"Alexandru Ioan Cuza" University of Iaşi
Faculty of Computer Science

Bachelor Thesis

Building guidance

Author:

Arvinte Razvan George

Session: July 2018

Advisor and coordinator:

Conf. dr. Lenuta Alboaie

Content

1. Introduction

1.1. Summary

This thesis represents an application that is design to help the user to locate different classes in UAIC campus by using Euristic Image Recognition and Optical Character Recognition (OCR). It also includes a module of Pathfinding that will draw for the user a path from a starting location of chose to a final location within a building.

It also provide geographic support with the help of the Google Maps API and Google Location API. This way, the user is able to navigate through the campus of the UAIC and around the city.

1.2. Motivation

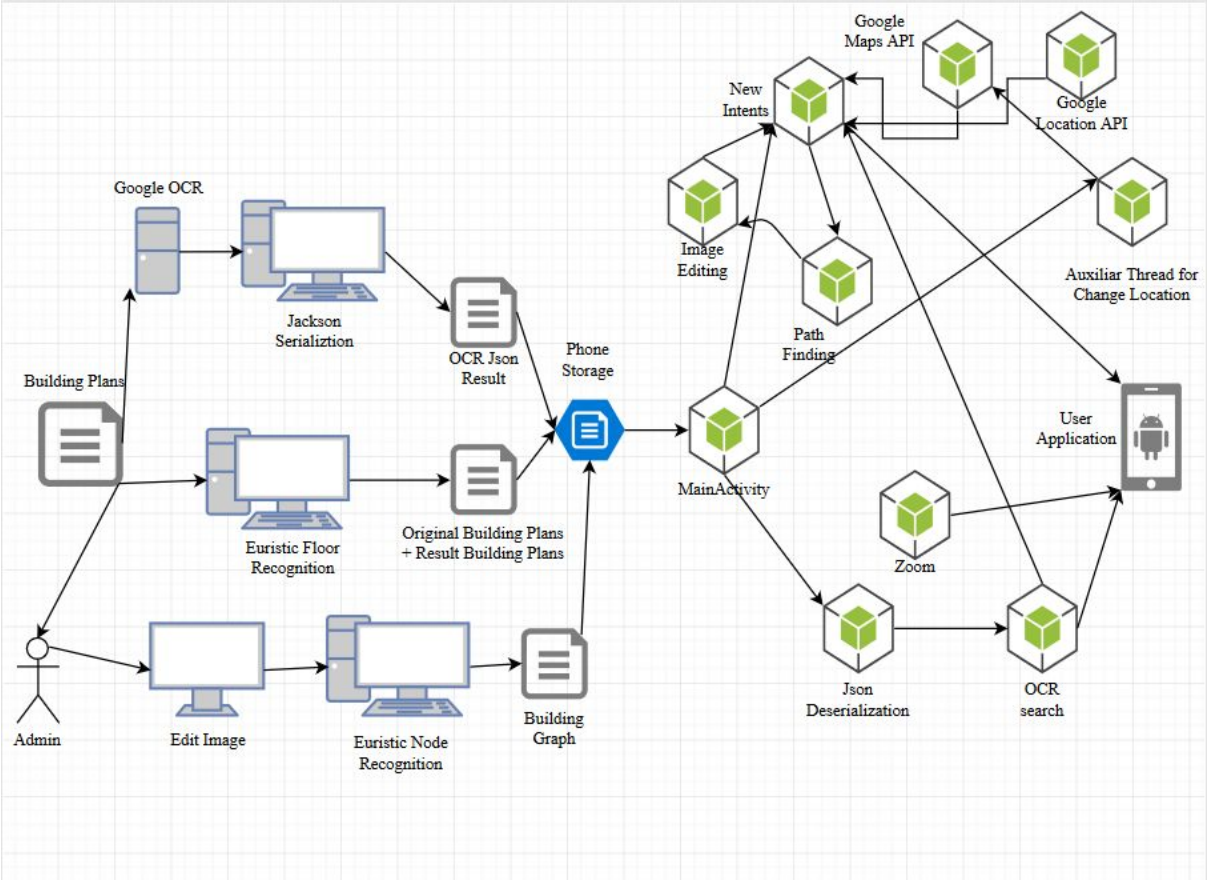
One of the problems that this application wants to solve is the lack of space awareness that fresh students usually have. Being new to the university, most of them don't know the location of the buildings where they have to attend classes, much less the floor or the exact location of the class.

This application is designed to be simple to use and user-friendly. Having a map to play around inspires the students to explore both the university and the city.

Also in case of emergencies the fire fighters can use this application to quickly locate the problem by using the buildings plans provided instead of trying to get them by other means which can take precious time.

2. Application flow

Application flow CPU -> Phone



2.1. Image processing

Parsing Floor Plan Images.

-neuronal networks-

One important article in this direction is an article written by Samuel Dodge, Jiu Xu and Bj¨orn Stenger. This article is meant to make a summary of a big set of algorithms and neuronal networks, proving their efficiency on test datas.

In their thesis they combine three methods to extract geometric and semantic information: Wall segmentation, object detection, and optical character recognition (OCR).

Those three are the key factors into recognizing and classifying the floor image.

Wall segmentation

They use fully convolutional networks (FCN) for segmenting wall pixels. They train models with different pixel-stride value and compare FCN architectures for different final stride layers to find the best stride value. Starting with a FCN-32s with a 32-pixel stride, initialized with (Visual Geometry Group)VGG-16 parameters , in sequential training each model is initialized with the parameters of the previous one. All models are trained by stochastic gradient descent with momentum and a batch size of 1.

Object detection

They use (Convolutional Neuronal Network) R-CNN for object detection and recognition. By doing so they can detect objects like: doors, sliding doors, stoves, bath tubs, toilets and sinks.

Optical Character Recognition

They used Google Vision API for detecting and recognizing text in their images. This proved useful since in this way they can make use of many more languages then just English.

The flow of their algorithms for floor image recognition.

