

2012

Architect Support Tool Cover

Alexandre Uroz Làzaro

Title: Development of a software tool to support architects

Volume: 1/1

Student: Alexandre Uroz Làzaro

Tutor: Dr. Krzysztof Waśko

Department: Systemy multimedialne i mobilne

Date: 24 of September 2012

This thesis is divided in three documents:

1 ARCHITECT SUPPORT TOOL FULL DESCRIPTION

Full description of the thesis intended for Informatics specialists.

2 ARQUITECT SUPPORT TOOL RESUME

Resume description of the thesis intended for Informatics specialists.

3 ARQUITECT SUPPORT TOOL MANUAL

Manual of the application intended for architects and end users.

Each one of them are independent documents, but they are enclosed together for delivery purposes.

2012

Architect Support Tool

Alexandre Uroz Làzaro

Title: Development of a software tool to support architects

Volume: 1/1

Student: Alexandre Uroz Làzaro

Tutor: Dr. Krzysztof Waśko

Department: Systemy multimedialne i mobilne

Date: 24 of September 2012

**Development of a software
tool to support architects**
[AST] by Alexandre Uroz

Dedicated to all my friends and specially my family without them I would had never finished
Àlex

*Moltes gràcies a tots els amics i companys que m'he anat trobant durant la carrera i
especialment als amics de l'escola, i a la magnifica gent de Wrocław*

1	INTRODUCTION	- 5 -
1.1	PROJECT JUSTIFICATION AND DEVELOPMENT'S ENVIRONMENT.....	- 5 -
1.2	PROJECT GOALS.....	- 6 -
1.3	SCOPE AND USED METHODOLOGY	- 7 -
1.3.1	<i>Project planning</i>	- 7 -
1.4	ENVIRONMENT DESCRIPTION.....	- 9 -
1.4.1	<i>Eclipse</i>	- 9 -
1.4.2	<i>Sony Ericsson Xperia X10 Mini</i>	- 9 -
1.4.3	<i>Android 2.1 framework</i>	- 10 -
1.5	DESCRIPTION OF THE ACTUAL SITUATION	- 11 -
1.6	DISADVANTAGES. ROOM FOR IMPROVEMENT.....	- 11 -
2	DESCRIPTION OF THE ACTUAL REQUIREMENTS.....	- 12 -
2.1	REQUIREMENT ANALYSIS.....	- 12 -
2.1.1	<i>Functional requirements</i>	- 12 -
2.1.1.1	Ability to use this application wherever as possible.....	- 13 -
2.1.1.2	Ability to represent maps.....	- 13 -
2.1.1.3	Ability to include and take photos.....	- 13 -
2.1.1.4	Ability to save and load the data	- 13 -
2.1.1.5	Ability to write descriptions	- 14 -
2.1.1.6	Ability to check a list of elements of occupancy	- 14 -
2.1.2	<i>Non functional requirements</i>	- 14 -
2.1.2.1	User interface and Human factor	- 14 -
2.1.2.2	Efficiency.....	- 14 -
2.1.2.3	Extensibility	- 14 -
2.1.2.4	Maintenance.....	- 15 -
2.1.2.5	Others	- 15 -
2.1.2.6	Environment development and platform.....	- 15 -
2.1.3	<i>Typical course of events from use cases</i>	- 16 -
2.1.3.1	Draw Core	- 16 -
2.1.3.2	Pathology Core	- 18 -
2.1.3.3	Checklist Core.....	- 21 -
2.1.3.4	Persistent Core	- 22 -
2.2	PROBLEM SPECIFICATION.....	- 24 -
2.2.1	<i>Conceptual data model diagram</i>	- 25 -
2.2.1.1	Presentation layer	- 25 -
2.2.1.2	Domain layer	- 26 -
2.2.1.2.1	Draw Core	- 26 -
2.2.1.2.2	Pathology Core.....	- 27 -
2.2.1.2.3	Checklist Core	- 27 -
2.2.1.3	Persistence layer.....	- 28 -
2.3	OTHER SOLUTIONS DESCRIPTION (COMMERCIAL ONES).....	- 28 -
3	DESIGN AND IMPLEMENTATION.....	- 29 -
3.1	PROPOSED DESIGN.....	- 29 -
3.2	ARCHITECT SUPPORT TOOL.....	- 29 -
3.2.1	<i>Main and Menu screen</i>	- 29 -
3.2.2	<i>Drawing screen</i>	- 30 -
3.2.3	<i>Entity screen</i>	- 32 -
3.2.4	<i>Gallery screen</i>	- 34 -
3.2.5	<i>Scrolling</i>	- 38 -
3.2.6	<i>Saving</i>	- 39 -
3.2.7	<i>Loading</i>	- 40 -
3.2.8	<i>Create a new map</i>	- 43 -
3.2.9	<i>Erase a wall</i>	- 44 -
3.2.10	<i>Connecting Points Core</i>	- 45 -
3.2.11	<i>Vector mode</i>	- 46 -
3.2.12	<i>Practical example</i>	- 47 -
3.2.13	<i>Door Input</i>	- 54 -
3.2.14	<i>Checklist</i>	- 55 -

3.2.15	<i>Clear data</i>	- 57 -
3.3	DESIGN OF THE DATA BASE	- 58 -
3.3.1	<i>Save drawing scenario</i>	- 58 -
3.3.2	<i>Save pathology scenario</i>	- 59 -
3.3.3	<i>Save temporary scenario</i>	- 60 -
3.3.4	<i>Save checklist scenario</i>	- 61 -
3.3.5	<i>Creation and purpose of each table</i>	- 62 -
4	TESTING AND PERFORMANCE	- 63 -
5	PLANNING	- 65 -
6	BUDGET	- 66 -
7	POSSIBLE IMPROVEMENTS AND UPGRADES	- 67 -
8	CONCLUSIONS	- 68 -
9	ANNEX A: IMAGES INDEX	- 69 -
10	ANNEX B: CODE INDEX	- 70 -
11	BIBLIOGRAPHY AND REFERENCES	- 71 -

1 Introduction

1.1 *Project justification and development's environment*

The idea of doing this project began back at 2010, in Spain, I have a friend who is an architect and in a casually chat we were having he commented me of how, at the moment, there was no significant application to help architect's like him when they were working in the streets, analyzing, reviewing and studying the buildings.

In this particular case, his job consisted of visiting flats and buildings in order to find pathologies, circumstances that appear on buildings that diminish their security, value and comfort. They need all this information for several purposes, sometimes because the client requested a check of the security of the building and if it is needed to do some fixings, other times they need it in order to issue the certificate of occupancy, other times it might be just simple as to document the state of a building during time.

Usually all this information is gathered by hand making notes and taking photos so as to late introduce them in computer and use their main popular architect program like AutoCad. He told me that would be really useful to have some sort of application that could be launched though a mobile device, like an smartphone or a tablet pc and could assist them in this work.

I thought it was an interesting concept to further develop so when finally I got my change to develop it I proposed it here in Wrocław. And I am happy that the tutor find it interesting too.

It is no secret that an application like this, though small, merges several areas of informatics, like graphical design, graphical interface, user interaction and databases. And finally it is based from a real necessity which encourages to do its development

As up today there is no other free application like this (nor a paid one that I have knowledge) I finally decided to try to do it by myself and this is the result.

After some deliberation, I intended for this application to be for Android systems, the reason is quite simple, first I really like the Open Source environment that surrounds Android Operative System, and second, I had an Android smartphone, and could not afford to buy another smartphone, being said that I am not closing the possibility to develop it for other devices and OS (Operative System's) in the future if it is needed. But for the moment and for starters like me in this Android world, this would be good enough.

