1 L'inserimento dei marker

Giunti al completamento dello step di rendering della mappa e dei suoi overlay, si è passati all'interazione dell'utente con l'applicazione; è bene far notare che, anche nel caso del rendering è possibile, tramite la tecnologia touchscreen, spostarsi all'interno della mappa ed effettuare molteplici operazioni di zoom, a seconda delle esigenze.
Dall'activity principale si può attivare la funzionalità in questione attraverso il bottone *Aggiungi Marker*; come detto in precedenza, l'applicazione permette al momento, l'inserimento di quattro tipi diversi di marker. Attraverso i Dialog di Android è stato creato un menu e la possibilità di inserire una descrizione testuale del marker da aggiungere. Dopo aver inserito una descrizione (facoltativa) e scelto il tipo, è possibile muoversi sulla mappa e tramite degli appositi bottoni per lo zoom o tramite doppio tocco veloce posizionarsi sul punto desiderato ed in seguito, con un tocco lungo (Longpress), salvare il marker. Vediamo ora come è stata implementata tale Activity.

Questa parte di codice implementata riguarda i Dialog, per l'inserimento di testo e la scelta del tipo di marker da aggiungere:

```
final String[] options={"Passaggio/Scivolo","Barriera","Strettoia","Buca/Strada Dissestata"};
[...OMISSIS...]
final AlertDialog.Builder builder=new AlertDialog.Builder(this);
final AlertDialog.Builder saved=new AlertDialog.Builder(this);
final AlertDialog.Builder textinfo=new AlertDialog.Builder(this);
builder.setTitle("Scelta tipo di Marker");
saved.setTitle(" Marker Aggiunto");
builder.setItems(options, new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
whichMarker = which + 5; //valore che permette il riconoscimento del tipo di marker da aggiungere
} });
builder.show();
[...OMISSIS...]
//testo per descrivere il marker
LayoutInflater factory = LayoutInflater.from(this);
final View textEntryView = factory.inflate(R.layout.text_entry_marker, null);
final EditText input1 = (EditText) textEntryView.findViewById(R.id.EditText1);
input1.setHint("Inserisci descrizione o note");
input1.setText("");
textinfo.setView(textEntryView).setPositiveButton("Salva testo", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
        text = input1.getText().toString();
        Log.i("AlertDialog","Testo inserito: "+ text);
    }
}).setNegativeButton("Nessuna descrizione",new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog,int whichButton) {
        text = "";
        Log.d("AlertDialog", "Testo inserito: "+ text);
    }
});
textinfo.show();
```

In seguito, dopo la fase di rendering della mappa e di tutti i dati presenti nel database (capitolo precedente), attraveso la classe MotionEvent e Gesture-detector, l'applicazione attende che l'utente indichi il punto su cui salvare il marker. Nello specifico attendiamo un Longpress sul touchscreen:

```
final GestureDetector gestureDetector = new GestureDetector(new GestureDetector.SimpleOnGestureListener() {
    public void onLongPress(MotionEvent e) {
    Log.e("", "Longpress detected");
    pressed = true;
    }
});
//Inserimento geopoint - longpress
mapView.setOnTouchListener(new MapView.OnTouchListener() {
    GeoPoint p = null;
```

```
    public boolean onTouch(View v, android.view.MotionEvent event) {
        gestureDetector.onTouchEvent(event);
        if (pressed){
            p = mapView.getProjection().fromPixels((int) event.getX(),(int) event.getY());
            double Lat = p.latitude;
            double Lon = p.longitude;
            Log.d("TAP", Lat + " " + Lon);
            SQLiteDatabase data;
            data = db.getWritableDatabase();
            db.inserisciGeopoints(data, whichMarker, text, Lat, Lon);
            Log.d("SALVA MARKER", whichMarker + " " + Lat + " " + Lon + " " + text);
            saved.setItems(ok, new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                }
            });
            saved.show();
            pressed = false;
        }
    return false;
    }
});
```

I geopoint si ottengono utilizzando il metodo *getProjection()* che calcola i punti geografici a partire dai pixel del touchscreen. Vediamo nelle seguenti immagini i due passi iniziali da fare per poter inserire i marker:

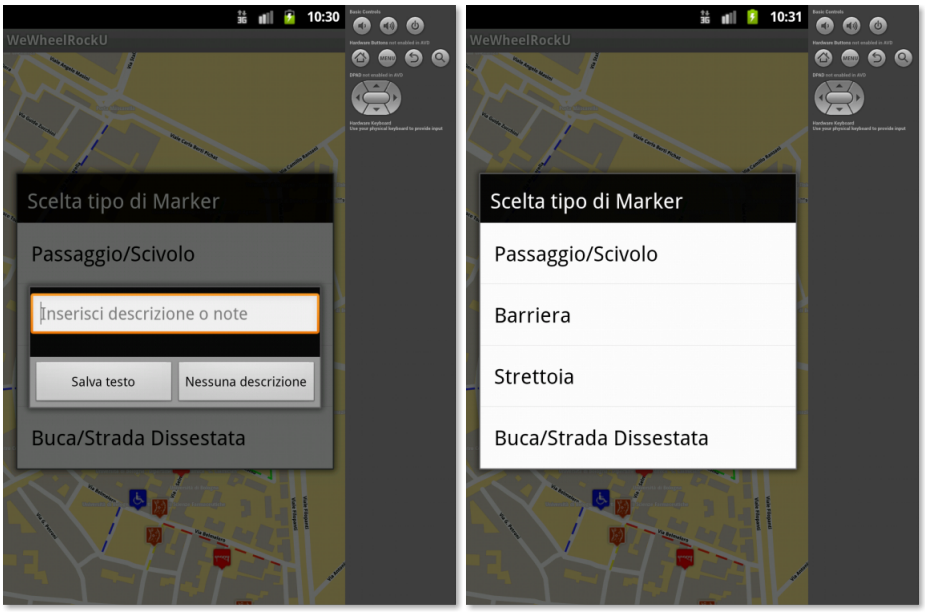

Figure 5.1 e 5.2 : screenshot delle fasi di inserimento dati

In seguito ai passi descritti dalle immagini precedenti è possibile selezionare un punto sulla mappa dove salvare il marker scelto, apparirà in seguito un messaggio di conferma. Il testo descrittivo apparirà in seguito ad un touch effettuato nelle vicinanze del marker e sarà visibile anche nel Log di debug dell'applicazione. Nel caso di sviluppi futuri sarà interessante migliorare la grafica del messaggio attraverso i classici Balloon che si utilizzano nelle mappe.

