Università degli Studi di Perugia
A.A. 2016/2017

DMImap!

Progetto di Ingegneria del Software

A cura di:

Bartoli Luca
Matr. 281606

Battistelli Annalisa
Matr. 281640

Sommario

Obiettivo

L'applicazione nasce dalla proposta di progetto “Mappa Indoor Path Finding tramite QRCode” ed è sviluppata per dispositivi con sistema operativo Android utilizzando l'ambiente di sviluppo integrato (IDE) Android Studio.

DMImap!, attraverso un layout semplice ed intuitivo, permette di trovare il percorso più vantaggioso da un punto a un altro in alcuni piani del Dipartimento di Matematica e Informatica (DMI, da qui il nome dell'app) dell'Università degli Studi di Perugia.

Mediante la lettura di QRCode distribuiti nelle aule, negli uffici e in altri punti che potrebbero risultare utili a ipotetici fruitori (come ad esempio l’ingresso e la portineria) è possibile localizzare la propria posizione all'interno dell'edificio.
Fatto ciò, l'applicazione permette di inserire il nome del luogo in cui si vuole arrivare e di selezionare il tipo scegliendo tra “ufficio”, “aula” o “altro” per poi costruire il percorso migliore dal punto di partenza al punto inserito.
Questo viene visualizzato in rosso sulla mappa.

Analisi

Si è scelto di offrire un'interfaccia molto elementare: due form, uno per la posizione, che comprende una TextView (non editabile, utilizzata come label) e un Button (in alto), e uno per la destinazione, che comprende una TextView (in questo caso editabile) e tre RadioButton per impostare il tipo (in basso); al centro abbiamo posto una View, che viene utilizzata come Canvas, su cui viene caricata la mappa, e infine in basso a sinistra un Button.