1.2 *Project goals*

Good, we have a necessity to fulfil and motivation to develop but now the first problem I encountered was that I did not have previous knowledge of Android, so taking into consideration that here are the goals of this project.

- 1) Learn Android language and be proficient developing for it
- 2) Develop a useful tool for architects that allow them to at least represent maps and introduce information.
- 3) Start a prototype which may be expanded in the future

In order to fulfil this the minimum application's requirements in order to be useful should be as follows:

1. Ability to represent maps
2. Ability to annex photos into sections of these maps
3. Ability to save and load these maps
4. Ability to write descriptions of the pathologies encountered
5. Ability to check a list of requirements of these maps
6. Ability to use this application on smartphone or tablet pc through touch screen

These have been the goals that I manage to accomplish to a certain degree, but this project has so much potential that it can be improved much further. See the section of 7. *Possible improvements and upgrades* for further reference.

1.3 Scope and used methodology

As to develop this project it has been required to do a planning of the several tasks to be performed into independent entities (Gantt diagram). This project is based on **the learning and investigation** required to acquire the appropriate knowledge , alongside with the competences learned through the career, have to made it viable. Once the sufficient knowledge is learned and a necessity study has been made (functional and non functional prerequisites) is when shall begin the **specification and design** phase of the system. This system will have several components that will interact between them (like the camera of the smartphone or the SQLite Database). Finally we will begin the **implementation** phase which will end into the **testing** phase. During the development through all these phases **the documentation of the project** will be performed.

So we can join all these tasks of work into each one of the phases that we already commented resulting into the following 5 main phases:

1. Investigation and Learning
2. Specification and Design
3. Implementation
4. Testing
5. Documentation of the project

The project follows a chronologic time-lapse trying to adapt itself into the classic design progress but with backward jumps to input new information (prototype methodology) to any of he previous four phases. (The documentation phase is being developed through these four phases).

1.3.1 Project planning

Following we show the estimated planning of the project with the 5 phases early stated clearly differentiated and divided into individual and indivisible work tasks.

In red we highlight the starting and finishing date of the projected project and also the dates of completion of the goals of the Specification and Design.

TASK NAME	START	END
1) Investigation and Documentation	1/03/12	30/04/12
Determining architect requirements	1/03/12	12/03/12
Investigating architect terminology	13/03/12	14/03/12
Preliminary analysis	15/03/12	19/03/12
Learning Android environment	20/03/12	22/04/12
Preparing framework	23/04/12	30/04/12
2) Specification and Design	1/05/12	31/05/12
Analysis requirement	1/05/12	7/05/12
Designing components	8/05/12	15/05/12
Draw module strategy	16/05/12	19/05/12
Interface module strategy	20/05/12	22/05/12
Task planning	23/05/12	30/05/12
Approved design	31/05/12	31/05/12
3) Implementation	1/06/12	5/08/12
Main screens and basic navigation	1/06/12	3/06/12
Draw Core	4/06/12	14/06/12
Connect points Core	11/06/12	14/06/12
Menu Option Interface	15/06/12	16/06/12
Menu Option Select	17/06/12	30/06/12
Popup Screen	17/06/12	19/06/12
Length module	20/06/12	26/06/12
Angle module	27/06/12	30/06/12

TASK NAME	START	END
Menu Option Erase	1/07/12	2/07/12
Menu Option Scroll	3/07/12	6/07/12
Menu Option Save	7/07/12	10/07/12
Menu Option Load	11/07/12	15/07/12
Menu Option New	16/07/12	17/07/12
Improvement Connect points Core	18/07/12	19/07/12
Camera Interaction	20/07/12	22/07/12
Gallery Module	23/07/12	28/07/12
Menu Option Door	29/07/12	30/07/12
Pathology Module	31/07/12	2/08/12
Checklist Module	3/08/12	4/08/12
End of Implementation	1/06/12	5/08/12
4) Testing	1/09/12	16/09/12
Fixing code	1/09/12	16/09/12
Preliminary documentation	1/03/12	24/09/12
5) Final Thesis Memory	24/03/12	28/09/12

1.4 Environment description

The environment in which this project has been developed is as follows:

- Eclipse
- Sony Ericsson Xperia X10 Mini
- Android 2.1 Framework

In the following sections we are going to comment why we used each of these technologies and which relation they have with the project.

1.4.1 Eclipse

There are several software development environment, and there is no particular reason to choose Eclipse over another one, just that it is perhaps one of the most used for programming Android, so that's why we choose it too, in order to follow the general trend. So if sometimes one have to look for information on Internet, it will be more easy to find as being more popular. The time in this project it has been critical factor so there were not time to lose trying to figure out which version would be best for developing this application.

1.4.2 Sony Ericsson Xperia X10 Mini

This smartphone is the one that was mainly used to develop this application, the reason is no other that was the one that I hold as my own, so in order to develop it I tested it on it. In order to use other smartphone I was using the emulator coming with the Android framework.

Though I have to comment there is some advantages to develop in this particular smartphone as it is one of the smallest that exist on the market, so, managing to develop a drawing application in such an small screen reinforces the confidence that the application will be working much better on higher screen size devices and it will perform as intended on smaller devices.

1.4.3 Android 2.1 framework

The decision to develop this application for Android is because:

First, the main goal is to develop it for smartphones or tablet pc's, we finally chose smartphones because there are more devices than tablet pc's, so it may be more useful for more people.

Second, as we did not have in our reach other devices with different OS we decided to do it for Android.

Third, I have a preference to certain degree for Android platform as how Open Source is involved, but discussing that here is beyond the scope of this project. Just for starting we chose to develop it for Android because was the faster and feasible way to us in order to generate fast results but in the future it may be coded too for other environments.

Finally, version 2.1 is old, but it is still a quite distributed version between android users¹, as there were no problems to make this application for that version it was a good idea to do it in order to maintain as much as possible the compatibility of older devices.

¹ ¹ Android version 2.1 is the third most used version as of today as of <http://developer.android.com/about/dashboards/index.html>

1.5 Description of the actual situation

First thing to notice in the world of architects is that there are several roles in their work, for example:

- *Valuer*: One who is responsible to evaluate properties and give an approximate price.
- *Issuer of occupancy*: One who is responsible to evaluate properties and issue the certificate of occupancy.
- *Inspector*: One who is responsible to evaluate the legal conditions of the property.
- *Interior designer*: One who tries to maximize or rearrange or decorate the available space inside of a property to meet the expectations of the customer.

All of them, in order to do their job needs the representation of the property into a 2D representation (sometimes in 3D), usually called a map. The most basic thing is to be able to have a map of the property from a point of view from above to bottom.

They usually perform this making photographs, drawing in papers, and sometimes using notebooks (or netbooks) with AutoCad (the main architect's software).

But the process is quite "manual", and later they need to put all this information together in order to use it.

1.6 Disadvantages. Room for improvement.

Architects are forced to carry notebook or netbooks if they want to fully take advantage of the informatics in their job at the "streets", which lots of them do not bring them. In some countries to have a notebook may be a normal thing, but not for this case.

AutoCad is a heavy application that usually needs a pretty powerful computer, depending of the severity of the property that one wants to represent. Usually architects have desktop computers which are powerful to do that job. There is also the explanation that not everywhere in the world they can afford for a notebook or netbook, and finally, when they need to work almost all day in the streets usually the battery autonomy of the notebooks restrict them.

So in the end, they usually take photos, and put all the information in physical format like the always reliable pencil/pen and paper.

But that can be improved if they could be able to gather all this information in one place, also this tool could support them in order to check further information readily available from Databases.

And so, here is the scope, and the main goal of this project. For the moment, this tool is not pretending to substitute any previous methodology, it is still far from it. But as a first prototype that can be fully expanded and test it is a good beginning.