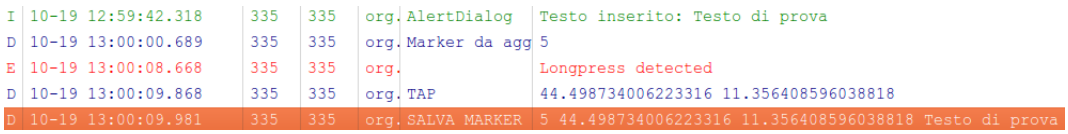

Figura 5.3: visualizzazione del Log con le informazioni del marker aggiunto

5.2 L'inserimento dei segmenti

Sulla base di quanto detto nel terzo capitolo, quando si parlava dei criteri di valutazione, è stata implementata la funzionalità di inserimento dei segmenti di strada accessibili, fornendo all'utente la possibilità di inserire i dati che caratterizzano il percorso che vuole segnalare. Nel caso in cui l'utente non abbia a disposizione questi dati, egli può consigliare comunque un segmento di strada; possiamo considerare tale tratto di strada come un dato pressochè affidabile, non verificato sulla base degli standard di accessibilità descritti nei termini legislativi, ma in base all'esperienza personale della persona che lo condivide. Utilizzando ancora la classe Dialog, permettiamo l'interazione dell'utente, dandogli la possibilità di inserire i dati del tratto pedonale in questione. Vediamo la parte di codice che permette tale interazione.

```
LayoutInflater factory = LayoutInflater.from(this);
final View textEntryView = factory.inflate(R.layout.text_entry, null);
final EditText input1 = (EditText) textEntryView.findViewById(R.id.EditText1);
input1.setHint("Larghezza (min 70 cm)");
final EditText input2 = (EditText) textEntryView.findViewById(R.id.EditText2);
```

```
input2.setHint("Dislivello massimo (0 - 3.5 cm)");
final EditText input3 = (EditText) textEntryView.findViewById(R.id.EditText3);
input3.setHint("Pendenza (0 - 8.5%)");
final EditText input4 = (EditText) textEntryView.findViewById(R.id.EditText4);
input4.setHint("Pendenza laterale (0 - 2%)");
input1.setText("");
input2.setText("");
input3.setText("");
input4.setText("");
final AlertDialog.Builder builder=new AlertDialog.Builder(this);
final AlertDialog.Builder saved=new AlertDialog.Builder(this);
final AlertDialog.Builder saved2=new AlertDialog.Builder(this);
builder.setTitle("Inserisci i dati o consiglia una strada");
[...OMISSIS...]
builder.setView(textEntryView).setPositiveButton("Salva e inserisci punti", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int which) {
    String l, d, p, p_l;
    Double larg, disl, pend, pend_lat;
    [...OMISSIS...]
    l = input1.getText().toString();
    larg = Double.parseDouble(l);
    d = input2.getText().toString();
    disl = Double.parseDouble(d);
    [...OMISSIS...]
} }).setNegativeButton("Consiglia Strada (non ho i dati)",new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog,int whichButton) {
    //strada tipo - consigliato
    streetType = 3;
    whichMarker = streetType + 9;
    Log.d("Link da aggiungere", whichMarker + " ");
    }
});
```

In seguito a questo abbiamo, tramite un'implementazone simile a quella della sezione precedente, la fase di posizionamento e inserimento dei Geopoint che formano il segmento stradale. I punti da inserire per poter ottenere gli overlay tratteggiati sono due.

```
public boolean onTouch(View v, android.view.MotionEvent event) {
    if (event.getAction() == android.view.MotionEvent.ACTION_DOWN){
    gestureDetector.onTouchEvent(event);
    if (pressed){
        point++;
        p = mapView.getProjection().fromPixels((int) event.getX(),(int) event.getY());
        if (point == 1) {
            Lat = p.latitude;
            Lon = p.longitude;
            Log.d("PRIMO LINK", Lat + " " + Lon);
        }
        if (point == 2) {
            Lat2 = p.latitude;
            Lon2 = p.longitude;
            SQLiteDatabase data;
            data = db.getWritableDatabase();
```

```
            db.inserisciLinks(data, whichMarker, "", Lat, Lon, Lat2, Lon2);
            Log.d("SECONDO LINK", Lat2 + " " + Lon2);
        }
        [...OMISSIS...]
});
```

Il segmento di strada (polyline) viene quindi inserito nel database, nella tabella contenente i dati riguardanti i tratti di strada dello stesso tipo. Vediamo qui di seguito un immagine con il Dialog e l'EditText per l'inserimento dei dati:

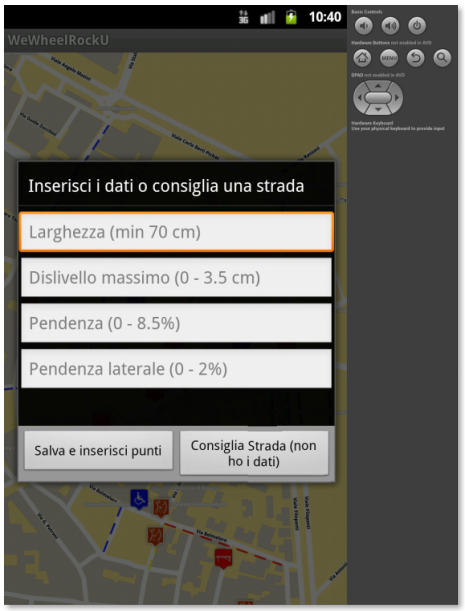

Figura 5.4: screenshot - inserimeto dati segmento

6 Il calcolo dei percorsi

Questo capitolo vuole descrivere una possibile soluzione al tutt'altro che banale problema del calcolo di percorsi adatti ai disabili. Come detto in precedenza, la libreria Mapsforge non fornisce ancora la possibilità di avere degli strumenti di navigazione, e tali funzionalità sono infatti ancora in una fase prototipale, questo è dovuto principalmente alle limitate risorse offerte dai device mobili e dalla necessità di ottenere una solida struttura dati a partire dalle informazioni ottenute dal progetto OpenStreetMap. Analizzeremo quindi quali sono le attuali possibilità per risolvere il problema e, basandoci su un progetto svolto da me e dal collega Giuseppe Balistreri nell'ambito della simulazione ed inerente a tale lavoro di tesi, vedremo sotto l'aspetto implementativo quali sono le migliori scelte da prendere. Va detto che il suddetto progetto di simulazione verrà descritto dettagliatamente nel prossimo capitolo e che momentaneamente ci è sufficiente estrarre alcune delle caratteristiche principali che non necessitano di lunghe presentazioni.
Parlando di navigatori basati sul progetto OpenStreetMap non possiamo evitare di parlare di alcuni progetti open source presenti in rete ed in fase di sviluppo. Durante la mia esperienza di tirocinio curricolare, svolto per il corso di Laurea in Informatica presso l'azienda tedesca Opensynergy Gmbh di Berlino, ho avuto modo di effettuare diversi test con i più comuni software di navigazione per Android e legati ad OSM[12]. Lo scopo di questo lavoro è stato quello di individuare un sistema di navigazione che lavorasse completamente offline, che fosse distribuito con una licenza software libera (tipicamente GPL o BSD) e che potesse essere facilmente integrato al progetto aziendale. Sulla base di tali premesse ho effettuato le mie ricerche ed individuato come migliori soluzioni i progetti GraphHopper, OsmAnd e ZANavi. Tali progetti permettono, a differenza della gran parte dei navigatori per Android, di funzionare completamente offline. Un punto fondamentale che tutti questi progetti hanno in comune è quello di effettuare una particolare estrapolazione e riorganizzazione

[12] Lista dei progetti OSM per Android disponibile su http://wiki.openstreetmap.org/wiki/Android

dei dati OSM, che spesso necessitano di un'ottimizzazione a livello strutturale per avere un'efficiente sistema di navigazione. In maniera del tutto intuitiva, il lavoro svolto da me e Giuseppe si avvicina molto al progetto GraphHopper. L'idea generale è quella di avere una struttura a nodi che permetta poi con un algoritmo quale A* di trovare il percorso più breve per raggiungere una destinazione specifica. L'algoritmo A* si basa, sostanzialmente, sul calcolo della distanza di ogni nodo rispetto alla destinazione; ad ogni passo si sceglie il prossimo nodo con distanza minore, fino ad arrivare al target. Si vuole, a questo punto, applicare tale idea alla nostra applicazione per disabili, con una differenza sostanziale: partendo da una struttura di nodi ed archi che formano la nostra rete urbana di strade ed incroci (OSM) si vuole, al momento dell'inserimento di un segmento di strada accessibile, effettuare un assegnamento ai punti OpenStreetMap, in maniera da avere la corretta posizione (geopoint) dei nostri dati. Da tale idea si vorrebbe in futuro creare un tool che, analizzando i dati dei file di formato *.osm*, crei il database con tutti i nodi rappresentabili sulla mappa. Vediamo come possiamo individuare i nodi aprendo il file con un semplice editor di testo. Questi file sono caratterizzati da un linguaggio di markup e i nodi stradali che ci interessano sono rappresentati dal tag *</node>*:

```
</node>
 <node id="1704257912" visible="true" version="2" changeset="12221788" timestamp="2012-07-14T23:07:41Z" user="mcheck" uid="478872" lat="44.4970473" lon="11.3538566">
 <tag k="addr:city" v="Bologna"/>
 <tag k="addr:country" v="IT"/>
 <tag k="addr:district" v="San Vitale"/>
 <tag k="addr:housenumber" v="2"/>
 <tag k="addr:postcode" v="40126"/>
 <tag k="addr:street" v="Via San Giacomo"/>
 <tag k="source" v="Comune di Bologna"/>
</node>
<node id="1704276112" visible="true" version="2" changeset="12221788" timestamp="2012-07-14T23:07:42Z" user="mcheck" uid="478872" lat="44.4969905" lon="11.3540403">
 <tag k="addr:city" v="Bologna"/>
 <tag k="addr:country" v="IT"/>
 <tag k="addr:district" v="San Vitale"/>
 <tag k="addr:housenumber" v="4"/>
 <tag k="addr:postcode" v="40126"/>
 <tag k="addr:street" v="Via San Giacomo"/>
 <tag k="source" v="Comune di Bologna"/>
</node>
<node id="1704276521" visible="true" version="2" changeset="12221788" timestamp="2012-07-14T23:07:42Z" user="mcheck" uid="478872" lat="44.4970873" lon="11.3543321">
 <tag k="addr:city" v="Bologna"/>
 <tag k="addr:country" v="IT"/>
 <tag k="addr:district" v="San Vitale"/>
 <tag k="addr:housenumber" v="4/2"/>
 <tag k="addr:postcode" v="40126"/>
 <tag k="addr:street" v="Via San Giacomo"/>
 <tag k="source" v="Comune di Bologna"/>
</node>
```

Non è importante, nel nostro caso, trovare delle regole che mettano in relazione tali nodi, questo compito lo lasciamo infatti a chi inserirà in seguito le informazioni sui tratti di strada accessibili. Si vuole creare un'interfaccia

SQLite che caratterizzi il tipo nodo urbano, più precisamente ogni nodo, ben riconoscibile nei file OSM, deve essere descritto in una tabella di questo tipo:

ID	Via/Piazza	Numero civico	Latitudine	Longitudine
0	Via San Giacomo	2	44.4970473	11.3538566
1	Via San Giacomo	4	44.4969905	11.3540403
2	Via San Giacomo	4/2	44.4970873	11.3543321

Tabella 6.1: esempio di rappresentazione di un nodo cittadino

Tali dati dovranno, nella funzionalità di inserimento segmenti, essere visualizzati sulla mappa come gli altri overlay e dovranno tipicamente essere rappresentati da un punto. Al momento dell'inserimento di un nuovo segmento l'utente selezionerà tramite touchscreen i punti visibili in base al percorso che vuole segnalare, di fatto creando una linea che va ad unire tali punti. A questo punto servirà un'ulteriore interfaccia che metterà in relazione i diversi nodi cittadini. Questo dovrà essere rappresentato da una tabella, che indicherà per ogni nodo quali sono gli ID degli altri nodi collegati, ed il tipo di collegamento creato per ognuno di essi. Il tipo di strada sarà determinato dai criteri di valutazione spiegati nel capitolo precedente. Naturalmente dovrà essere possibile segnalare anche percorsi lunghi, e l'utente dovrà in maniera accurata indicare, nell'ordine corretto, tutti i punti che formano il percorso. In questo modo, si potrà creare la rete dei percorsi accessibili ai disabili. Ciò che si vuole ottenere è sostanzialmente un grafo che rappresenti tutti i percorsi accessibili della città, con evidenziate le principali caratteristiche di ogni arco. Indicando nel database la tipologia di collegamento creato sarà possibile effettuare delle richieste personalizzate, escludendo o includendo determinate caratteristiche del percorso; in questo modo verrà creato il percorso più adatto all'esigenza del disabile. Vediamo a questo punto quali sono i risultati, a livello applicativo, che vogliamo ottenere. Nella funzionalità di inserimento dati avremo la possibilità di individuare i nodi stradali:

Figura 6.1: nodi stradali rappresentati nella mappa

Se l'utente decide di inserire un segmento che collega i tre punti rappresentati in tabella 6.1 avremo la possibilità di vedere, dalla funzionalità di visualizzazione della mappa, un segmento di questo tipo:

Figura 6.2: il segmento visualizzato

Ciò che genera tale operazione è l'inserimento di alcuni dati significativi in una tabella che mette in relazione i nodi urbani in questione. In presenza di tali dati possiamo, attraverso l'algoritmo A*, calcolare percorsi ed evidenziarli utilizzando gli overlay di Mapsforge. Vediamo come sarà strutturata la tabella,

supponendo per ogni nodo un massimo di altri tre nodi collegati (nella realtà dovranno essere di più).

ID NODO	ID_LINK1	tipo_l1	ID_LINK2	tipo_l2	ID_LINK3	tipo_l3
0	1	3	-	-	-	-
1	0	3	2	3	-	-
2	1	3	-	-	-	-

Tabella 6.2: tabella per il routing

Per ogni link definito si specifica anche il tipo; ricordiamo che i tipi di strada sono suddivisi in quattro gradi diversi e che in questo caso parliamo della quarta tipologia, ovvero i tratti di strada consigliati dagli utenti, ma privi della verifica dei valori necessari alla valutazione.

7 Un'esperienza di simulazione

Il lavoro di simulazione che presento in questo capitolo è stato svolto allo scopo di eseguire uno studio sulle caratteristiche generali della nostra applicazione. Si tratta di una parte del progetto svolto per il corso di Simulazione tenuto dal Prof. Lorenzo Donatiello e dal Prof. Gabriele d'Angelo, nell'ambito della simulazione parallela e distribuita. Uno dei motivi principali che mi ha spinto a pensare ad un lavoro di simulazione come questo è stato quello di comprendere quanto potesse essere utile nella realtà la soluzione applicativa adottata. Si potrebbe affermare con un buon grado di certezza che la soluzione applicativa possa migliorare la situazione per le persone disabili dotate dell'applicazione rispetto a quelle prive della stessa. Se questo può essere affermato nel caso di ostacoli (barriere) fissi all'interno dell'area urbana, l'affermazione non vale sempre nel caso di ostacoli temporanei; non è detto, infatti, che un utente, che nel nostro caso si chiamerà city user, informato sugli ostacoli presenti in città possa arrivare prima di quello non informato, questo è dovuto soprattutto alla temporaneità degli ostacoli. Oltre a questo, il progetto svolto è stato importante per la definizione di un algoritmo di navigazine e una solida struttura dati creata a tale scopo. I dati utilizzati e le strutture sono molto simili a quelli presentati nel capitolo precedente, stessa cosa vale per la procedura di navigazione. Vediamo nelle sezioni sottostanti quali sono gli strumenti che si sono utilizzati, in particolare parleremo del simulatore e del middleware utilizzato; in seguito parleremo del modello creato, della sua implementazione e nella parte finale parleremo di come è stata pianificata la simulazione, della fase di raccolta dei dati e di analisi dei risultati finali.

7.1 Il simulatore

Gli strumenti utilizzati per questo lavoro provengono dal progetto ARTÌS / GAIA[13] del Dipartimento di Informatica - Scienza e Ingegneria del nostro Ateneo. Si tratta di un'implementazione nello standard Open Source rivolta alla creazione di un middleware, ARTÌS (Advanced RTI System), adattivo ed espressamente orientato alla simulazione parallela e distribuita. Oltre a questo si aggiunge il framework GAIA, sistema che fornisce un approccio alla simulazione basato sulla migrazione e la comunicazione fra le entità presenti nella simulazione. Il nostro lavoro è partito dal modello MIGRATION-WIRELESS presente nella Limited Edition di ARTÌS / GAIA disponibile per il download. Per completare il pacchetto di strumenti utili al nostro lavoro, è stato creato dal collega Giuseppe Balistreri un visualizzatore, ViewSim, che permette, attraverso un'interfaccia grafica OpenGL, di verificare l'andamento della simulazione. Infine, mi sono occupato della creazione degli script utilizzati per effettuare prove ripetute sulla simulazione in questione in modo da ottenere stima puntuale ed intervallare di alcune metriche prese in considerzione e che vedremo in seguito.

```
WIRELESS SIMULATION ...

Starting SIMA 1 ...
[LP 0] Starting wireless 0       ...
mahboo bound to port 5000
GAIA_____[     15413]: initialization
GAIA_____[     15413]: load balance code -> NOT included
ARTÌS-TS_[     15413]: initialization
COMM_____[     15413]: initialization
COMM_____[     15413]: cloning code -> NOT included
COMM_____[     15413]: compression -> NOT included
COMM_____[     15413]: marshalling -> included
COMM_____[     15413]: mpi interface -> NOT included
COMM_____[     15413]: multi-thread version
Recd connection #0
Group 0 has 1 nodes
Global Lookahead defined to "1.000000".
SIMA End ...
GAIA_____[     15413]: migration state -> 1
GAIA_____[     15413]: Migration Factor (MF) set to -> 3.000000

real    0m8.993s
user    0m8.653s
sys     0m0.200s
[0 OK]
```