L'applicazione può essere adattata a qualsiasi mappa, dal momento che all'avvio essa va a caricare dalle risorse di tipo Drawable le cartine (con la sola condizione che i file devono chiamarsi "floor_<i>.jpg" con <i> il numero del piano, che deve partire da 0), e stampa il piano 0 sulla canvas, mentre va a leggere da un file JSON (che deve chiamarsi "map_description.json"), la descrizione del grafo che modellizza la mappa (che chiaramente deve rispettare la sintassi JSON e la semantica specifica dell'applicazione).

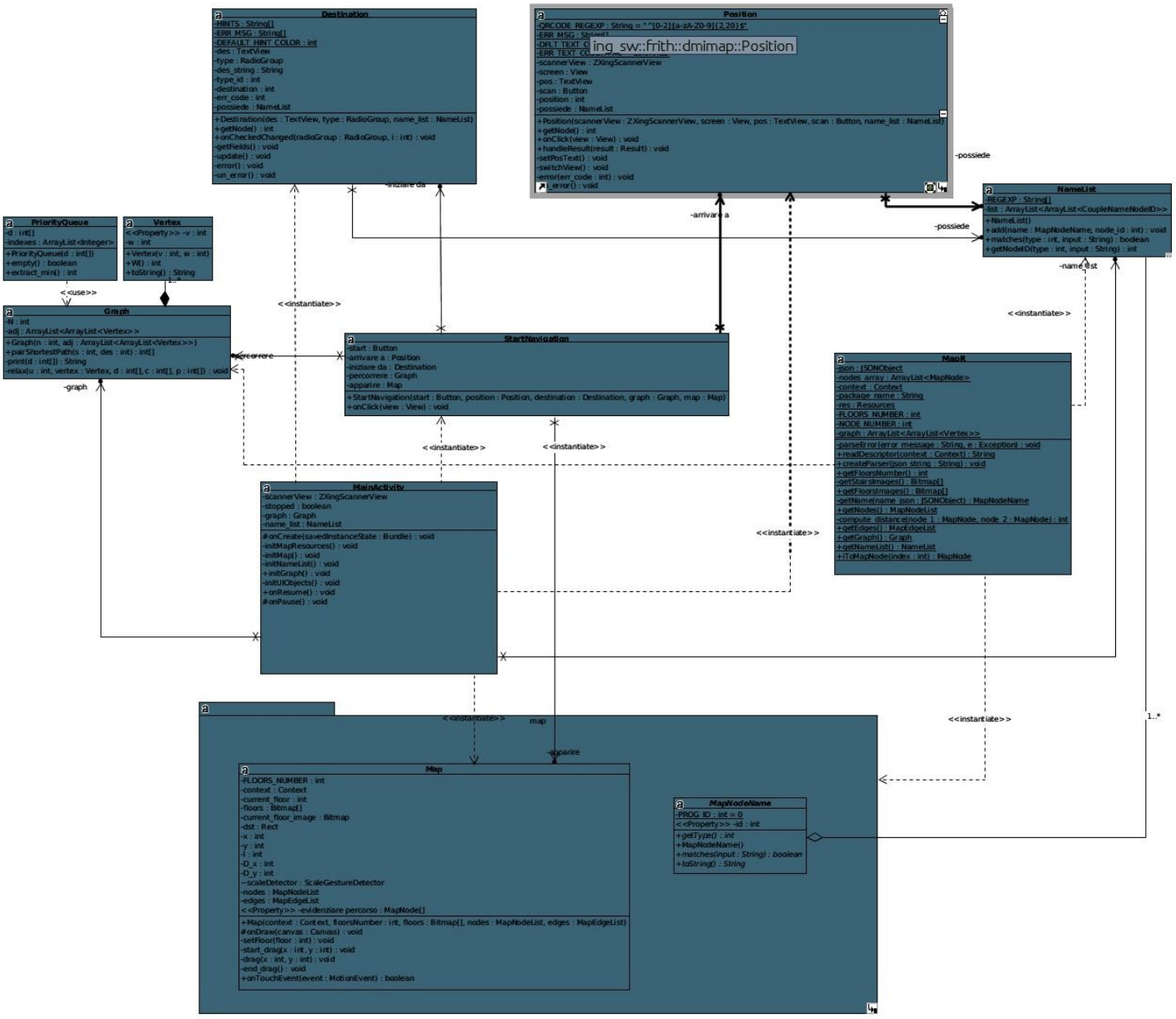
Il file JSON contiene un array "nodes" diviso a sua volta in altri due array contenenti ognuno i nodi all'interno di un piano e un altro array "edges" diviso anch'esso a sua volta in due array, contenenti ciascuno gli archi di un piano.
All'interno di ciascuno dei due array vi sono degli oggetti rappresentanti i nodi del grafo. Questi nodi sono di vario tipo.
I nodi "named" sono quelli che indicando dei luoghi di partenza/arrivo; graficamente sono rappresentati con dei cerchi pieni. Essi sono costituiti dai campi "x" e "y", ovvero la loro posizione all'interno della mappa in percentuale, e un altro campo specifico a seconda del sottotipo di nodo: per i nodi di tipo ufficio si utilizza il campo "office", contenente un array dei cognomi dei professori in un determinato ufficio, per quelli di tipo aula "classroom", contenente la stringa rappresentante i nomi delle aule, e per gli altri il campo "other".
Se possiedono solamente i campi "x" e "y", i nodi sono incroci, perciò non hanno nome; non rappresentando un luogo d'arrivo essi vengono utilizzati solo per disegnare la mappa, pur non essendo visibili graficamente.
Infine i nodi per le scale, oltre alle coordinate "x" e "y", hanno un campo "stairs" che è serve per distinguere i casi in cui bisogna scendere o salire