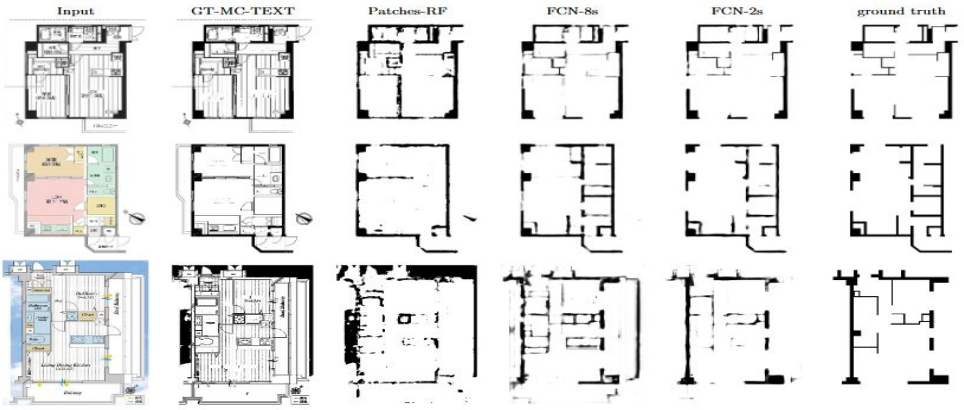


Figure 3: Wall segmentation on images from the new R-FP data set. *Global thresholding (GT-MC-TEXT) does not handle decorative lines and symbols. The patch based segmentation (Patches-RF) cannot handle different drawing styles with the same parameter settings. Segmentations by FCN-8s [14] are more blurred than those by the proposed FCN-2s. The bottom row shows difficult example on which all methods perform poorly.*

The image above is a screenshot of a scheme from their article.

(6.1 Bibliography)

Their conclusion

"We show that an FCN with stride of 2 achieves better performance than a stride of 8. The smaller stride is effective because some walls can be easily described by low level features. The FCN model combines features from subsequent layers that describe contextual information and texture information with these low level features. Secondly, the smaller stride allows us to predict more precise wall locations. Walls are often very thin structures that cannot be precisely predicted at larger strides.

From the floor plan we extract a parsed representation of wall locations, objects, and size information."

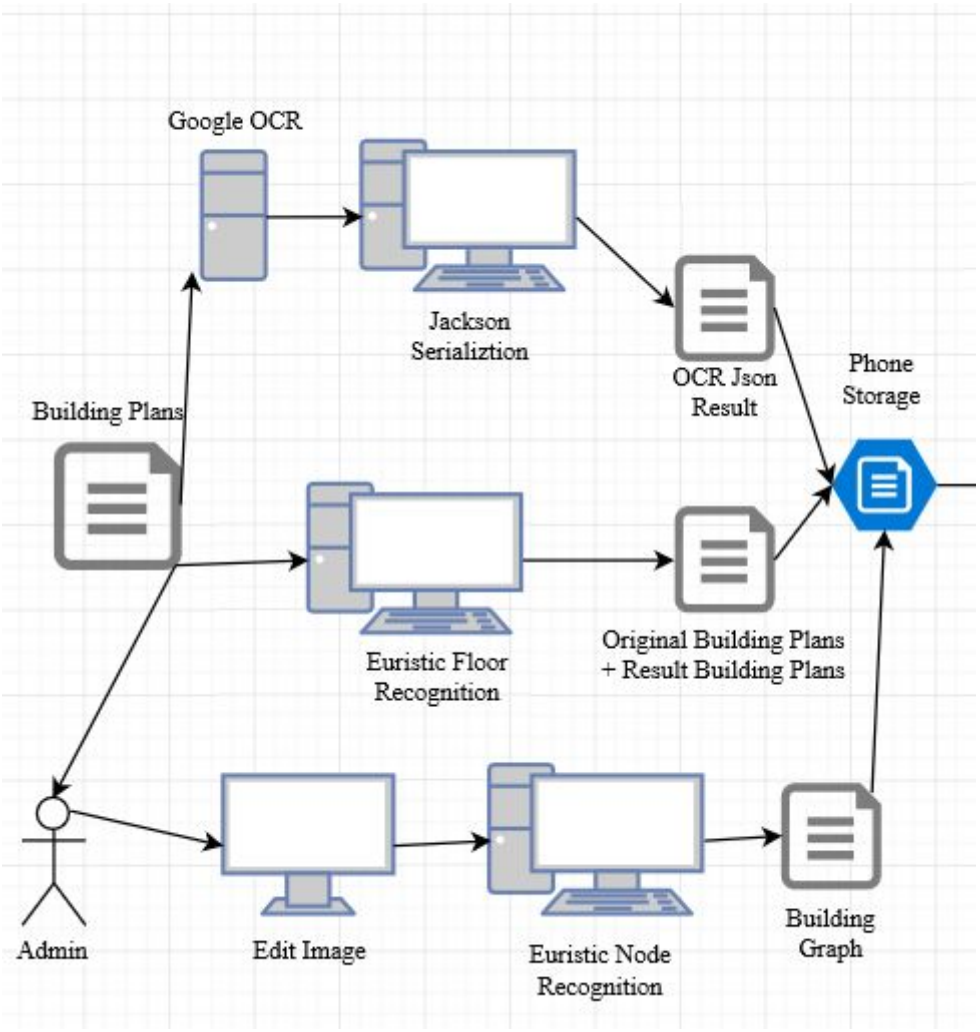
Personal Conclusion

This article is very useful and accurate at detecting floor plans, but the main problem of this solution is that they use and train neuronal networks. In the context of android development, the computational power if a big problem since it would have a big system resource requirement and a problem with the power consuming.

Use case: Accuracy very good but performance lacks time-wise. The memory used is not a problem for a high specifications android device.

This being said, I decided that I should inspire myself from their solution and try to find a faster and simpler one, suited for android development.

Computer process



Parsing Floor Plan Images.

-euristic solution, personal solution-

The solution chosen is an euristic one, based on the generic propriety of the building wall. That propriety is that the wall is represented by of the biggest groups of adjunct points that are black.

To do so, an auxiliar matrix is created with the purpose to memorise all the black group of pixels in the picture. Those groups are starting to get formed while iterating the image pixels. A black pixel is considered black if the it has strong enough RGB colors. Since their colors can variate from white to black, a pixel is considered black if it has over the color

intensity set by the application. Once a black pixel is found, a fill algorithm similar to the Lee's Algorithm is used to find all the points associated with the starting pixel by looking in all the eight direction of the search. The Lee Algorithm takes all the non-visited neighbours and add them to a queue. By iterating the growing queue we can visit all the black pixels of a group.

With the image processed and all the groups formed in the matrix, the biggest group of black pixels is considered the wall of the building plan while the rest of the pixels that are not from that group, no matter what color they had are turned white. This way, the only remaining pixels that aren't white are the one selected previously.

The following code takes the input image and checks for each pixel if it is considered black. If it's not the pixel color turns white, while in the other case it is changed into clear black.

Checks the intensity of the pixel colors.

```
image = ImageIO.read(inPath);
int bin=80;
for(int i=0;i<image.getWidth();i++)
   for(int j=0;j<image.getHeight();j++)
   {
       try {
           int k=image.getRGB(i, j);
           int b = 0xFF & (k); //blue
           int g = 0xFF & (k>>8); //green
           int r = 0xFF & (k>>16); //red
           int a = 0xFF & (k>>24); //alpha
           if((b>bin|r>bin|g>bin)) // intensity check
               k=0xFFFFFF;
           else
               {k=0;}
           image.setRGB(i, j, k);
       }
       catch(Exception error)
       {
           out.println(Integer.toString(i) + " "+    Integer.toString(j));
       }}
```

The following code initializes the matrix that will memorize each pixel and in each group group it is. Also iterates the matrix to check if there are black pixels that are not yet assigned to a group. In the case that the pixel is black and is not yet assigned to a group and new group is formed, composed of it and all his black pixel neighbours that are not yet assigned to a group.