2 Description of the actual requirements

After introducing the scope of this project, speaking about which tools we are going to use and for which purpose, now we are going to show in this chapter the complete study of requirements that our solution will have to accomplish.

2.1 *Requirement analysis*

Any solution, regardless of its kind, has to discover and describe before starting to specify nothing, which needs should be fulfilled. These needs are what is called project's requirements, and is the previous step before specifying in order to have a clear idea of what we want to do. We will find requirements of 2 types:

- Functional requirements: They are the inherent functionalities that the system will have to offer to the user; for example, show a map.
- Non functional requirements: they are the requirements not related with user functionalities.

2.1.1 Functional requirements

The desired solution should be helpful and useful for architects, being its strong point the availability to use it whenever it is needed without restrictions of hardware.

The bare minimum objectives to be useful for an architect are the followings: (already mentioned them on introduction chapter)

- Ability to use this application wherever as possible
- Ability to represent maps
- Ability to take and include photos
- Ability to save and load the data
- Ability to write descriptions
- Ability to check a list of elements for occupancy

2.1.1.1 Ability to use this application wherever as possible

One of the main goals is to help as many architects as we can, so the solution has to be intended for as many devices as possible. Also, these devices have to be as popular enough as possible.

With these premises, the most extended portable device would be the regular cell phone, but it is instantly discarded because there are not cell phones with touch screen capability. One critical point needed for interaction with this application. So, the next popular devices are smartphones.

In this category we have two major OS in the market, Android from Google and iOS from Apple, the best solution would be for both of them but as we do not have time nor resourced to do development for both devices at the same time we will focus on Android devices.

Alternative: Instead we could use tablet pc's, this would be an interesting alternative because tablet pc's have higher screen size and are portable, only problem, they are not as common as smartphones, so that's why we chose to use smartphone. Anyway this is not a problem at this stage because the application can be easily ported to table pc in the future if needed.

2.1.1.2 Ability to represent maps

As we stated on the chapter *1.5 Description of the actual situation*, all architects regardless of its role, need to represent the property that they are analyzing into a 2D surface, at least.

Our solution gives the user the ability to draw into the screen in order to recreate a map of the flat that he is watching. That's why we needed a touch screen and so a smartphone instead of a cell phone.

2.1.1.3 Ability to include and take photos

Architects works taking photos, in the interest of documenting pathologies the best way is to take a photo of the pathology itself.

Our solution gives the user the ability to take photos and includes them into the drawn map at the particular place where this pathology was, so he can look at it later in the future.

2.1.1.4 Ability to save and load the data

This point almost does not need justification, if architects wouldn't be able to save or load previous works the entire application would be useless because sooner or later the architect will exit the application and so it is needed to save all the progress and retrieve later if needed.

2.1.1.5 Ability to write descriptions

As architects made notes of the pathologies that they encounter, our solution will give the ability to write descriptions and explanations of the pathology that have been found, and it will offer too to add the affected element into the system, both information would be related to the photo taken before (the pathology itself), which was already related into the map.

2.1.1.6 Ability to check a list of elements of occupancy

Architects need to verify several characteristics of the property they are visiting in order to issue a certificate of occupancy. Our solution will give a checklist for each map in order to revise if the flat fulfils those requirements or not.

2.1.2 Non functional requirements

The non functional requirements that we want our application to comply are:

2.1.2.1 User interface and Human factor

As the intended application is for architects we have to realize that they are not informatics specialists, so the user interface has to be simple and easy to understand in order to be easy for use.

It will contain few screens, and almost everything will be performed using the same combination of buttons, option's menu. As for screen, there will be few of them just the required ones.

2.1.2.2 Efficiency

The application will be small enough to be fast, and all the data structures generated have been created thinking about it's global performance. As this will be used for devices that have small power of CPU (smartphones) and with low memory, this step is critical.

2.1.2.3 Extensibility

As this is a prototype its modular code has to be easy enough in order to extend its possibilities. This is after all, a first attempt of this type of application and so it can be much much extended in order to add new functionalities and extend those that are already working.

2.1.2.4 Maintenance

The solution has to be maintainable as it will be extended, so far this is fulfilled because it's not that big enough for the moment so it has been coded using a Facade pattern.

2.1.2.5 Others

We did not speak about **security** and **stability**, and that's because as it is a prototype unfortunately these two characteristics have been left a unattended. If the user wants to put input that it is not expected, the application more than probably will crash. This needs further refinement, but it was the least of the worries because we are dealing with architects which will try to use this application accordingly and as intended, and not trying to do some weird behaviours like a regular user would do. And as for the security, there is non... because there is no possible data corruption for the user, or possible leakage of data as far as maps is concerned. So the security level is predetermined of the Android device for which will work on, if it is infected by a virus we cannot do nothing against it.

2.1.2.6 Environment development and platform

The solution will work on Android systems from version 2.1 up until the newest one, this is intended for trying to please the maximum number of users. As for today android 2.1 is the third most popular android version, being 4.03 the second and 2.3 the first.

The environment development is effective as simple, Eclipse framework with the option to run the Android emulator for several phones.

2.1.3 Typical course of events from use cases

These are the main use cases that our application has to achieve after our analysis of requirements.

2.1.3.1 Draw Core

This is the main feature of our application; it allows the user to represent a property into a 2D surface through a touch screen. The user will be required to draw on it and the application needs to detect this movement and represent it. This functionality will offer various tools for the user in order to help him to draw as easy as possible.

Use case: Draw a wall

Actors: Simple user (starter)

Description: Represent a wall into the system through the movement of the finger.

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to draw a wall.	
2. He touches the screen.	
	3. The system catches the touch point and begins to draw a line following the movement of the user's finger.
4. User can see how his wall is being represented; eventually he stops touching the screen finishing its drawing.	
	5. The system detects the release of the touch screen from the user and makes persistent the representation of the wall through a line.

Possible errors and alternative courses of the use case.

There is no other course as for this use case; If the user does not like the represented wall, or wants to change it he can use another use case to change its representation or delete it

Use case: Delete a wall

Actors: Simple user (starter)

Description: Delete the selected wall from an existing map.

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to delete a wall.	
2. User touches the "Erase" option.	
	3. The system activates the erase mode.
4. User touches the wall to be erased.	
	5. The system detects which wall has to be deleted and remove its existence from the system.

Possible errors and alternative courses of the use case.

There is no other alternative course for this use case; if the user made a mistake he will need to draw again that wall using the previous use case.

Use case: Create a new map

Actors: Simple user (starter)

Description: Create a new blank map with no walls painted on it

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to create a new map.	
2. User touches the "New" option.	
	3. The system deletes everything previously shown on the screen and creates a new empty map.

Possible errors and alternative courses of the use case.

There is no other alternative course for this use case; if the user made a mistake he will lose all non saved information from previous map.

Use case: Draw a door

Actors: Simple user (starter)

Description: Represent a door into the map.

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to represent a door.	
2. User touches the "Door" option.	
	3. The system activates the door mode.
4. User touches the point of the particular wall where he wants to put a door.	
	5. The system catches the touch point and begin to delete any drawing behind it.
6. User can move its finger through the screen in order to increase or decrease the size of the door.	
	7. The system catches this finger's movement and represents it into the map.
8. User stops touching the screen.	
	9. The system stops deleting the drawing behind he finger's user and makes persistent the new resulted map.

Possible errors and alternative courses of the use case.

There is no other alternative course for this use case; if the user made a mistake he will need to draw again that wall or load the map again.