le scale. Essi nella mappa si distinguono per l'icona che rappresenta le scale.
è true, altrimenti false.
Nell'array "edges" le coppie specificano gli archi, le triple anche il peso dell'arco. Se non viene specificato il peso allora viene calcolato, altrimenti viene utilizzato quello indicato. Il caso in cui occorre specificare il peso è quello dei nodi su due piani diversi (cioè quelli che collegano nodi di tipo scala).

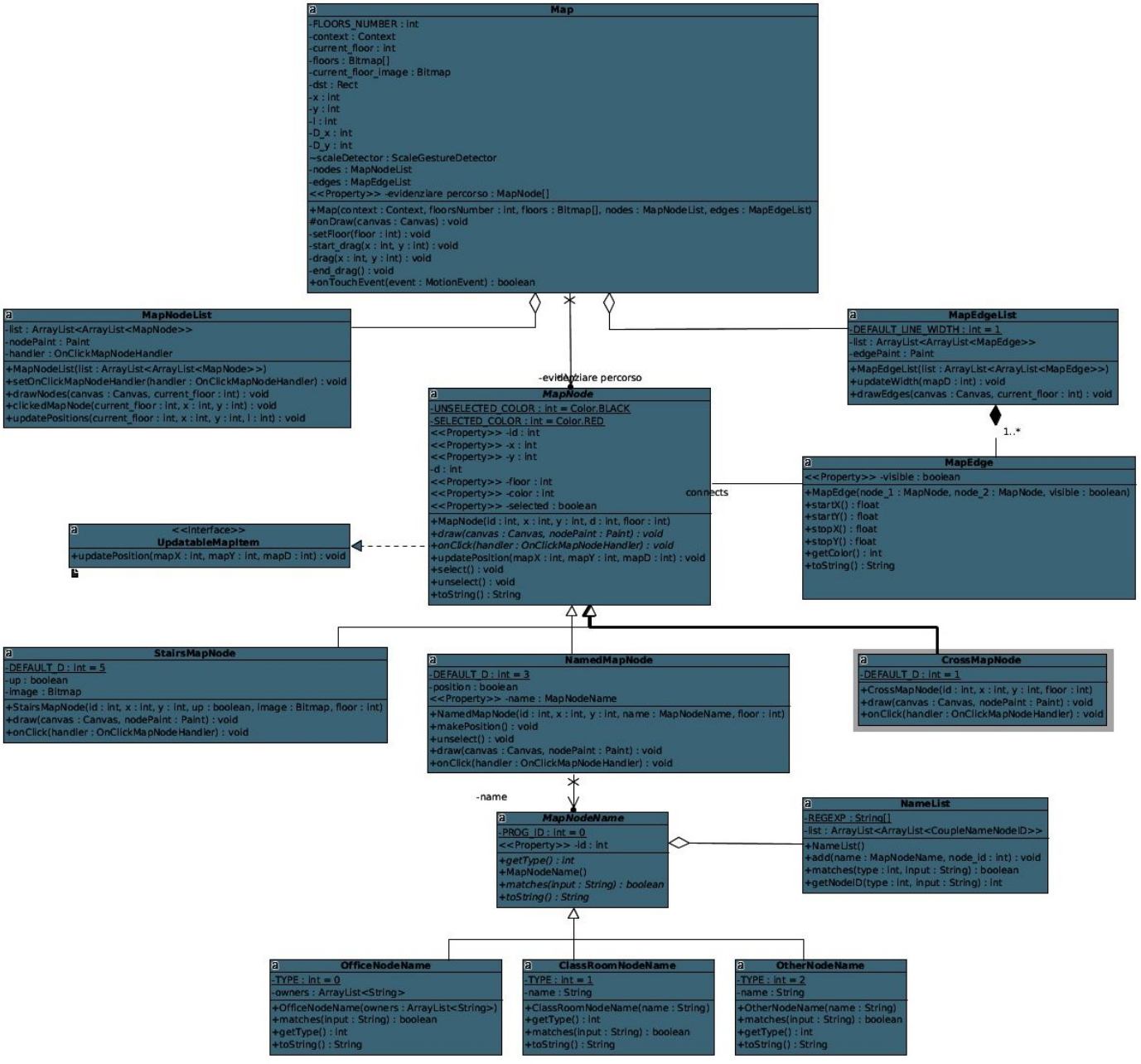
Per la localizzazione, l'applicazione si basa sulla lettura di QRCode: ognuno di questi serve per identificare uno specifico luogo. Questa operazione si effettua semplicemente premendo il Button "SCAN" in alto a destra, il quale avvia una nuova View, quella dello scanner, per l'implementazione della quale è stata utilizzata la libreria apposita della ZXing Team. Il risultato della lettura è caricato nella TextView sotto forma di testo.
Una volta fatto ciò, si può procedere impostando, nel form in basso, il tipo di nodo che si vuole raggiungere ("ufficio", "aula" o "altro") , e scrivendo il nome del luogo nella TextView.
A questo punto, se sia la posizione che la destinazione sono validi, allora si può avviare la navigazione premendo "START". L'applicazione elabora lo shortest path utilizzando l'algoritmo di Dijkstra. Il path risultante diventa visibile sulla mappa, anche qualora si estenda tra più piani (in tal caso deve essere l'utente a cliccare l'icona delle scale nel momento in cui le raggiunge, per visualizzare il resto del percorso sull'altro piano).

Diagramma di classe

package principale:



sottopackage map:



Un caso d'uso

Caso d’uso: Avvio navigazione

Attore primario: Utente

Precondizioni: nessuna

Postcondizioni: l’utente ottiene il percorso migliore dal luogo di partenza alla destinazione scelta.

Scenario principale:
1. L’utente imposta la posizione scannerizzando il QRCode, mediante la pressione dell’apposito bottone
2.L’utente imposta il tipo del luogo che desidera raggiungere scegliendo tra i tre radio buttons
2. L’utente inserisce il nome del luogo che vuole raggiungere
3. L’utente clicca l’icona di avvio della navigazione

Estensioni:
a. Nel caso in cui l’utente provi a scannerizzare un QRCode del formato non permesso, o del formato permesso ma di una posizione inesistente:
   1. viene stampato un messaggio d’errore nel form della posizione che ricorda all’utente di immettere la posizione
   2. viene data la possibilità di impostare la posizione con il QRCode scanner

b. Nel caso in cui l’utente provi ad avviare la navigazione senza aver impostato la posizione di partenza:
   1. viene stampato un messaggio d’errore nel form della posizione che ricorda all’utente di immettere la posizione
   2. viene data la possibilità di impostare la posizione con il QRCode scanner

c. Nel caso in cui l’utente scrive il luogo di destinazione ma non imposta il tipo, una volta cliccato il pulsante per l’avvio della navigazione:
   1. viene stampato un messaggio d’errore al posto della stringa che informa l’utente della necessità di inserire il tipo

2. viene data la possibilità di inserirle nuovamente stringa e tipo

d. Nel caso in cui l'utente inserisce una stringa non valida per il tipo selezionato (nel caso in cui sia vuota, o non corrisponda ai parametri) una volta cliccato il pulsante per l'avvio della navigazione:

1. viene stampato un messaggio d'errore al posto della stringa che informa l'utente della non validità della stringa inserita
2. viene data la possibilità di inserirle nuovamente stringa e tipo