Code that is creating the groups of black pixels:

```
int aux[][]=new int[image.getWidth()][];
for(int i=0;i<image.getWidth();i++)// matrix initialization
{
   aux[i]=new int[image.getHeight()];
   for(int j=0;j<image.getHeight();j++)
       aux[i][j]=0;
}
int number=1;
ArrayList<Long> strength=new ArrayList<Long>(); //group list and their
                                                                    strength
for(int i=0;i<image.getWidth();i++)
{
   for(int j=0;j<image.getHeight();j++)
   {
       int k=image.getRGB(i, j);
       k=0xFFFFFF&k;
       if(k==0 && aux[i][j]==0) // in case the pixel is black and is not
                                                                assigned yet
       {
           strength.add(fill(i,j,number,aux,image));//create a new group
           number++;
       }
   }
}
```

This code that is selecting the largest group of black pixels from the existing groups. Here the strength vector memories the number of pixels of each group. The number variable memorizes the unique identifier of the group.

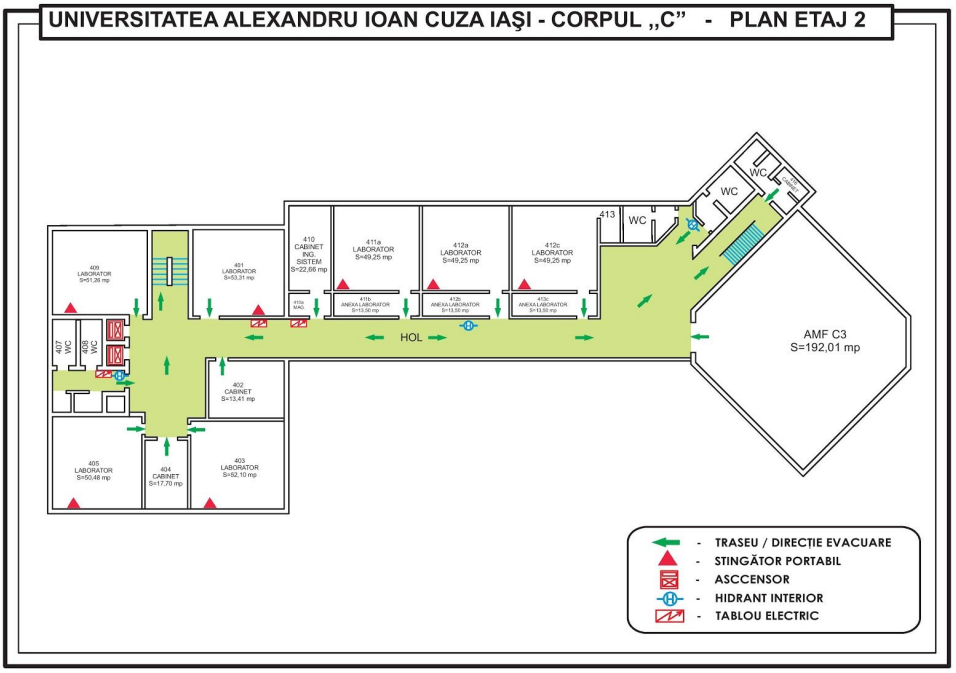
This being said, the pixel that need are not in the “number” groups will be set to white, remaining only the pixels selected.

Code that is keeping only the biggest group of black pixels:

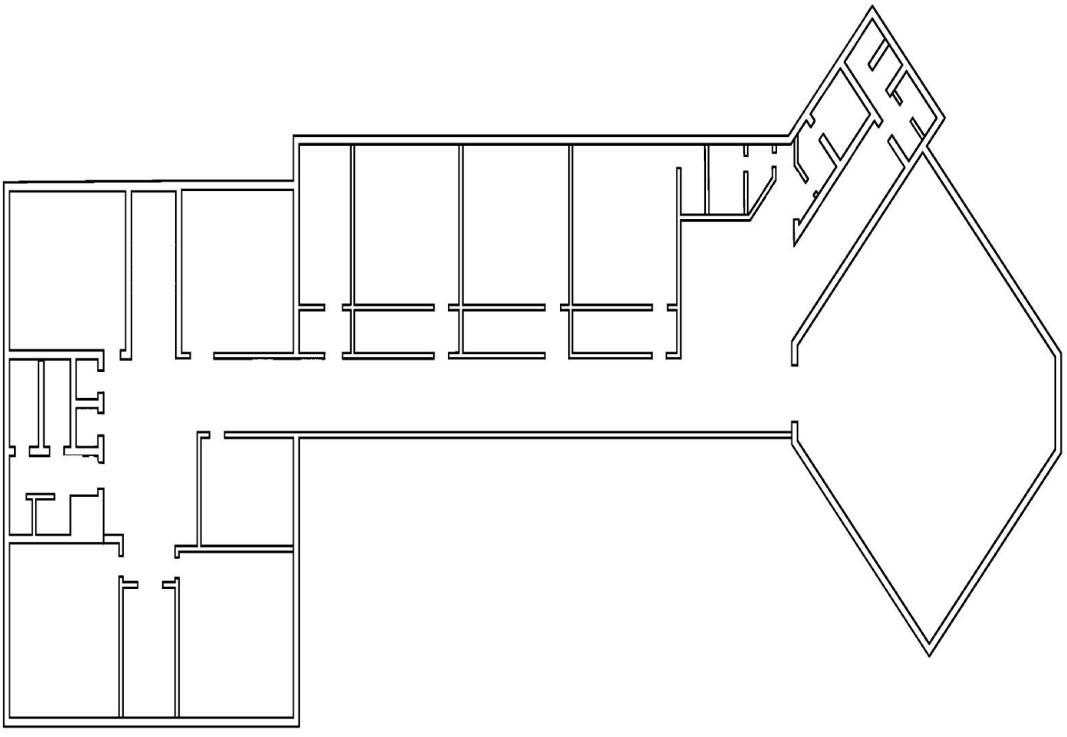
```
number=0;
long val=0;
for(int i=0;i<strength.size();i++)
{
   if(val<strength.get(i))
   {
       val=strength.get(i);
       number=i;
   }
}
number+=1;//array starts from 0 but the numbering started with 1
strength.sort(null);
for(int i=0;i<image.getWidth();i++)
   for(int j=0;j<image.getHeight();j++)
   {
       if((image.getRGB(i, j)&0xFFFFFF)==0&&aux[i][j]!=number)
           image.setRGB(i, j, 0xFFFFFF);

   }
```

And now the result is that this new image represents the walls of the floor plan image.