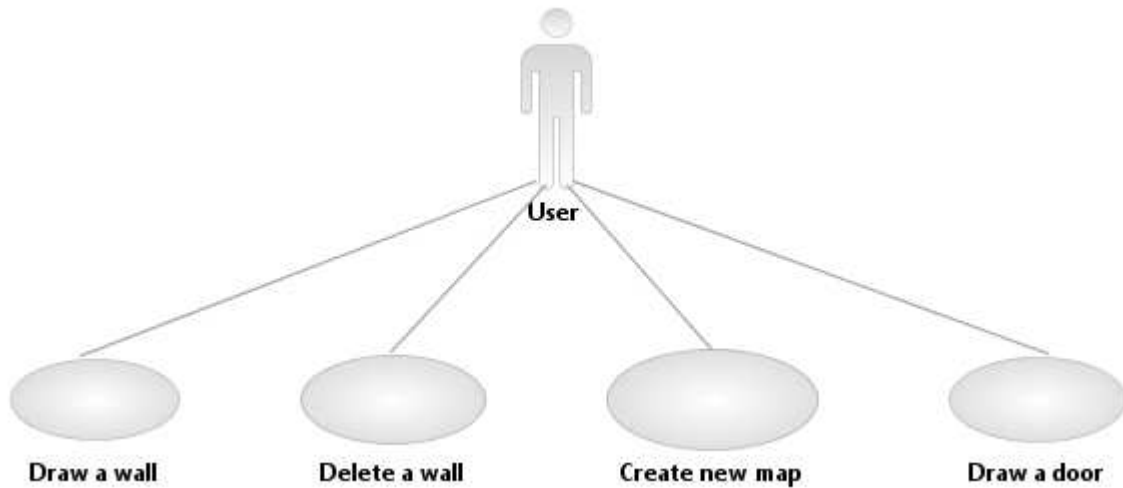


Image 2.1: Use case diagram of the Draw Core

2.1.3.2 Pathology Core

This feature will be responsible to allow the user to input and retrieve pathologies that he may encounter during his analysis of the property. Its main goal is to launch the camera application of the smartphone, take a picture and insert it into the map and the desired point where the pathology exists and write relevant information regarding it.

Use case: Input a pathology

Actors: Simple user (starter)

Description: Introduce a new pathology into the map

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User detects a pathology and wants to introduce it into the system	
2. User touches the "Select" option.	
	3. The system activates the select mode.
4. User touches the wall in which the pathology will be related.	
	5. The system detects the wall and launch a pop up window with information of the wall
6. User touches the camera icon.	
	7. The system launches the external camera application.
8. User takes a picture.	
	9. The system retrieves and save the picture into the system linking it into the selected wall. It repaints that wall with different colour to represent that wall has a pathology attached to it

Possible errors and alternative courses of the use case.

If something goes wrong like no camera application is detected there will be no picture attached to this pathology; It can be retried again later.

Use case: Input description of a pathology

Actors: Simple user (starter)

Description: A pathology description is inserted into the system

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to input a description	
2. User touches the "Select" option.	
	3. The system activates the select mode.
4. User touches the wall in which the pathology will be related. Black walls have already a pathology inserted so, will touch one of these.	
	5. The system detects the wall and launch a pop up window with information of the wall.
6. User touches the gallery icon.	
	7. The system launches the gallery window where the user can see all pictures of this pathology.
8. User selects the text field in order to insert the new description	
	9. The system opens up the virtual keyboard if there is no physical keyboard in this smartphone.
10. User types the new relevant information. And press Ok.	
	9. The system save this new information and close the gallery window, going back to the previous pop up window.

Possible errors and alternative courses of the use case.

User does not want to save the new description; then he only needs to push the back button and not the "Ok" button in order to not make any change.

Use case: View pictures of a pathology

Actors: Simple user (starter)

Description: Viewing all photos taken for a particular pathology

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to view pictures of a pathology.	
2. User touches the "Select" option.	
	3. The system activates the select mode.
4. User touches the wall in which the	

pathology will be related. Black walls have already a pathology inserted so, will touch one of these.	
	5. The system detects the wall and launches a pop up window with information of the wall.
6. User touches the gallery icon.	
	7. The system launches the gallery window where the user can see all pictures of this pathology.
8. User touches the pictures to right or left to see more pictures.	
	9. The system shows the new pictures as requested.

Possible errors and alternative courses of the use case.

There is not possible error during this action.

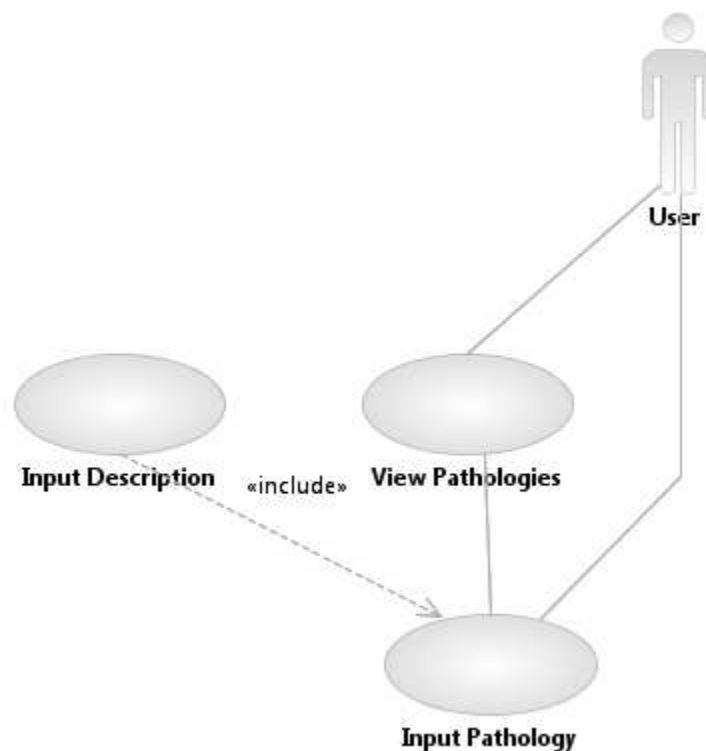


Image 2.2: Use case diagram of the Pathology Core

2.1.3.3 Checklist Core

This feature will be responsible to allow the user to input and retrieve a list of checked elements that the property accomplish (or not) to have. Its main purpose is to aid architects of elements to be checked.

Use case: Check the list of elements

Actors: Simple user (starter)

Description: Checklist of the elements to be revised from the property

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to check the elements of the property.	
2. User touches the "Checklist" option.	
	3. The system launches the checklist window related to this map.
4. User can see, check or uncheck the desired elements. And press Ok.	
	5. The system saves these data and makes it persistent into the system.

Possible errors and alternative courses of the use case.

If the user does not want to make any changes it only needs to push the back button instead of "Ok" button.



Image 2.3: Use case diagram of the Checklist Core

2.1.3.4 Persistent Core

This feature will be responsible to allow the user to save or retrieve the data from the system. Mainly the maps that have been created.

Use case: Save current map

Actors: Simple user (starter); system

Description: Save all relevant data of the current map

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to save the current map.	
2. User touches the "Save" option.	
	3. The system saves all data related to this current map. If it already existed before will replace it for this new one, if it didn't exist before it will create all the data structure to make it possible.

Possible errors and alternative courses of the use case.

No enough space storage or no SD card inserted will throw exceptions; user will be expected to solve these problems, at this prototype stage.

Use case: Retrieve a previously saved map

Actors: Simple user (starter); system

Description: Load the selected map previously saved

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to load a previous map.	
2. User touches the "Load" option.	
	3. The system launches a Load window with a list of all the current saved maps.
4. User touches the selected map.	
	5. The system close this window, and loads the selected map as it was the last time was saved.