Diagramma di sequenza

Diagramma relativo al caso d'uso sopra descritto, mostra come gli oggetti reagiscono agli eventi dell'utente dall'immissione della posizione fino all'avvio della navigazione (i nomi dei metodi nel codice sono diversi, sono stati cambiati per essere più chiari):

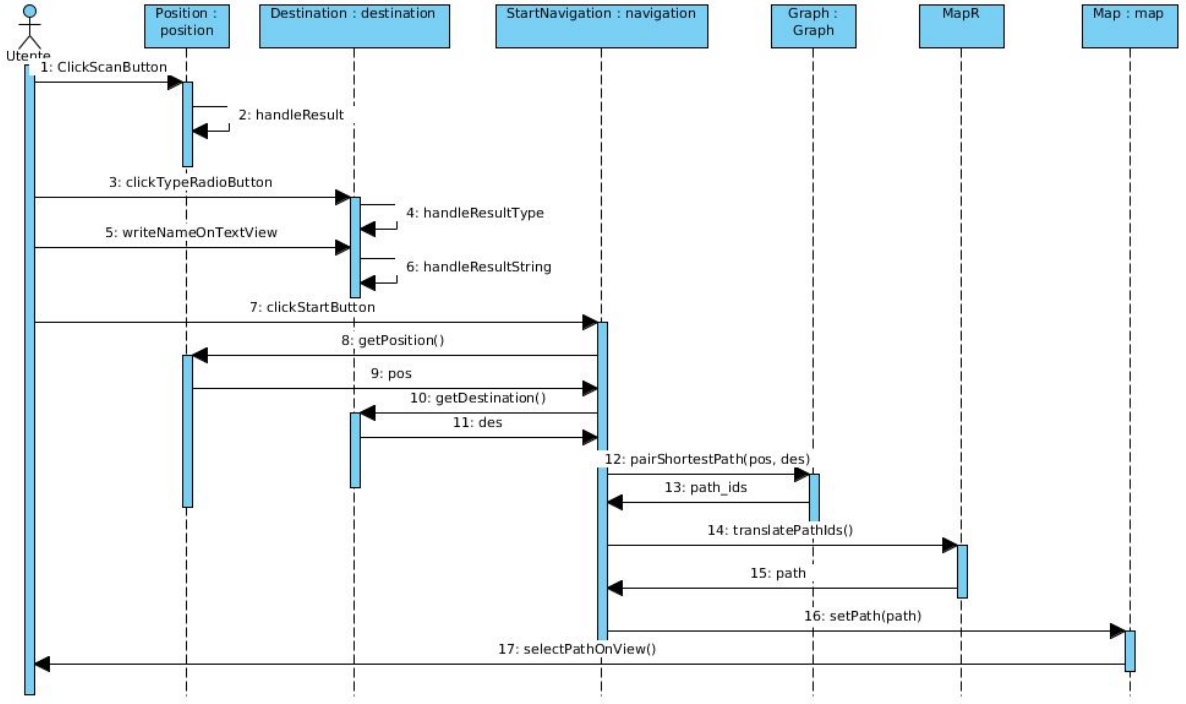


Diagramma di stato

Il diagramma che segue ritrae le varie fasi dell'utilizzo dell'applicazione, a partire dall'apertura della finestra fino alla sua chiusura, prevedendo anche eventuali casi di errore:

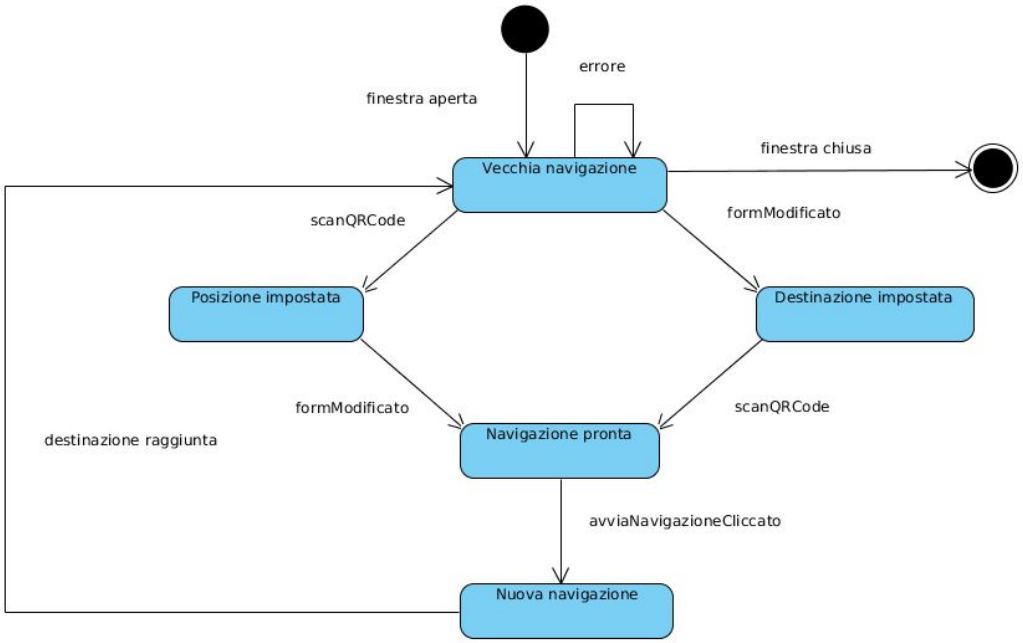
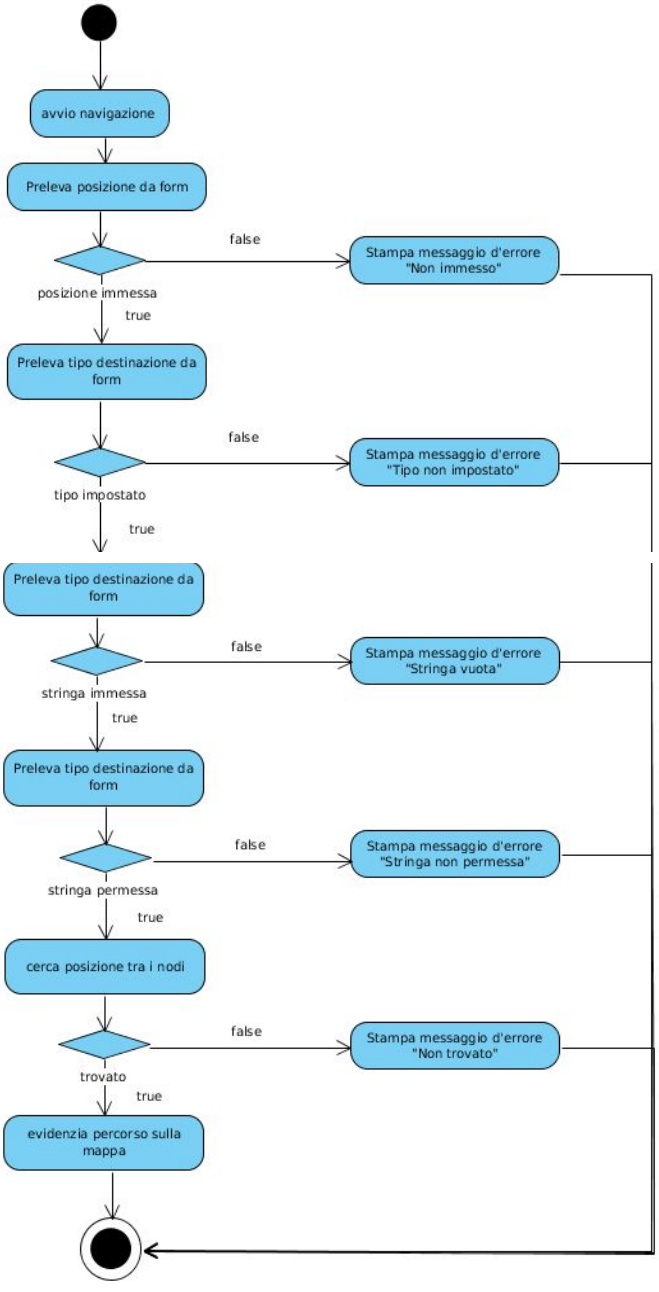


Diagramma di un'attività

Il diagramma seguente mostra i vari controlli che l'applicazione compie quando viene premuto il pulsante avvio navigazione. Si è ritenuto di mostrarlo con più chiarezza con tutti i controlli in cascata:



Codice sorgente

Di seguito sono inserite alcune parti del codice sorgente ritenute significative.

Classe MainActivity

```
public class MainActivity extends AppCompatActivity{

    private Graph graph;
    private ZXingScannerView scannerView;
    private boolean stopped;
    private Map map;
    private NameList name_list;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);



        initMapResources();
        initMap();
        initNameList();
        initGraph();
        initUIObjects();

    }




    private void initMapResources() {

        String description = MapR.readDescriptor(this);
        MapR.createParser(description);
    }
```

```
  private void initMap() {

     map = new Map(this, MapR.getFloorsNumber(), MapR.getFloorsImages(),
MapR.getNodes(), MapR.getEdges());
     LinearLayout map_window     = (LinearLayout)   findViewById(R.id.map_window);
     map_window.addView(map);

  }




  private void initNameList() {

     name_list = MapR.getNameList();

  }




  public void initGraph() {

     graph = MapR.getGraph();

  }




  private void initUIObjects() {


     scannerView =new ZXingScannerView(this);
     scannerView.setVisibility(View.INVISIBLE);
     addContentView(scannerView, new
ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.MATCH_PARENT));


     View screen        =              findViewById(R.id.screen);
     TextView pos       = (TextView)      findViewById(R.id.pos);
     Button   scan      = (Button)        findViewById(R.id.scan);



     Position position    = new Position(scannerView, screen, pos, scan, name_list);

     TextView   des       = (TextView)      findViewById(R.id.des);
     RadioGroup type      = (RadioGroup)    findViewById(R.id.type);

     Destination destination = new Destination(des, type, name_list);
```

```
        Button start      = (Button)        findViewById(R.id.start);

        new StartNavigation(start, position, destination, graph, map);
    }


    @Override
    public void onResume() {
        super.onResume();

        if(stopped) {

            scannerView.startCamera();
            stopped = false;

        }


    }


    @Override
    protected void onPause() {
        super.onPause();

        if(scannerView.getVisibility() == View.VISIBLE) {

            scannerView.stopCamera();
            stopped = true;

        }


    }
}
```