Initial Image, Building C Second Floor, UAIC



Result Image, Building C, Second Floor, Uaic[1]



Conclusion

Use case: Accuracy can be a problem depending on the consistency or quality of the images. The application is using little memory when running the euristic algorithm because it uses only a matrix of the size of the picture. Also it have linear time in raport with the size of the image because of its one time iteration through the pixels.

With this resulted image [1] above, the walls are extracted successfully and simple operations can be applied to the image like search algorithms.

On the other hand, the image[1] can be re-edited for a more common use later on (since all the symbols can be considered a nuisance in some cases).

OCR (Optical Character Recognition)

For the OCR part was used the Google Cloud Vision API. The received data was stored into a custom class that later will be serialized with Jackson API for Json serialization. The reason the default class serialization was not used that when serializing the class instances, the package data is stored within the class so that later it will require the same package for deserialization. Taking the fact that we move the data from an java application on the computer to an android application on phone, the package serialization/deserialization becomes a problem.

Solution: Serialization in Json format using Jackson, hand-programmed deserialization for the OCR data on android. Also in the filename field is concatenated the floor and the building name of the specific room that was found.

```
floor_data_wrapper wrapper=new floor_data_wrapper(all_floor_data);
ObjectMapper mapper = new ObjectMapper();

mapper.writeValue(new
File("c:\\work\\Licenta_2018_Arvinte_Razvan\\result_data\\Floor_data.json"),
wrapper);
```

Pathfinding-image recognition

Starting problem: taking in account the walls that have already been found in the images, there is still the problem with the stairs. Since the stairs don't have a standard, a decision have to be taken for pathfinding to be possible.

Solution: Ask the admin to mark the stairs with a personal standard in the image so that later we can recognise them in the image. In the current case, it was chosen a purple circle. With this, calculating the center of the circle means finding the exact point from where the user can go up or down a level.

Considering different algorithms for pathfinding, one of the most precise is the Lee's Algorithm. The method used by the algorithm is simply described in pseudocode as follows:

```
Initial_matrix = input_matrix
Queue.add( first_point )
While ( !Queue.isEmpty( ) )
    current_point=Queue.pop( )
    if ( current_point == end_point )
        Print " destination reached"
        break
    Queue.addAll( notVisitedYet( current_point.allNeighbours() ) )
    Initial_matrix.mark( notVisitedYet( current_point.allNeighbours(),
                                        Current_point.value+1 )
Initial_matrix.printPath( end_point, first_point )
```

The problem in our case with this algorithm is that he uses a lot of memory and computational time for big matrix. Some of the images we use contain tens of thousands of pixels.

Solution: transform the problem of pathfinding in a matrix into a pathfinding problem into a tree. With way fewer points to check, the problem becomes simple even for telephones. So the admin will have to mark possible joints on the building plans and then make the connection between them.

Initial reduced picture

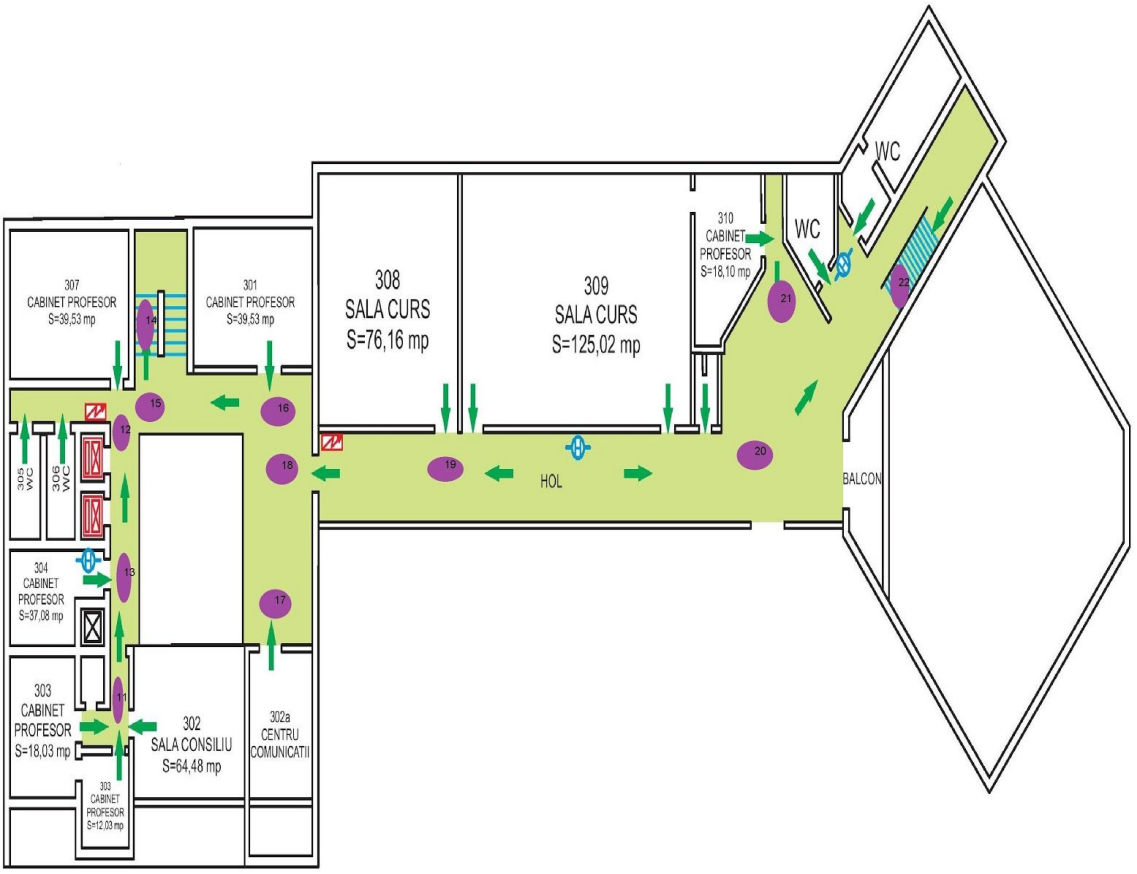
Modified by admin picture with joint points for stair and corners