Possible errors and alternative courses of the use case.

No possible errors with this use case, if there is not previous saved maps the user will not be able to load a map.

Use case: Clear database

Actors: Simple user (starter)

Description: Deletes all the data from the application

Typical dialog between actors and system and resulted produced changes:

User actions	System's answer
1. User wants to delete all data	
2. User touches the "Clear Data" option.	
	3. The system deletes and removes all data from the application.

Possible errors and alternative courses of the use case.

If the user mistakenly chose this option, there will be no recovery. This should be extended in the next step of the prototyping.

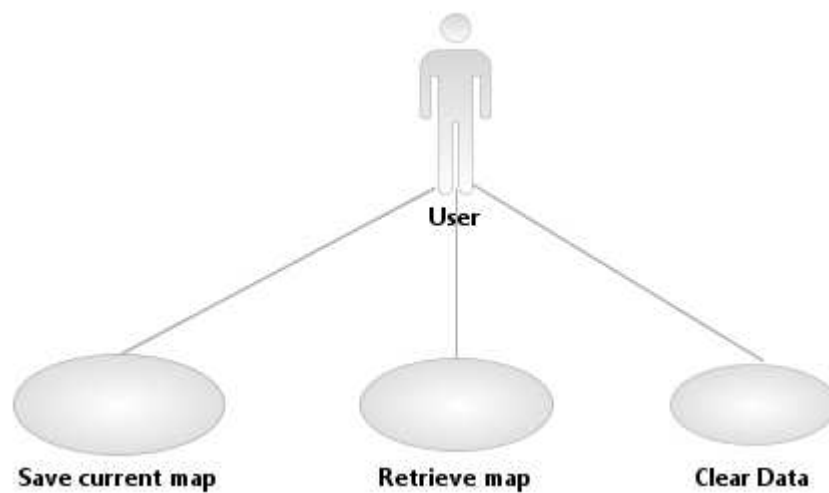


Image 2.4: Use case diagram of the Persistent Core

2.2 Problem specification

In this section we are going to detail which are going to be entities that will intervene and what they will have to exactly do. To make the specification we have to chose a methodology in order to guide us through all the construction process.

The architectonic pattern that we chose is the **N-layer architectonic pattern** and **oriented model objects**

First one is because is one of the most extended methodology and is quite natural to develop into it, second one, even I support for it in order to make more modular codes, it's a thing that we cannot change because Android is based on Java language and de facto this is oriented object language.

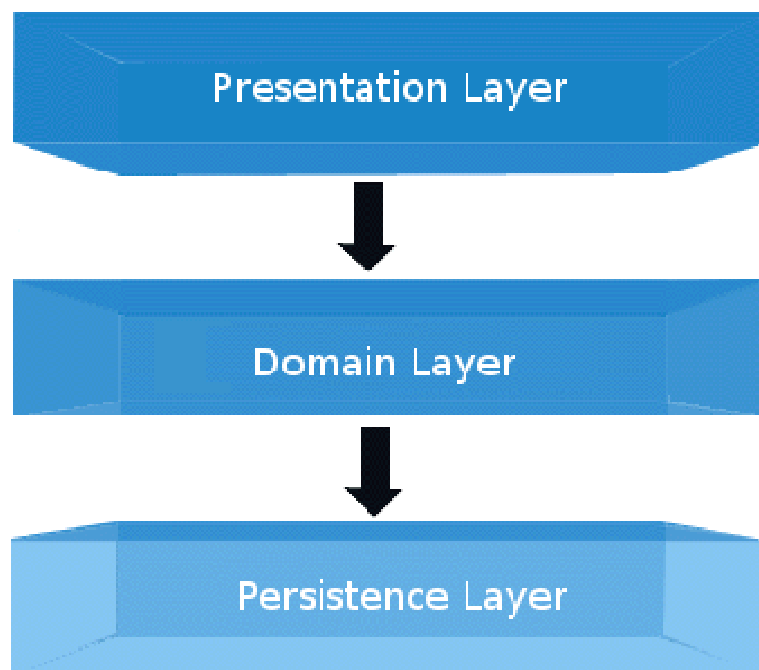


Image 2.5: 3-layer architectonic pattern

Our application will be based on the design of the standard 3-layer pattern that we can observe on the above image. But with some considerations::

- 1) The presentation layer will be just the design of the screens of the Android application.
- 2) The persistence layer will be just something simple and small in order to save the data.

2.2.1 Conceptual data model diagram

The solution will be formed as a set of modules implemented independently who they will interact with each other in order to accomplish the desired goal. Presentation layer

2.2.1.1 Presentation layer

The presentation layer is the one responsible to interact with the user. In our solution it is something granted because we will be using an smartphone screen to interact with the user through its touch capabilities. Besides this layer is quite simple, we are going to show the different windows needed to have to achieve our goals.

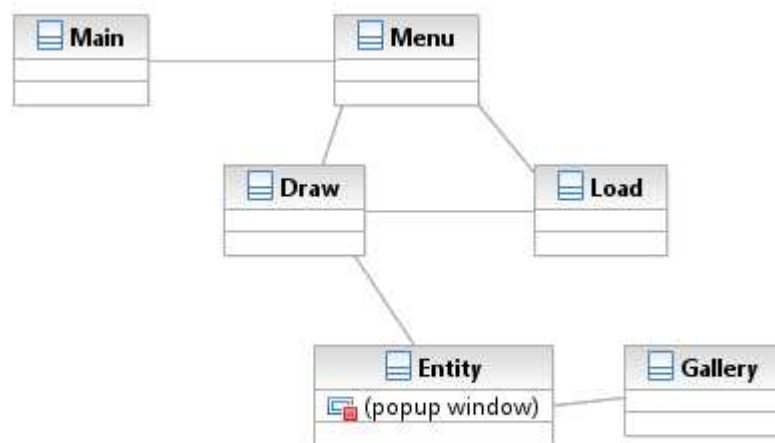


Image 2.6: UML conceptual data model of Layer Presentation

After the main screen, which will be a transitional screen we will reach the menu screen, from there the user can chose if wants to draw or load a previous map.

In Draw screen we can navigate to Load screen, or to the Entity screen, the screen that shows information of the walls painted in the map, in the future will be considered to be a popup window in the Design phase. Then from this window the user can navigate to the Gallery window where the Pathology Core is located.

2.2.1.2 Domain layer

The domain layer contains the logic of the application. The classes that we find here are responsible to maintain the control of the application and its necessary data structure in order to maintain the data in execution memory.

2.2.1.2.1 Draw Core

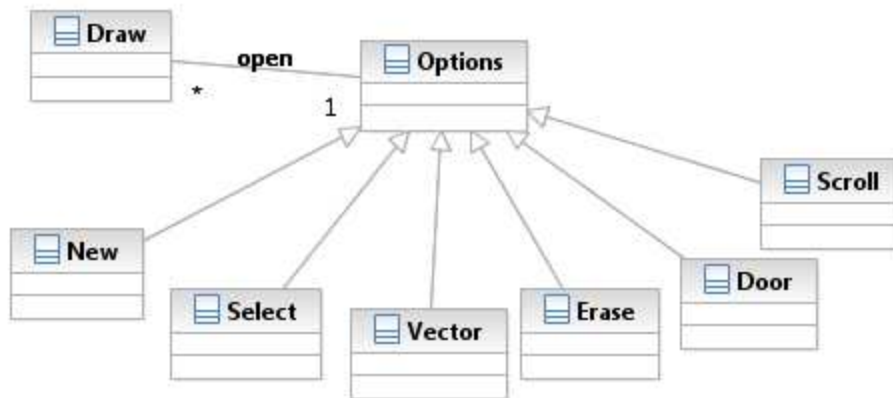


Image 2.7: UML conceptual data model of Draw Core Domain Layer

Draw class is the responsible to have the maps represented into the screen, the user will be almost all the time interacting with this class. Options class will be executed each time the user pushes the option button, and it will offer several options:

- New: Responsible to create a new "Draw" class.
- Select: Responsible to detect the entities drawn (walls).
- Vector: A helpful mode for the user to draw maps which only allows the user to draw straight lines.
- Erase: Responsible for deleting the entity selected.
- Door: Responsible for drawing doors in the map.
- Scroll: Responsible to change the current view of the map.