Classe Graph

```
public class Graph {


    private final int N;
```

```
private ArrayList<ArrayList<Vertex>> adj;


public Graph(int n, ArrayList<ArrayList<Vertex>> adj) {

   this.N = n;
   this.adj = adj;
}


public int[] pairShortestPath(int s, int des) {

   int[] d = new int[N];
   int[] c = new int[N];
   int[] p = new int[N];

   for(int i = 0; i < N; i++) {

      d[i] = Integer.MAX_VALUE;
      p[i] = -1;

   }

   d[s] = 0;
   c[s] = 0;

   PriorityQueue Q = new PriorityQueue(d);

   boolean flag = false;
   int u = -1;

   while(!Q.empty() && !flag) {

      u = Q.extract_min();ì

      if(u == des) {

         flag = true;

      } else {


         for (Vertex vertex : adj.get(u)) {

            relax(u, vertex, d, c, p);

         }

      }

   }


   int[] path = new int[c[u] + 1];
```

```
    int i = u, j = c[u];

    while(i != -1) {

        path[j] = i;
        i = p[i];
        j--;

    }

    return path;
}
```

```
private void relax(int u, Vertex vertex, int[] d, int[] c, int[] p) {


    int v = vertex.getV(), w = vertex.W();
    int sum;


    if(d[u] == Integer.MAX_VALUE) {

        sum = d[u];

    } else {

        sum = d[u] + w;

    }


    if(d[v] > sum) {

        d[v] = sum;
        c[v] = c[u] + 1;
        p[v] = u;

    }

}
```

```
  private class PriorityQueue {

     private int[] d;
     private ArrayList<Integer> indexes;

     public PriorityQueue(int[] d) {

        this.d = d;

        indexes = new ArrayList<>();

        for(int i = 0; i < N; i++) {

           indexes.add(i);

        }

     }

     public boolean empty() {

        return indexes.isEmpty();
     }


     public int extract_min() {

        int min_v = -1, min_i = -1, min = Integer.MAX_VALUE;

        for(int i = 0; i < indexes.size(); i++ ) {

           int v = indexes.get(i);

           if(d[v] < min) {

              min_i = i;
              min_v = v;
              min = d[v];
           }

        }


        indexes.remove(min_i);

        return min_v;

     }
  }


}
```

File JSON


```
{
 "floors_number" : 2,

 "nodes":
 [
  [
   {
    "name": {
     "classroom": "B1"
    },
    "x": 35,
    "y": 41
   },
   {
    "name": {
     "classroom": "A2"
    },
    "x": 11,
    "y": 50
   },
   {
    "name": {
     "office": ["Capria", "Pambianco"]
    },
    "x": 56,
    "y": 56
   },
   {
    "name": {
     "office": ["Poggioni"]
    },
    "x": 67,
    "y": 56
   },
   {
    "name": {
     "other": "ingresso"
```

```
  },
  "x": 49,
  "y": 68
},
{
  "x": 17,
  "y": 31,

  "stairs": true
},



{
  "x": 35,
  "y": 46
},
{
  "x": 11,
  "y": 46
},
{
  "x": 17,
  "y": 46
},
{
  "x": 49,
  "y": 46
},
{
  "x": 63,
  "y": 56
},
{
  "x": 63,
  "y": 52
},
{
  "x": 49,
  "y": 52
},

{
  "name": {

    "other": "portineria"

  },
  "x": 44,
  "y": 52
},
{
  "x": 44,
  "y": 46
},
{
  "x": 64,
  "y": 46,
```

```
    "stairs": true
  }


],
[
  {
    "name": {

      "classroom": "B2"

    },
    "x": 40,
    "y": 42
  },
  {
    "name": {

      "classroom": "D2"

    },
    "x": 32,
    "y": 42
  },
  {
    "name": {

      "classroom": "C2"

    },
    "x": 55,
    "y": 62
  },
  {
    "name": {

      "office": ["Gerace", "Stramacci"]

    },
    "x": 72,
    "y": 63
  },
  {
    "name": {

      "office": ["Carpi", "Guerra"]

    },
    "x": 75,
    "y": 60
  },
  {
    "x": 6,
    "y": 29,

    "stairs": false
  },
```

```
    {
      "x": 72,
      "y": 50,

      "stairs": false
    },
    {
      "x": 6,
      "y": 36
    },
    {
      "x": 14,
      "y": 36
    },
    {
      "x": 14,
      "y": 47
    },
    {
      "x": 37,
      "y": 47
    },
    {
      "x": 37,
      "y": 42
    },
    {
      "x": 63,
      "y": 47
    },
    {
      "x": 63,
      "y": 50
    },
    {
      "x": 63,
      "y": 56
    },
    {
      "x": 55,
      "y": 56
    },
    {
      "x": 72,
      "y": 56
    },
    {
      "x": 72,
      "y": 60
    },
    {
      "x": 55,
      "y": 47
    }
  ]
],
"edges":
[
```

```
  [
    [1,7], [7,8], [8,5], [8,6], [6,0], [6,14], [9,12], [12,4], [12,11], [11,10],
    [10,2], [10,3], [13,14], [9,15], [9,14], [5, 21, 8], [15,22,8]
  ],
  [

    [17,27], [16,27], [27,26], [21,23], [23,24], [24,25], [25,26], [26,34], [34,31],
    [31,18], [31,30], [30,29], [29,28], [28,34], [29,22], [30,32], [32,33], [33,19],
    [33,20]
  ]
]

}
```