At this point there is no way for the administrator to understand how can the computer see the circles he drawn, so I implemented an euristic image recognition algorithm for finding

the circles and marking their centers with numbers, so that the admin can now make the connection between the joints.

```
static pair getCenter(int x,int y, int k, int[][] mat,BufferedImage img)
{
   ArrayDeque<pair> coada = new ArrayDeque<pair>();
   List<pair> points= new Vector<pair>();
   coada.push(new pair(x,y));
   pair current;
   int strength=0;
   while(!coada.isEmpty())
   {
       current = coada.pop();
       points.add(current);
       mat[current.x][current.y]=k;
       strength++;
       for(int i=-1;i<=1;i++)
           for(int j=-1;j<=1;j++)
           {

if(current.x+i>=img.getWidth()||current.y+j>=img.getHeight()||current.x+i<0||cur
rent.y+j<0)
                   continue;
               if((img.getRGB(current.x+i, current.y+j)&0xFFFFFF)!=0xA349A3 ||
mat[current.x+i][current.y+j]!=0)
                   continue;
               coada.push(new pair(current.x+i,current.y+j));
           }
   }
   pair center=new pair(0,0);
   for (int i=0; i<points.size();i++)
   {
       current=points.get(i);
       center.x+=current.x;
       center.y+=current.y;
   }
   center.x=center.x/points.size();
   center.y=center.y/points.size();

   if (points.size()<1000){
       return new pair(0,0);
   }
   return center;
}
```

The result of the joint-recognition algorithm

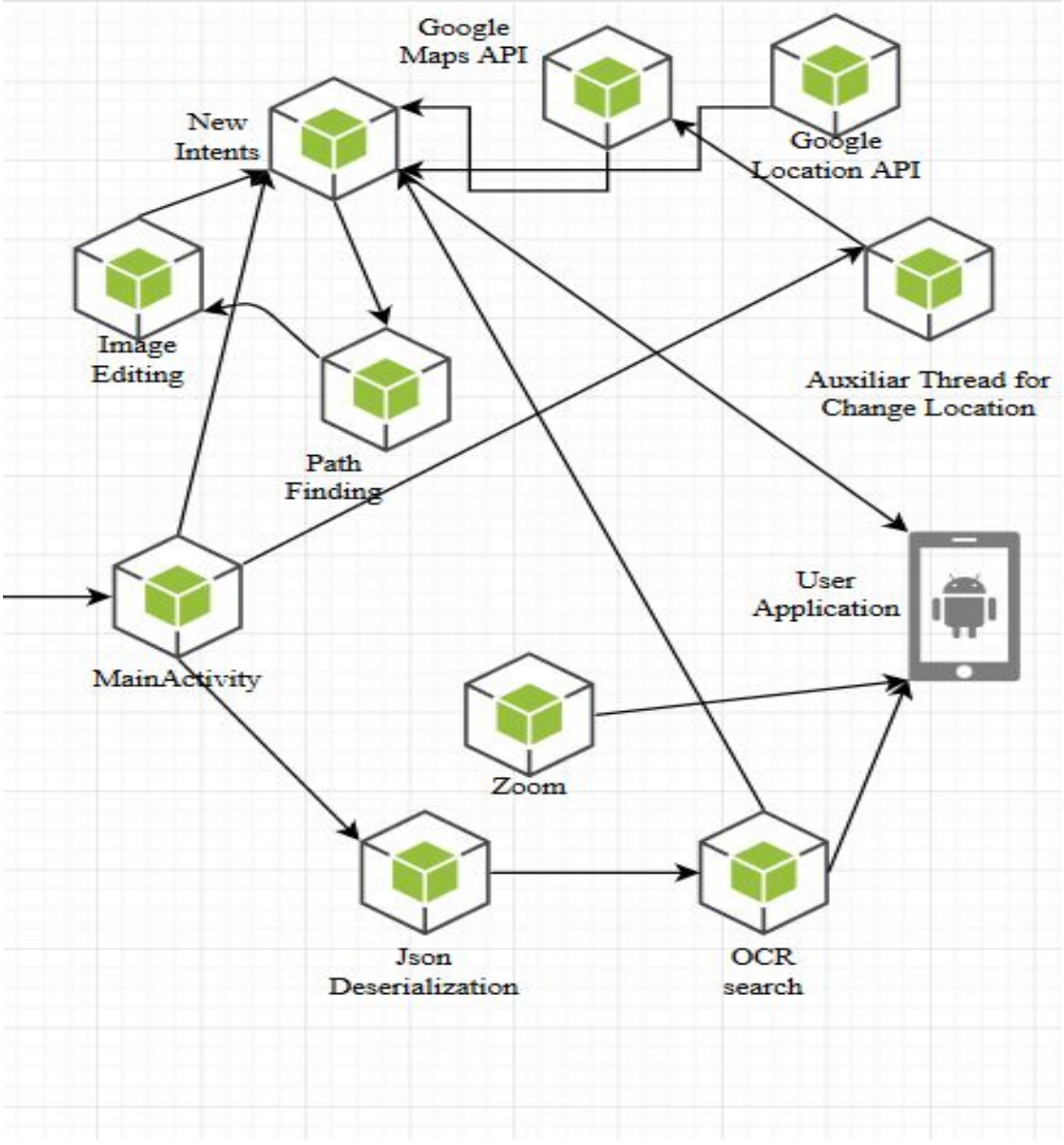


At this point the admin will fill up a list of edges between the points presented. He will also consider that there should be a connection between floors through the “stair” joints.

With this done, the next phase is serialize into a Json and send the data to the android device.

2.2. Android follow-up

On the Android side of the application flow, the phone will get the resources generated by the computer and use them to provide content to the user's interactions in searching for a class, listing the building plans and listing possible suggestions in case of multiple choices.



Json Deserialization

With the data generated from the computer , the android application has to deserialize the Json and to fill it's own data structures that will be alter used in most of the application's build interaction.

The classes that have to be deserialized.

```
static class pair implements java.io.Serializable
{
   public int x,y;
   pair(int a,int b)
   {
       x=a;
       y=b;
   }
};

static class floor_data implements java.io.Serializable{
   List<pair> centers;
   List<String> words;
   String fileName;
   public floor_data(List<pair> centers, List<String> words, String
fileName) {
       this.centers = centers;
       this.words = words;
       this.fileName = fileName;

   }
}
static List<floor_data> all_floor_data=new Vector<floor_data>();
```

The deserialization function that is used for the data that came from the computer:

```
public void Deserializare(){
   JSONParser parser= new JSONParser();
   try {
       JSONObject myjson = (JSONObject) parser.parse(new
FileReader("/storage/emulated/0/poze-root/result_data/Floor_data.json"));
       JSONArray all_floors = (JSONArray) myjson.get("floors");
       Iterator<JSONObject> itFloor = all_floors.iterator();
       //parse all the floors
       while(itFloor.hasNext()){
           JSONObject current_floor= itFloor.next();
           List<pair> Pairs=new Vector<pair>();
           //parse the centers
           JSONArray all_centers= (JSONArray)    current_floor.get("centers");
           Iterator<JSONObject> center= all_centers.iterator();
           while(center.hasNext()){
               JSONObject current_pair=center.next();
               int x=((Long)current_pair.get("x")).intValue();
               int y=((Long)current_pair.get("y")).intValue();
               pair temp_pair = new pair(x,y);
               Pairs.add(temp_pair);
           }
           //parse all words
           JSONArray all_words = (JSONArray) current_floor.get("words");
           Iterator<String> word= all_words.iterator();
           List<String> Words=new Vector<String>();
           while(word.hasNext()){
               Words.add(word.next());
           }

           String filename=(String) current_floor.get("fileName");
           System.out.println(filename);
           floor_data temp_floor=new floor_data(Pairs,Words,filename);
           all_floor_data.add(temp_floor);
       }
   }
   catch (Exception e){
       e.printStackTrace();
   }
}
```

The OCR search is composed of an adapter on a SearchView, the SearchView and the ListView for the items. This adapter filters the information and offers the user for each search result the specific building and floor of it.