2.2.1.2.2 Pathology Core

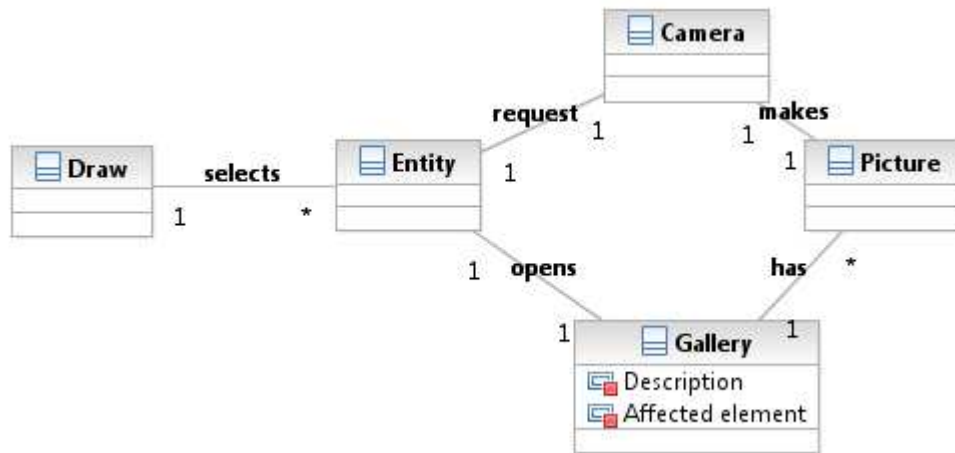


Image 2.8: UML conceptual data model of Pathology Core Domain Layer

We already explained about the Draw class before, it is here to note where the Pathology Core begins. In Draw class the user selects and entity, so the class itself communicates with the Entity class. Here is the description of the new classes:

- Entity: Responsible to show all the important data from the selected entity.
- Camera: This one is not a class of our application, is external, but needed in order to take a picture.
- Picture: The photo itself.
- Gallery: Class responsible to show all the pictures related to this entity and to show the description and elements affected by this pathology.

2.2.1.2.3 Checklist Core



Image 2.9: UML conceptual data model of Checklist Core Domain Layer

This core is quite simple as it is now, but in the future could be much more complex, we will explain on that on the chapter *Possible Improvement*. All maps will have a Checklist linked to them, so the Draw Class will communicate to the Checklist class responsible to show all the current elements that the flat complies (or not complies).

2.2.1.3 Persistence layer

The persistence layer are composed of all the classes responsible to make persistent the data of the application when this one it is not being executed and so retrieve it later.

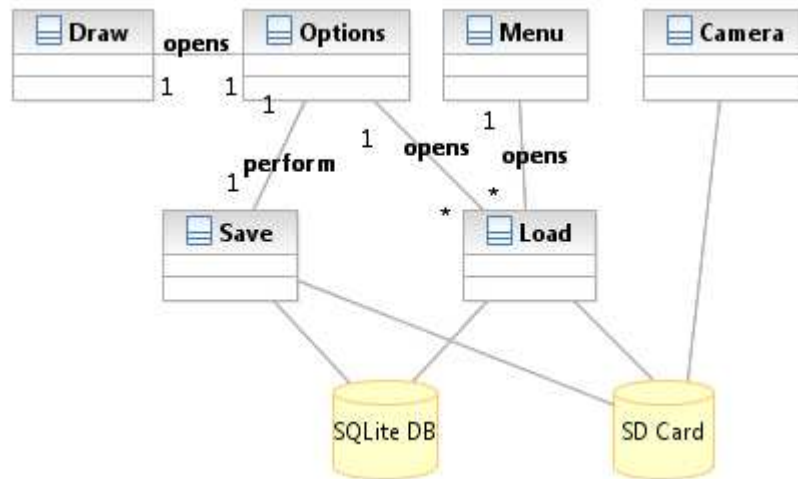


Image 2.10: UML conceptual data model of Persistence Layer

For tracking purposes we represent the Draw, Options and Menu Classes for clarification of how the classes interact with the persistence layer.

As one may guess the Save class is responsible to save all the data structure into an SQLite database for data structures, and into SD Card for bitmaps representing the maps being drawn.

The Load class is responsible to load all this data either from SQLite DB or SD Card, and finally we have the Camera Class, which is not a class of our application, but an external one who also saves the picture into the SD Card.

2.3 Other solutions description (commercial ones)

There are few applications regarding to help architects for small devices, but the most similar to the concept we are searching is OrthoGraph, only problem with this application is that its intended for iPad and costs 74€ (around 290PLN).

In reality as we explained in the introductory chapter there are not many applications as this.

3 Design and Implementation

3.1 *Proposed design*

As this is a first prototype we decided to be it as useful as possible, so there are not so many special features that made this application beautiful but rather practical and simple. A few windows that share the same design are been implemented and we are going to present them in the following sections.

3.2 *Architect Support Tool*

This is the name of the application and we are going to describe how it works.

3.2.1 Main and Menu screen

This is the first screen presented to the user, where he can choose to Draw a map, Load a previous saved map, or Clear all the data of the application.

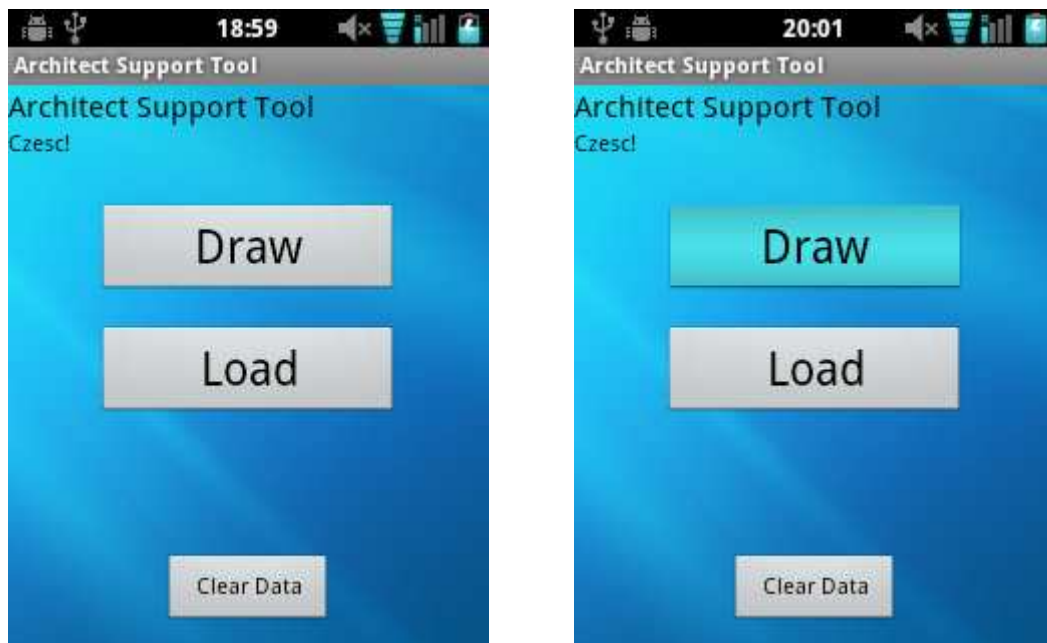


Image 3.1: Menu screen of AST

In this occasion the user has touched the Draw button.