Casi di test funzionale

E' stato scelto di fare i casi di test funzionale per la maschera di input per l'inserimento della destinazione.
Tra parentesi vengono indicati con (V) le classi di equivalenza valide, con (NV) le classi di equivalenza non valide. Le classi sono numerate per comodità.

Radio Buttons - Tipo Boolean

Intervalli:

Una nota da fare è che i radio buttons permettono solamente di fare scelte esclusive. Perciò gli unici casi da prendere in considerazione sono quello in cui non ne viene selezionato nessuno e quello in cui ne viene selezionato solamente uno.

```
    0        1
    |        |
(NV1)    (V1)
```

Classi:

- Ufficio T (V2) F (V3)
- Aula T (V4) F (V5)

- Altro T (V6) F (V7)

<u>Stringa - Tipo String</u>

Il massimo del contenitore è 30.
La stringa ha numero di valori in input ammissibili dipendente dal radio button selezionato:

<u>Intervalli</u>:

- Stringa per Ufficio: 0_____1 2_____20 21_____30
  (NV2) (V8) (NV3)
- Stringa per Aula: 0_____1 2 3_____30
  (NV4) (V9) (NV5)
- Stringa per Altro: 0_____1 2_____20 21_____30
  (NV6) (V10) (NV7)

<u>Composizione stringhe</u>:

- Stringa per Ufficio:
  - stringa di soli caratteri alfabetici (V11)
  - stringa contenente caratteri non alfabetici (NV8)
- Stringa per Aula:
  - stringa composta di una lettera maiuscola e un numero (V12)
  - stringa diversa dal formato precedente (NV9)
- Stringa per Altro:
  - stringa di soli caratteri alfabetici (V13)

- stringa contenente caratteri non alfabetici(NV10)

CASI DI TEST - INPUT VALIDO

Stringa	Ufficio	Aula	Altro	
1. “Poggioni”	T	F	F	(V8) (V2) (V5) (V7) (V11)
2. “A2”	F	T	F	(V9) (V3) (V4) (V12)
3. “portineria”	F	F	T	(V10) (V6) (V13)

CASI DI TEST - INPUT NON VALIDO

Stringa per Ufficio	Ufficio	Aula	Altro	
1. “”	T	F	F	(NV2)
2. “abababababababababababababa”	T	F	F	(NV3)
3. “Po1234”	T	F	F	(NV8)

Stringa per Aula	Ufficio	Aula	Altro

4. “”	F	T	F	(NV4)
5. “abababababababababababababa”	F	T	F	(NV5)
6. “aA”	F	T	F	(NV9)

Stringa per Altro	Ufficio	Aula	Altro	
7. “”	F	F	T	(NV6)
8. “abababababababababababababa”	F	F	T	(NV7)
9. “port1234” (NV10)	F	F	T	

Design Pattern

-Iterator

Iterator è un design pattern comportamentale, il cui scopo è quello di fornire un modo per scandire gli elementi di una classe senza esporne il contenuto, e mantenendo un’interfaccia ridotta. Nella realizzazione del progetto, allo scopo di gestire a parte le funzionalità di alcuni gruppi di oggetti (di tipo MapNode, MapEdge, MapNodeName, che sono rispettivamente i vertici sulla cartina, gli archi e i nomi dei nodi con QRCode), si è scelto di creare classi (MapNodeList, MapEdgeList, NameList) nelle quali sono state inserite: le liste degli oggetti corrispondenti come campi privati e metodi pubblici con i quali applicare un’operazione su tutti gli oggetti della lista o ad un sottoinsieme. Si sarebbe potuto in alternativa, implementare il design pattern Iterator,

fornendo una modalità di accesso sequenziale agli oggetti. Se ne offre una possibile implementazione:

```
public class MapNodeList implements Iterable<MapNode>{

  private ArrayList<ArrayList<MapNode>> list;
  private int current_floor;

  //    ...

  @Override
  public Iterator<MapNode> iterator() {

     return new Iterator<MapNode>() {

        private int i = 0;

        @Override
        public boolean hasNext() {

           return i < list.get(current_floor).size();
        }


        @Override
        public MapNode next() {
           MapNode mapNode = list.get(current_floor).get(i);
            i++;
           return mapNode;
        }
     };
  }

}
```