The filter function for the searchView checks if the search is empty. In that case the view doesn't change. But if the search is not empty the function will try to inflate the ListView asignated to the searchView results.

The filter function for the SearchView

```
public void filter(String charText) {
   Log.i("Eu_search","filtering_text: " + charText );
   if (charText.equals(""))
   {
       arrayRooms.clear();
       arrayLocations.clear();
       notifyDataSetChanged();
       return;
   }
   charText = charText.toLowerCase(Locale.getDefault());
   arrayRooms.clear();
   arrayLocations.clear();
   Log.i("Eu_search",room_list.toString());
   int count;
   for (count=0;count<room_list.size();count++) {

       if
(room_list.get(count).toLowerCase(Locale.getDefault()).contains(charText)) {
           arrayRooms.add(room_list.get(count));
           arrayLocations.add(filename_list.get(count).split("\\.")[0]);
           count++;
           //Log.i("Eu_search","Add in list: " + wp);
       }
   }
   Log.i("Eu_search","Result"+room_list.toString());

   notifyDataSetChanged();}
```

The function that adds element to the View that will be inflated with search results.

```
public View getView(final int position, View view, ViewGroup parent) {

   final ViewHolder holder;
   if (view == null) {
       holder = new ViewHolder();
       view = inflater.inflate(R.layout.list_view_items, null);
       // Locate the TextViews in listview_item.xml
       holder.name = (TextView) view.findViewById(R.id.name);
```

```
        holder.location = (TextView) view.findViewById(R.id.location);
        view.setTag(holder);
    } else {
        holder = (ViewHolder) view.getTag();
    }
    // Set the results into TextViews
    holder.name.setText(arrayRooms.get(position));
    holder.location.setText(arrayLocations.get(position));

    return view;
}
```

Google Maps API

With the use of google maps, the application has a frangement that represents the map of the user's city and some custom polygons that are link to the plan of the buildings. Those polygons are drawn by the admin.

Initialization of the map

```
private void initMap(){
    if (CheckGps()) {
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }
}
```

This code draws the polygons that the user will work with. The user will use those polygons to open the plans of each building.

```
public void onMapReady(GoogleMap googleMap) {
    polygons=new Vector<Polygon>();
    mMap = googleMap;
    final Polygon polygon_info = mMap.addPolygon(new PolygonOptions()
            .clickable(true)
            .add(
                    new LatLng(47.173919, 27.574654),
                    new LatLng(47.173968, 27.574802),
                    new LatLng(47.173876, 27.574885),
                    new LatLng(47.173821, 27.574733)));
    polygon_info.setFillColor(ContextCompat.getColor(this, R.color.argb_black));
```

```
   polygon_info.setStrokeColor(ContextCompat.getColor(this,
R.color.argb_black));
   polygons.add(polygon_info);

   mMap.setOnPolygonClickListener(new GoogleMap.OnPolygonClickListener() {
       @Override
       public void onPolygonClick(Polygon polygon) {
           if (polygon.equals(polygon_corp_D)){
               Intent i = new Intent(getBaseContext(), Inside.class);
               i.putExtra("corp","Corp_D");
               startActivity(i);
           }
           if (polygon.equals(polygon_info)){
               Intent i = new Intent(getBaseContext(), Inside.class);
               i.putExtra("corp","Corp_C");
               startActivity(i);
           }
           if (polygon.equals(polygon_feaa)){
               Intent i = new Intent(getBaseContext(), Inside.class);
               i.putExtra("corp","Corp_B");
               startActivity(i);
           }
           if (polygon.equals(polygon_corp_A)){
               Intent i = new Intent(getBaseContext(), Inside.class);
               i.putExtra("corp","Corp_A");
               startActivity(i);
           }
       }
   });
```

Google Location API

With Google Location API, the application inserts markers and draws the shortest path to the destination from the current location of the user. Also while the application runs there is an auxiliar thread that keeps track of the user's position and in case of a change in coordinates, the line will redrawn.

The code that will draw the first path on the map. On the moment the first path leads to Building C from UAIC.

```
if (mLocationPermissionsGranted) {
   getDeviceLocation(new LatLng(47.173951, 27.574782));// will draw the first
path, default Building C

   if (ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)
           != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this,
           android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
       return;
   }
   mMap.setMyLocationEnabled(true);
   mMap.getUiSettings().setMyLocationButtonEnabled(true);
}
```

This is the function that is drawing a line from the current location to the selected destination.

```
private Polyline addPolyline(DirectionsResult results, GoogleMap mMap) {
   List<LatLng> decodedPath =
PolyUtil.decode(results.routes[0].overviewPolyline.getEncodedPath());
   return mMap.addPolyline(new PolylineOptions().addAll( decodedPath ));
}
```

The function that is drawing the markers cam makes the user change his destination with OnMarkerClick. This way the user can choose the destination from the buildings marked on the map by the administrator.

```
private void addMarkersToMap(DirectionsResult results, GoogleMap mMap) {
   Log.i("EU_Marked", String.valueOf(results));

   mMap.addMarker(new MarkerOptions().position(new
LatLng(47.174829,27.571814)).title("Corp_A"));//Building_A
   mMap.addMarker(new MarkerOptions().position(new
LatLng(47.175061,27.573703)).title("Corp_B"));//Building_B
   mMap.addMarker(new MarkerOptions().position(new
LatLng(47.173716,27.575032)).title("Corp_C"));//Building_C
   mMap.addMarker(new MarkerOptions().position(new
LatLng(47.172847,27.571887)).title("Corp_D"));//Building_D

   mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
       @Override
       public boolean onMarkerClick(Marker marker) {
           LatLng poz=marker.getPosition();
           myLastPolyline.remove();
           getDeviceLocation(poz);
           moveCamera(poz, 15f);
           return false;
       }
   });
}
```