Figura 7.1: Esecuzione della simulazione attraverso ARTÌS/GAIA

[13] Progetto disponibile su http://pads.cs.unibo.it/dokuwiki/doku.php?id=

7.2 Il modello

Il modello in questione si colloca nell'area di studio dei mobility model, modelli che rappresentano in maniera realistica il movimento di nodi mobili (Mobility Nodes o MN) all'interno di uno spazio con caratteristiche di diverso tipo. Generalmente vengono utilizzati per effettuare studi a livello network, ma non solo, essi sono utili per analisi in campo agricolo, militare e sociale, soprattutto per la gestione dei grandi eventi. In letteratura sono presenti molti modelli di mobilità, e tra i più conosciuti troviamo il Random Walk, Random Waypoint, Nomadic Community e City Section. Proprio da quest'ultimo parte la definizione del nostro modello. Innanzitutto va detto che esistono due tipi di modelli di mobilità: a entità (o cellulare), dove ogni nodo è indipendente rispetto agli altri nodi e modelli a gruppo, dove ogni nodo dipende dagli altri nodi facenti parte del proprio gruppo. Nel nostro caso parliamo di modello a entità, ma a differenza di molti altri modelli presenti in letteratura, il City Section è caratterizzato da alcune restrizioni dovute alla conformazione delle strade cittadine, si tratta infatti di un modello che prevede una serie di percorsi cittadini definiti e limitati che il city user è costretto ad attraversare per spostarsi da un punto ad un altro all'interno dell'area di simulazione. Vediamo nella figura seguente un esempio di movimento da parte di un MN nel caso del City Section:

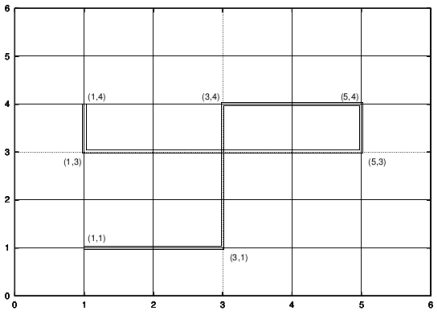

Figura 7.2: esempio di path di un MN nel modello City Section

Tale modello è stato adattato alle nostre esigenze, applicandolo ad uno scenario reale rappresentato dalla mappa della città di Bologna. L'area di simulazione è uno spazio limitato che rappresenta una sezione del centro di Bologna.
Il modello vuole descrivere una situazione urbana di tipo pedonale rappresentata da una serie di strade bidirezionali e può venire visto come una griglia composta da nodi (incroci) ed archi di collegamento (strade). Si assume che gli incroci siano privi di semafori e che i pedoni non si fermino nelle intersezioni durante il loro tragitto; inoltre la loro sovrapposizione nello spazio di simulazione è possibile. All'interno di questa mappa, abbiamo un insieme definito di punti di interesse (punti di aggregazione). Ogni city user si posiziona in maniera casuale in un punto su una strada e sceglie una destinazione (POI) all'interno della mappa. Viene effettuato il calcolo del percorso migliore attraverso un algoritmo che calcola il percorso minore e il city user si muove in direzione della sua destinazione, seguendo le strade a disposizione. La velocità dipende dalla strada sulla quale si trova il city user, infatti le strade sono di due tipi: strade ad alta e bassa velocità. In aggiunta a queste caratteristiche vengono inseriti in maniera random, durante la simulazione, degli ostacoli temporanei con diversa durata. Ogni city user, una volta raggiunto un ostacolo, torna al nodo precedente e dopo una fase di calcolo di un nuovo percorso, riprende il suo movimento in direzione della destinazione scelta in precedenza. Arrivato a destinazione il city user si ferma per un tempo random ed in seguito sceglie una nuova destinazione in modo casuale.
La simulazione vuole valutare l'impatto che potrebbe avere un'applicazione di navigazione e segnalazione di ostacoli per i pedoni (nello specifico per disabili). Molto interessante sarà il confronto dei risultati in termini di velocità media, distanza percorsa e numero di destinazioni raggiunte tra city user informati e quindi in possesso dell'applicazione e city user non informati. Il fatto che gli ostacoli e i momenti di pausa o ricalcolo del percorso abbiano una durata del tutto casuale, rende il nostro scopo non banale. Va detto che determinati range sono stati scelti attraverso un bilanciamento accurato; il range di durata degli ostacoli, ad esempio, rappresenta nel punto medio la media del tempo necessario ad un city user per partire da un punto ed arrivare a destinazione. Nel nostro caso di studio non è necessario suddividere la categoria city user

in ulteriori tipologie. Assumiamo infatti che le destinazioni siano di interesse comune e non vi siano sottoclassi di interesse. I city user per tutta la simulazione si muovono da un punto di interesse ad un altro. Oltre a questo si assume anche che si possa avere sempre a disposizione una rete mobile per l'aggiornamento di un database comune.