Si osserva che l'iteratore permette di scandire i nodi che fanno parte del piano attuale (current_floor), ci sarà quindi al posto dell'ellissi, fra gli altri metodi, un metodo "setCurrentFloor(int current_floor)".
Come da design pattern, la classe Iterator (qua creata come classe anonima interna a MapNodeList) dispone di un metodo che serve per stabilire se la lista è stata scandita interamente e uno per ottenere il successivo elemento della lista. La classe MapNodeList, sempre come

vuole il design pattern offre un factory method per ottenere un'istanza dell'iteratore.
Si possono sfruttare le funzionalità dell'iteratore in due modi (per entrambi si suppone di aver gia creato un'istanza di MapNodeList, il cui riferimento sta in mapNodeList):

```
Iterator<MapNode> iterator = mapNodeList.iterator();

while(iterator.hasNext()) {

  MapNode mapNode = iterator.next();

  // operazioni con mapNode

}
```

altrimenti si può sfruttare il costrutto forEach offerto da java, che rende la scansione ancora più semplice:

```
for(MapNode mapNode : mapNodeList) {


  // operazioni con mapNode

}
```

Glossario

Algoritmo di Dijkstra - è un algoritmo utilizzato per cercare gli shortest paths (ovvero i cammini di costo minimo) da sorgente unica verso tutti gli altri vertici in un grafo pesato, con pesi non negativi. Tale algoritmo trova applicazione in svariati contesti, come l'ottimizzazione delle reti.

Button - particolare tipo di View (vedi View), che offre le funzionalità di un pulsante. La classe Button estende View, e sta nel package android.widget.

Canvas - una canvas, in generale, è un'area rettangolare sullo schermo sulla quale l'applicazione può disegnare, e che può gestire eventi legati all'interazione con l'utente.

Design Pattern - nell'ambito dell'Ingegneria del Software può essere definito come una soluzione progettuale generale ad un problema ricorrente. Si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi durante le fasi di progettazione e sviluppo del software.

Grafo - in matematica è la configurazione formata da un insieme di punti (detti vertici o nodi) e un insieme di linee (dette archi) che uniscono coppie di nodi.

IDE - è un ambiente di sviluppo integrato (in lingua inglese Integrated Development Environment) è un software che, in fase di programmazione, aiuta i programmatori nello sviluppo del codice sorgente.

JSON - acronimo di JavaScript Object Notation, è un formato adatto all'interscambio di dati fra applicazioni client-server, e in generale alla serializzazione di oggetti.

Label - dall'inglese etichetta, area rettangolare su uno schermo nella quale un'applicazione può far apparire testo e immagini, e a cui possono essere associati sensori d'eventi.

RadioButton - particolare tipo di View (vedi View), detti radio per la forma, generalmente sono utilizzati per la selezione di una tra più voci, generalmente infatti sono raggruppati e hanno la caratteristica di poter essere selezionati solo uno alla volta.

Path finding - in informatica è la ricerca di un cammino tra nodi di un grafo.

QRCode - (abbreviazione di Quick Response Code) è un codice a barre bidimensionale composto da moduli neri disposti all'interno di uno schema di forma quadrata. Viene impiegato per memorizzare informazioni generalmente destinate a essere lette tramite la fotocamera di un dispositivo (ad esempio di uno smartphone o di un tablet).

TextView - particolare tipo di View (vedi View), che offre le funzionalità di un campo di testo; una TextView può essere editabile o meno, nel secondo caso ha caratteristiche più simili ad una label che ad un campo di testo. La classe TextView estende View, e sta nel package android.widget.

UML - (Unified Modeling Language, in italiano "linguaggio di modellizzazione unificato") è un linguaggio di modellizzazione e di specifica basato sul paradigma orientato agli oggetti.

View - Con questo termine si intende un qualunque elemento che appare in un'interfaccia utente e che svolge due funzionalità: mostra un aspetto grafico, gestisce eventi relativi all'interazione con l'utente. Con View si intende anche la corrispondente class android.view.View.

QRCODES

- Uffici

  - Poggioni

  - Gerace, Stramacci

- Carpi, Guerra

- Capria, Pambianco

- aule:

  - A2

  - D2

-B1

-B2

-Altro:

-Ingresso:

-Portineria:

-Errori:

-QRCode non permesso (“adhbbd”):

-Nome non permesso (“Pog43”)

-Non trovato (“Panbianco”, invece che “Pambianco”):