When trying to get the location from GPS, the application uses Google Location API which is getting the current location of the user. Also the application is using its current location to send a request to Google asking what is the shortest path between the current location and the destination that is currently selected on the map.

```
final Task myTask = mFusedLocationProviderClient.getLastLocation();
Task task = myTask.addOnCompleteListener(new OnCompleteListener() {
   @Override
   public void onComplete(@NonNull Task task) {
       if (task.isSuccessful()) {
           if (myTask.isSuccessful()) {
               Location MyCurrentLocation = (Location) myTask.getResult();
```

```
                DateTime now = new DateTime();
                Double diff = abs(myCurrentOrigin[0].lat -
MyCurrentLocation.getLatitude()) + abs(myCurrentOrigin[0].lng -
MyCurrentLocation.getLongitude());
                if (diff > 0.0001) {
                    myLine.remove();

                    myCurrentOrigin[0] = new com.google.maps.model.LatLng(
MyCurrentLocation.getLatitude(), MyCurrentLocation.getLongitude() );
                    DirectionsApiRequest Path =
DirectionsApi.newRequest(getGeoContext());
                    Path.mode(TravelMode.DRIVING);
                    Path.origin(myOrigin);
                    Path.destination(myDestination);
                    Path.departureTime(now);
                    DirectionsResult result = null;
                    try {
                        result = Path.await();
                    } catch (ApiException | InterruptedException | IOException e)
{
                        e.printStackTrace();
                    }
                    myLine = addPolyline(result, myGoogleMap);
                }
            }
        }
    }
});
```

Zoom

There is a class implemented that in case of any image displayed in the building plans are to be scaled so that the user can better view the image.

Pathfinding

The pathfinding on the android side has as input the graph parsed from the computer. This being said, the problems is finding the fastest route from one point to another in a graph.

The only problem is determining the start and finish points. This is solved by calculating for example the euclidean distance between the start_text and all the joints from that level, taking the closest node.

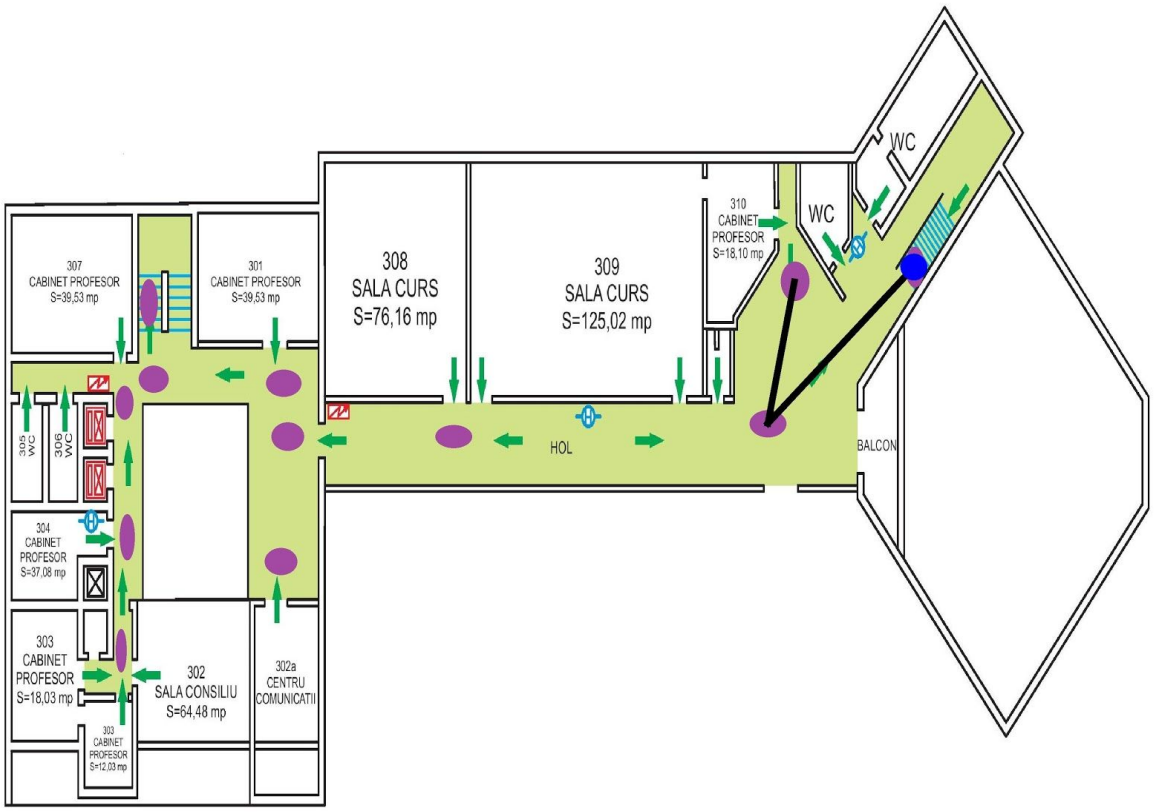
The algorithm used is a BFS(Breadth-first search). This algorithm uses a queue to go through all the graph. It is similar with the Lee's Algorithm described above, one of the

differences being that here the algorithm works with more complex data structures that has neighbours in lists and not in north,south,east,west.

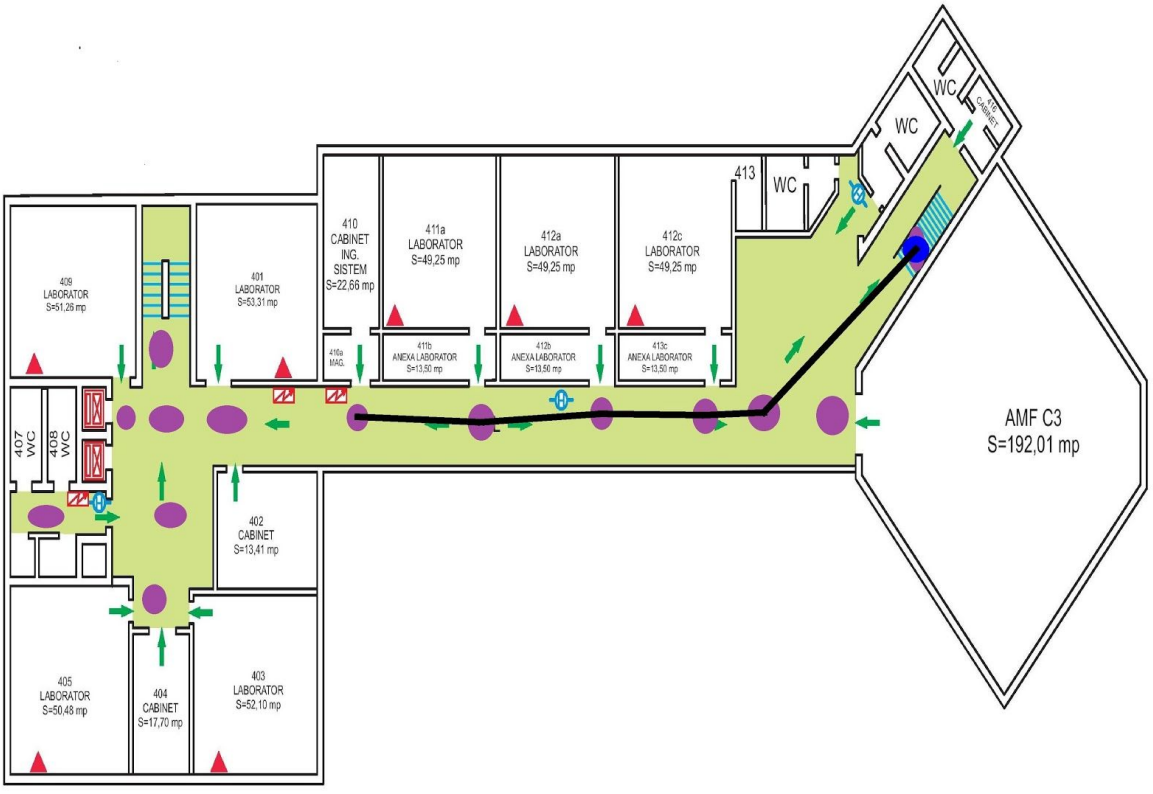
Having found the path, a function will start drawing the road to destination on copies of the current images of the building plans.