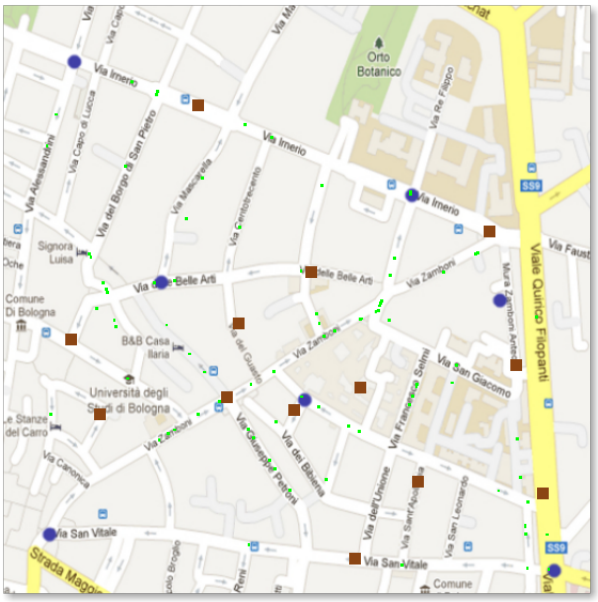

Figura 7.3: screenshot del visualizzatore della simulazione

La Figura 7.3 rappresenta uno screenshot del visualizzatore creato in OpenGL che ha permesso di effettuare accuratamente il debug della nostra simulazione. I punti blu rappresentano i POI comuni a tutti i nodi mobili, i blocchi marroni rappresentano gli ostacoli che si posizionano in maniera casuale in vari punti della mappa e ogni punto verde rappresenta un city user informato. Il city user non informato, che in questo caso non è presente, è rappresentato da un punto rosso. Per l'informato viene effettuato un calcolo che evita tutti gli ostacoli presenti in quel momento, mentre il non informato intraprende la via più breve per raggiungere la sua destinazione. Quando un utente informato incontra un nuovo ostacolo, egli tornerà indietro e rieffettuerà la richiesta di calcolo. A quel punto tutti gli altri nodi informati saranno a conoscenza del

nuovo ostacolo. Il non informato cambierà strada anch'egli, senza però poter segnalere tale problema.

7.3 L'analisi dei risultati

Le metriche prese in considerazione in fase di analisi servono a dare un giudizio generale alla nostra soluzione applicativa per disabili; i dati che si è pensato di rilevare sono i seguenti:

- **Velocità media**
- **Media delle destinazioni raggiunte**
- **Distanza media percorsa**

Va detto che, per quanto riguarda la pianificazione della simulazione, si è affrontato l'approccio delle prove ripetute; nel nostro caso, per ogni tipo di city user, si sono effettuate 100 run con 100 utenti e il numero di step di simulazione per ogni run è stato impostato a 5000; ad ogni step c'è la probabilità del 5% che si generi un nuovo ostacolo e la durata di un ostacolo è scelta in maniera casuale nell'intervallo [250, 750] step di simulazione. Nelle strade a bassa velocità, la velocità dell'utente si abbassa del 30%. Il generatore di numeri pseudocasuali che si è creato permette di ottenere gli stessi numeri per tutte le serie di run che si vogliono eseguire. Ad ogni run viene effettuato il calcolo delle medie, dopodichè con tali dati si passa al calcolo della stima puntuale, della varianza, della deviazione standard ed infine dell'intervallo di confidenza al 95 percentile.

Vediamo ora i risultati ottenuti:

CITY USER INFORMATO

velocità media = 23.962291;
varianza = 0.2188319332;
deviazione standard = 0.4677947554;
intervallo di confidenza =[media – 0.0916877721, media + 0.0916877721].

target raggiunti media = 9.3422;
varianza = 0.42529316;
deviazione standard = 0.6521450452;
intervallo di confidenza = [media – 0.1278; media + 0.1278].

distanza percorsa media = 119061.892145;
intervallo di confidenza = [media – 458.444159039, media + 458.444159039]

densità negli hotspot (POI) media = 0.2205571443;
varianza = 0.0065540433;
deviazione standard = 0.080957046;
intervallo di confidenza = [media – 0.015867581; media + 0.015867581].

CITY USER NON INFORMATO

velocità media = 23.754366;
varianza = 0.3230030238;
deviazione standard = 0.5683335498;
intervallo di confidenza = [media – 0.1113933758; media + 0.1113933758].

target raggiunti media = 6.1981;
varianza = 0.54318339;
deviazione standard = 0.7370097625;
intervallo di confidenza =[media – 0.1445; media + 0.1445].

distanza percorsa media = 118028.485407;
intervallo di confidenza = [media + 556.7519279174, media - 556.7519279174]

densità negli hotspot (POI) media = 0.21131569;
varianza = 0.0028501455;
deviazione standard = 340.053386754;
intervallo di confidenza = [media – 0.0104638038; media + 0.0104638038].

Vediamo ora i risultati espressi graficamente con l'ausilio del tool *gnuplot*:

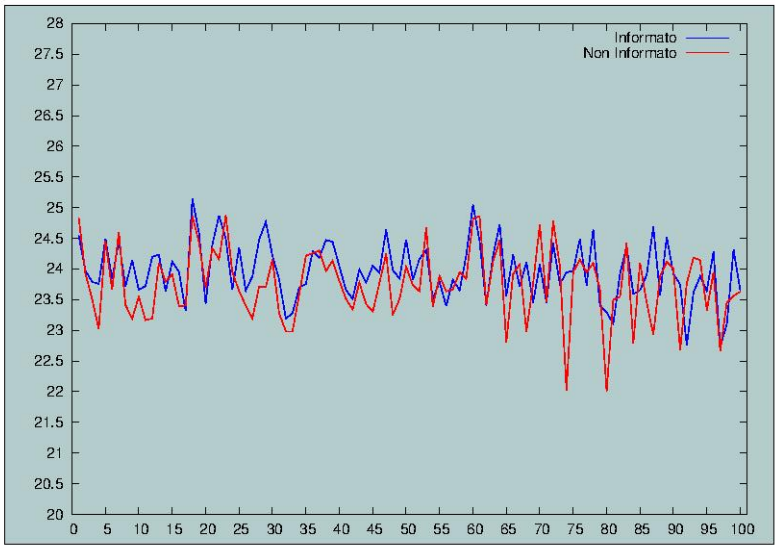

Figura 7.4: velocità media dei due tipi di city user - 100 run

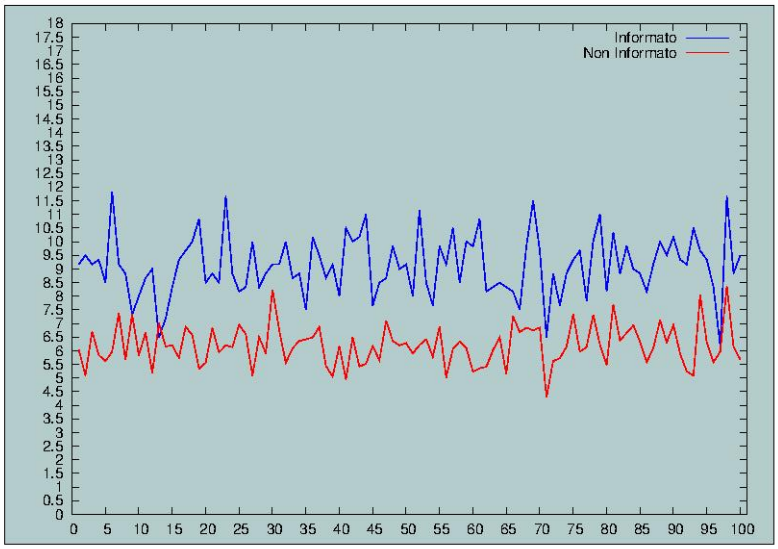

Figura 7.5: media delle destinazioni raggiunte dai due tipi di city user - 100 run

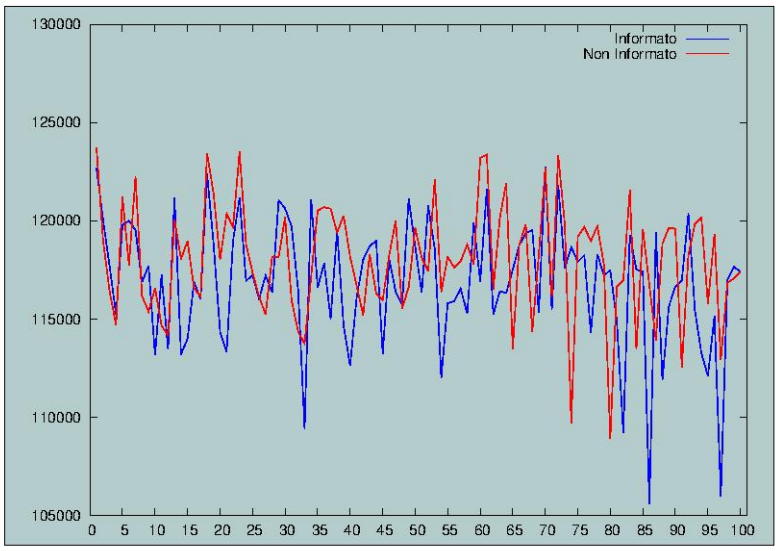

Figura 7.6: distanza media percorsa dai due tipi di city user - 100 run

Stima puntuale e gli intervalli di confidenza a confronto:

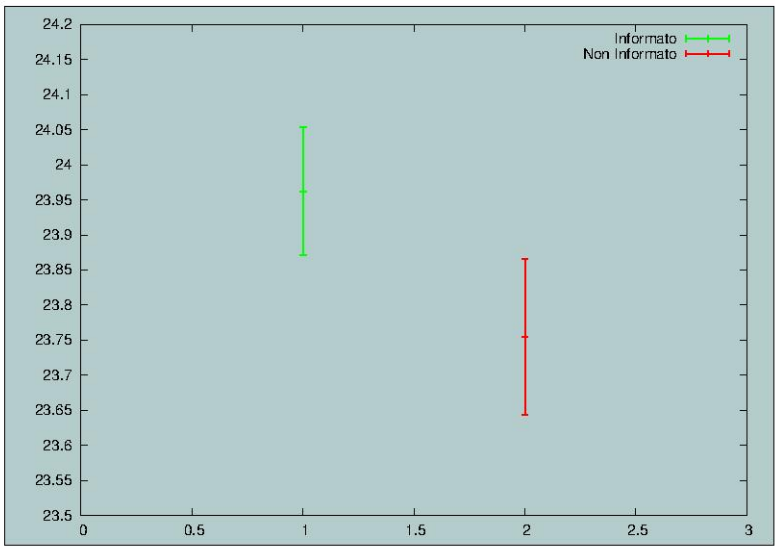

Figura 7.7: intervalli di confidenza - velocità

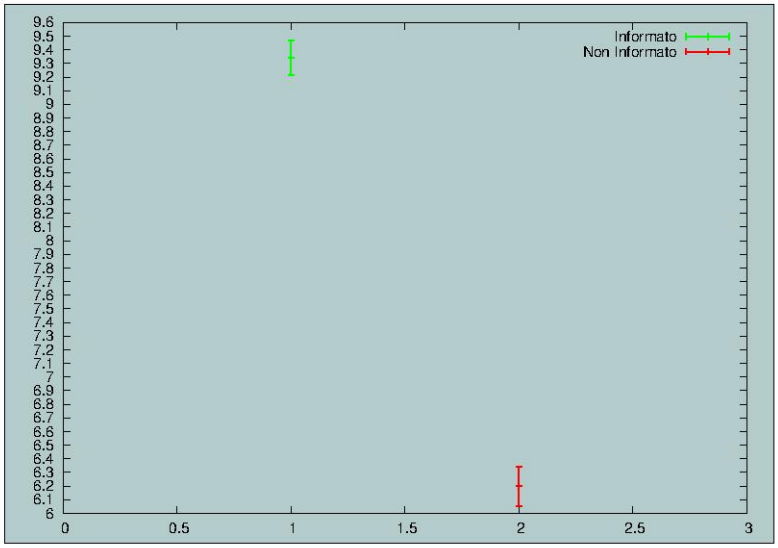

Figura 7.8: intervalli di confidenza - numero destinazioni raggiunte

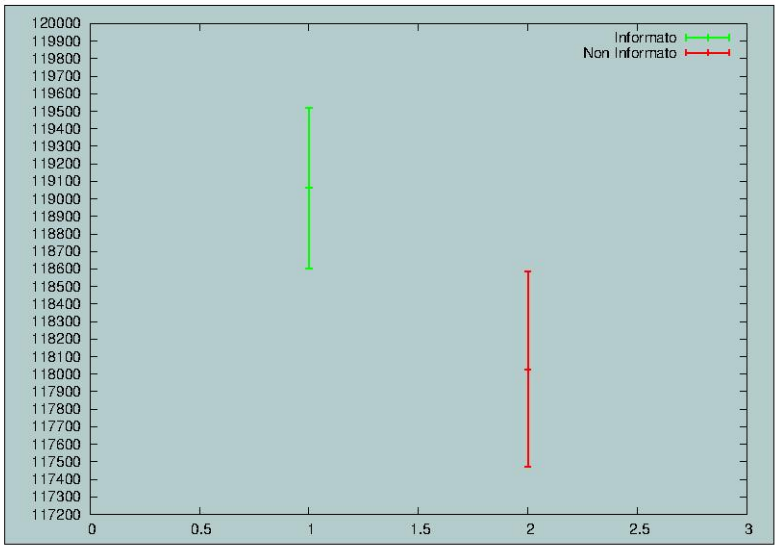

Figura 7.9: intervalli di confidenza - distanza percorsa

Come si può notare, la differenza sostanziale è data dal numero di target raggiunti. Questo è sicuramente un punto a favore della nostra soluzione applicativa, che migliora la situazione. Un fattore importante da considerare a questo proposito, e di cui non abbiamo ancora parlato è il fattore di rinuncia al raggiungimento del target. Ogni qual volta il city user informato, incontra un ostacolo o vuole ripartire in direzione di un nuovo target, viene richiamata la procedura di calcolo (navigazione); può succedere che la presenza di molti ostacoli, possa impedire la terminazione del calcolo e che, il target risulti impossibile da raggiungere. Se questo ricapita un certo numero di volte, nel nostro caso 5, si rinuncia a questo calcolo ed il city user sceglierà una nuova destinazione. Questo, a mio avviso, è un fattore importante e, soprattutto a livello sociale, permette alle persone di evitare la perdita di tempo e la spiacevole esperienza di non essere in grado di raggiungere una destinazione. In questi particolari casi, in cui il city user informato si trova in punti critici della città (molti ostacoli), il grafico dei target tende a scendere per l'informato, arrivando ad un numero pari o inferiore a quello del non informato; questo è direttamente connesso ad un forte numero di rinunce di raggiungimento di un target. Per quanto riguarda il non informato, esso intraprenderà comunque il tragitto verso la destinazione, probabilmente raggiungendola dopo molto tempo (temporaneità ostacoli), ma con un fattore di stress molto elevato.

8 Conclusioni

L'applicazione creata è al momento ad una fase prototipale, per renderla fruibile in futuro sarà infatti necessario avere la possibilità di aggiornare sistematicamente il database online, permettendo a tutti gli effetti la condivisione delle informazioni; gli argomenti da trattare a tal proposito sono molti, ai fini della tesi si è scelto di trattare solamente le problematiche a livello locale (device), ma la volontà è quella di procedere con tale progetto, cercando di mettere in comunicazione associazioni del settore e amministrazioni locali. Il progetto Mapsforge è di fondamentale importanza per lo sviluppo di tale applicazione ed è tutt'ora in fase di sviluppo, sarà quindi importante seguire i cambiamenti di questo progetto, soprattutto nel campo della navigazione; il nostro progetto, in questo campo, propone una soluzione che dovrà essere sicuramente approfondita e migliorata; dobbiamo ad esempio considerare il fatto che in molti casi gli accessi pedonali ad una strada si trovano in entrambi i lati e che i passaggi pedonali e le rampe devono essere considerati quali parti integranti dei nodi cittadini, per questo è importante al momento riuscire a tracciare ed avere un database con tutti questi dati; anche a tale scopo si è pensato ad un'applicazione come questa. Al momento la segnalazione di strade accessibili, abbinata all'aggiunta dei marker, permette un miglioramento della situazione attuale per i disabili, ma i passi da fare in avanti sono ancora molti.

Indice Figure e Tabelle

Riferimenti Bibliografici

[1] James Steele and Nelson To. *The Android Developer's Cookbook, building Applications with the Android SDK*, Addison-Wesley, 2010.

[2] Paul Deitel, Harvey Deitel, Abbey Deitel, Michael Morgano. *Sviluppare App per Android*, PEARSON, 2012.

[3] Gabbrielli M., S. Martini, *Linguaggi di Programmazione: Principi e Paradigmi*, McGraw-Hill, Milano, 2006.

[4] A.M. Law, W.D. Kelton, *Simulation Modeling and Analysis*, Mac Graw-Hill, 1982.

[5] Alessandro Solipaca, *La disabilità in Italia, il quadro della statistica ufficiale*, Istat, Argomenti n. 37, 2009.

[6] G. D'Angelo and M. Bracuto, *Distributed simulation of large-scale and detailed models*, Int. J. Simulation and Process Modelling, Vol. 5, No. 2, 2009.

[7] Wikipedia - The Free Encyclopedia. "*Mobility Model*". http://en.wikipedia.org/wiki/Mobility_model, 2013.

[8] Mapsforge - free mapping and navigation tools. http://code.google.com/p/mapsforge/.

[9] Il portale dei disabili italiani. http://www.disabili.com/.

[10] OpenstreetMap Wiki. http://wiki.openstreetmap.org/wiki/Main_Page.