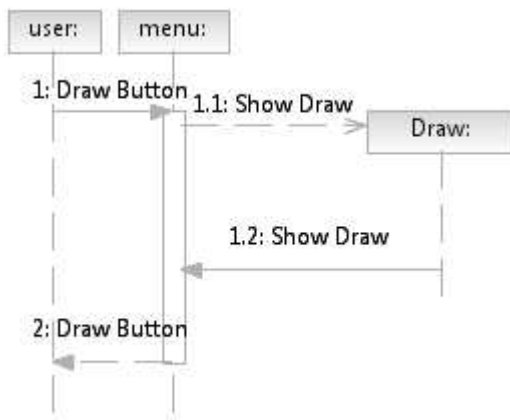
Sequence diagram involved:

Image 3.2: SD of Draw Button

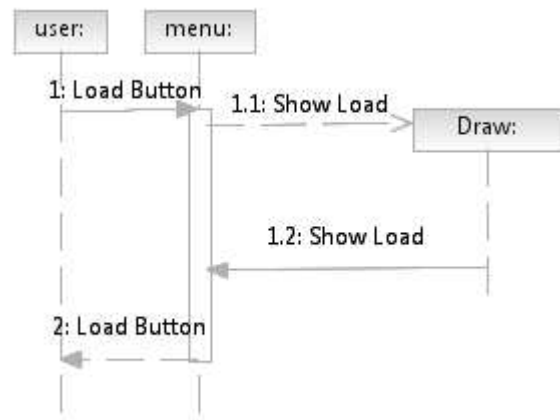


Image 3.3: SD of Draw Button

3.2.2 Drawing screen

This is the drawing screen, in this example we are already seeing a drawing map, that's because the system will load the last previous map, if there is no previously saved map in the system then there will be no map drawn in the screen.

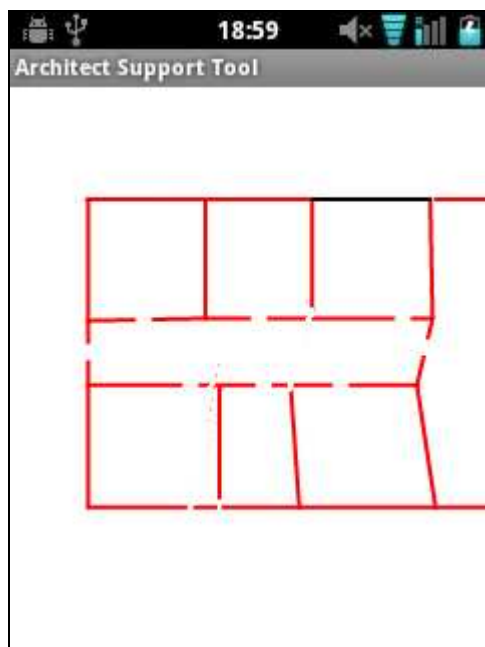


Image 3.4: Drawing screen

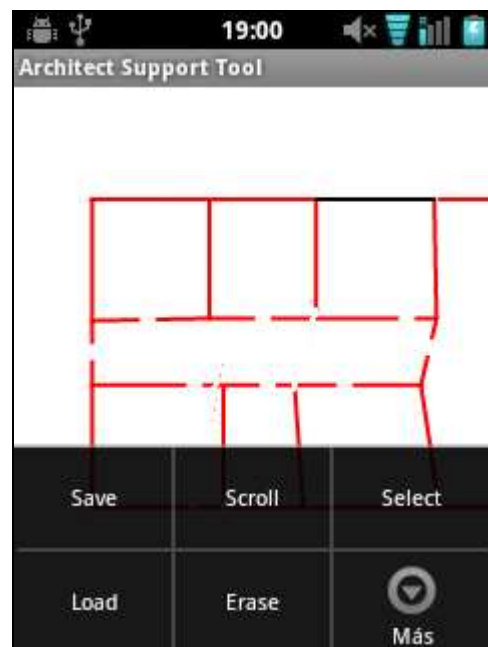


Image 3.5: Option menu

We can see here how the user push the "home" button of the smartphone causing this option menu to appear from the bottom of the screen.

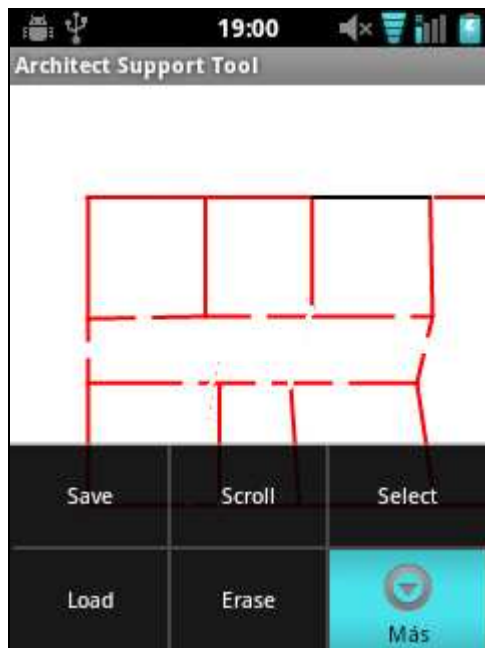


Image 3.6: Option menu "more"

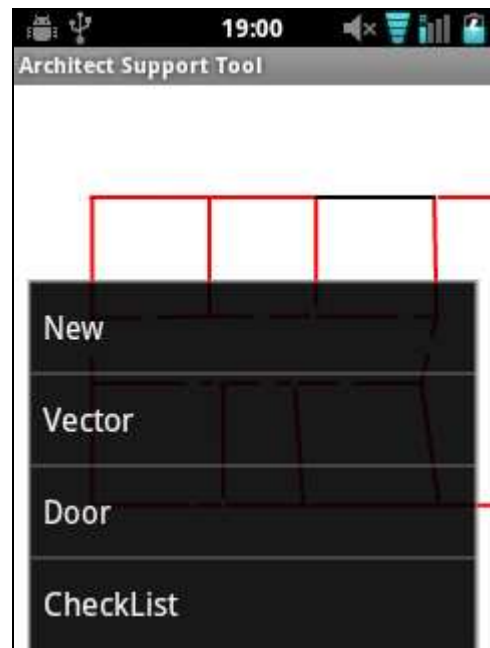


Image 3.7: Option extended menu

If the screen is too small to show all the options, it will give the option to touch an special button called "More" (depending on the Android native language, in this picture is Spanish language, "Más") which will result into an extended list of options that couldn't be displayed before.

Sequence diagram involved:

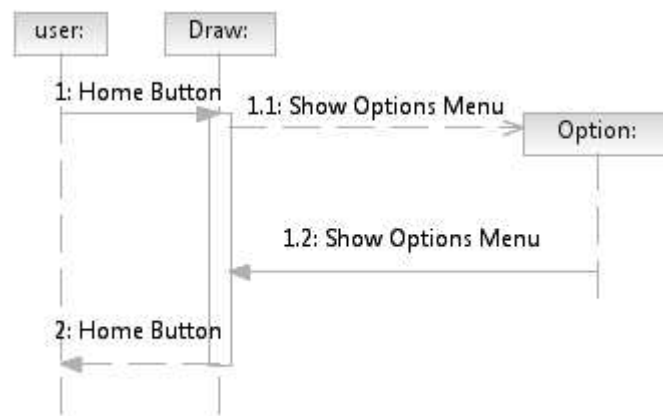


Image 3.8: SD of Option menu

3.2.3 Entity screen

Let's going to describe this screen with a practical example, let's say that the user have found a pathology in this encircled wall.