The following example starts has a path from room 310 first floor up to 410 second floor, from Faculty of Informatics UAIC.

First floor, Faculty of Informatics, UAIC



Second floor, Faculty of Informatics, UAIC



Those images are created and stored on the android device, the process of creating/editing images being the most costly action that the application will do resource-wise.

3. Use cases

3.1. Student

In these days there is a lot of information on the internet and so many of the fresh student have problems with finding some of the basic information that they need in their studying at UAIC. One of the most important basic information is the location of the classes they need to attend. This application is meant to be simple to use and easy to understand. The application not only that it have a quick search which can guide the student to the place he wants to reach, but it also makes it easy for him to explore the campus from the app itself.

3.2. Firefighters

In case of emergency, there is almost no time to get information about the structure of the building in trouble. Getting the building plans from the administrators, having problems with busy telephone lines and having the victims unaware of their current position when a problem rises can provoke casualties that normally wouldn't be hurt. This application provides quick support with basic important information for both the possible victims of an incident and the possible forces of rescue.

The application simplicity proves very useful in such situation that can happen rarely.

3.3. General Admin

This application is not meant only for the UAIC campus. With basic configuration any administrator of a building that can provide it’s plans should be able to easily create a simple and useful application no matter how large or high the building is. The scalability of the application makes it usable for almost any type of building, fact that can help to a better safety in the current times.

4. Personal Contributions

4.1. Short Introduction

This thesis can be developed very nicely with some ideas that i had on the way of solving the current needs that the article wants to solve. Unfortunately there were some problems with the development of these ideas that I researched, because of their complexity or the lack of accuracy.

4.2. Application contributions

4.2.1. The application's architecture

From the start I tried to be as efficient in time and memory when it comes to algorithms. I split the workload between computer and phone because usually the computer has better computational power then the phone. Also all the code was written in java so that it can work on both the android platform and the windows platform.

4.2.2. My own serialization/deserialization

Although it is not as big of a feat as others, the class's serialization / deserialization for cross-package/cross-platform in java has its own errors, most of them because of different package problem. My own deserialization was my "simple" solution to the problem.

Also it was interesting for me to have wrapper classes so i could properly serialize a vector of instances of other classes that also had a vector of instances of another class.

4.2.3. My euristic Floor Image Processing

I realised that the known methods that were researched in Rakuten Institute of Technology from Arizona State University were very quite difficult to implement and also slow in both memory and time, so i created my own solution to the problem; a solution that is fast time-wise and uses very little memory, 2 * size(image).

4.2.4. The pathfinding

The architecture of the image pathfinding application flow is as follows: building plans, admin node editing, image node recognition, admin adding the edges, building the graph, serializing the graph (*) , on android deserialize, find the start and finish node, find the path with a BFS algorithm, edit image and finally create a new activity .

Until (*) all the modules are implemented for the computer in java, after (*) all the modules are implemented for the device in android.

4.3. Added research on the application

4.3.1. Offline Tracking

Once you enter a building most likely the GPS signal will drop and the application won't be able to guide you to the selected destination. For this problem the solution was Offline Tracking, where the application would use the phone sensors to calculate the distance that you went and in what direction or triangulate the wifi to find your location given the fact that the application already knows the positions of the wifi routers in the building.

One of the first problem was the accuracy of the GPS, because the moment you lose the signal you can still be quite off from your real position depending on your phone performances.

A second problem was similar to the one with the GPS, sometimes the sensors of the phone are not accurate, so even if the starting position is a real one, it can drove off the track quite fast.

Conclusion: The offline tracking was not implemented.

4.3.2. 3D Building

With my euristic Floor Image Recognition I successfully made some simple plans for the buildings. With those new images there could be build a 3D model of the structures.

But without a good accuracy, the 3D model would only bother the user and would also use phone resources for no reason.

Conclusion: The 3D Model of the Building was not implemented.

5. Conclusions

5.1. This application solves some of the basic yet very important problems that fresh students have; that of locating where the next classes are going to be taken and the exploration of the UAIC campus without much effort.

Also with those functionalities it also provides good support for the possible forces of rescue in case of emergency.

5.2. With this application I look forward to help the society and have people informed of what they really want to know by giving them simple solutions.

5.3. The application is extensible and easy to adapt to the context of the necessity. This being said it can be further developed on necessity and can slowly become the application that every student will want to have as support for his years of study in UAIC.

6. Bibliography

6.1. Parsing Floor Plan Images by Samuel Dodge, Jiu Xu and Bj¨orn Stenger
http://bjornstenger.github.io/papers/dodge_mva2017.pdf

6.2. Offline Tracking- Triangulation of a phone
http://myphonelocater.com/category/triangulation/

6.3. Offline Tracking- Tracking a phone with sensors
https://gizmodo.com/how-to-track-a-cellphone-without-gps-or-consent-1821125371

6.4. Google maps API, SDK
https://developers.google.com/maps/documentation/android-sdk/intro

6.5. Google location API
https://developer.android.com/guide/topics/location/

6.6. 3D Model Building, online 3D builder
http://the2d3dfloorplancompany.com/product/2d-to-3d-floor-plan-conversion/

6.7. UAIC main page
http://www.uaic.ro/

6.8. Lee's Algorithm
http://www.techiedelight.com/lee-algorithm-shortest-path-in-a-maze/

6.9. BSF Algorithm
https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/