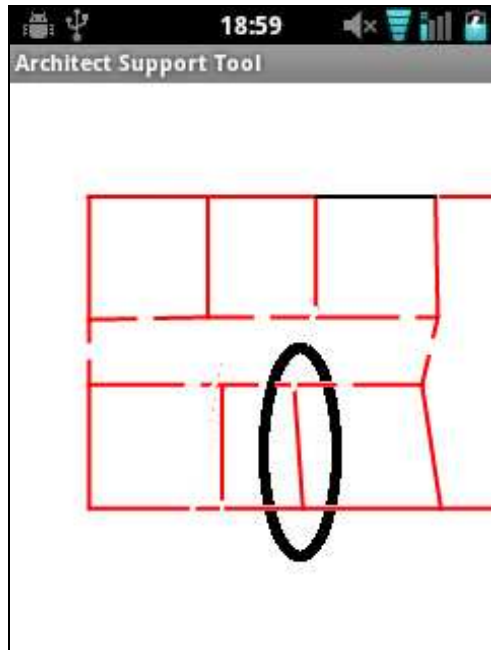


Image 3.9: Practical select example

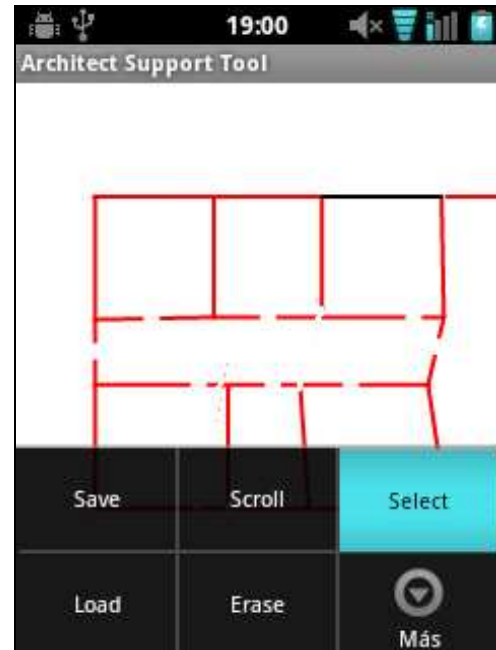


Image 3.10: Practical select example

All he has to do is to select the "select" option from the option menu, now he is on "select mode" so he touches that particular wall, and so the Entity screen (or also called "select screen") appears.

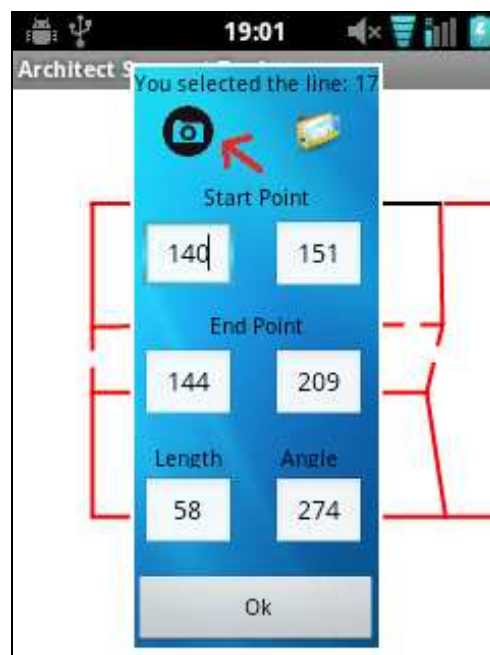


Image 3.11: Popup entity window

The entity window is a popup window, it has been decided as this because it's good to remember the user that this is just one of many windows (as many as entities there are on the current map) and he will not lose focus of the current map. We take notice on that camera icon (a red arrow is pointing into it), the user will touch it in order to launch the internal camera and make a picture of the pathology and so save it into this current entity.

We can also see that this window has some information:

- Start Point: The (x,y) coordinates in which the wall begins.
- End Point: The (x,y) coordinates in which the wall ends.
- Length: The actual length of the wall expressed in decametres (1 meter = 10 decametres), the reason is because for the moment there were not enough time for translating floating points meters into the system. Instead of saying 9,5 meters the user will be required to put 95 decametres.
- Angle: The relative angle of the wall.

The parameters **Length**, and **Angle**, can be modified by the user to meet new criteria, and the system will redraw the new wall as the desired result. This ensures more flexibility of the drawing capabilities of the application.

After the user has made a picture the resulting map will be displayed like this, note the change of colour of the wall, **red** means no pathology inserted, **black** means that wall has pathology.

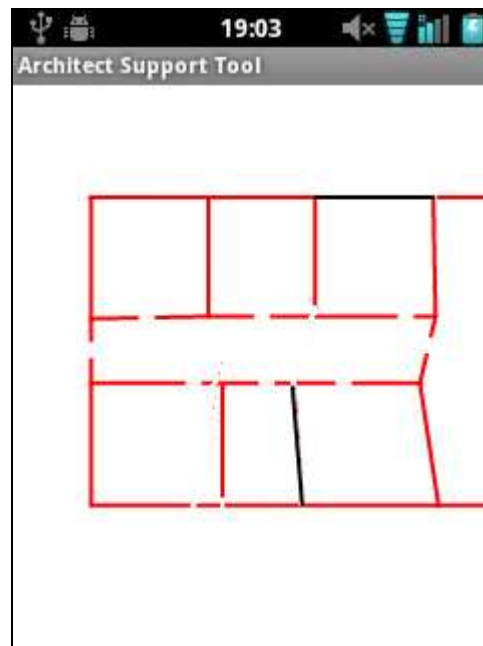
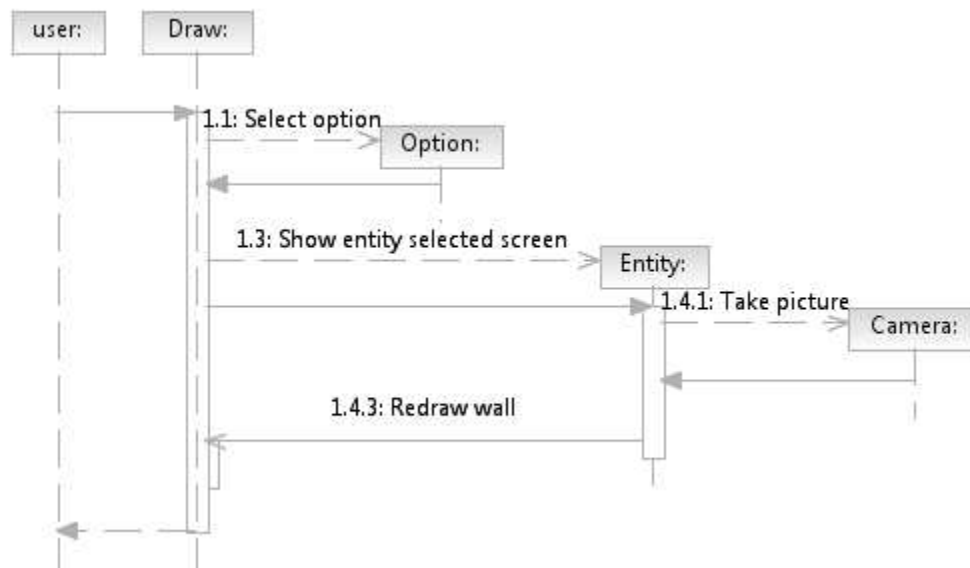
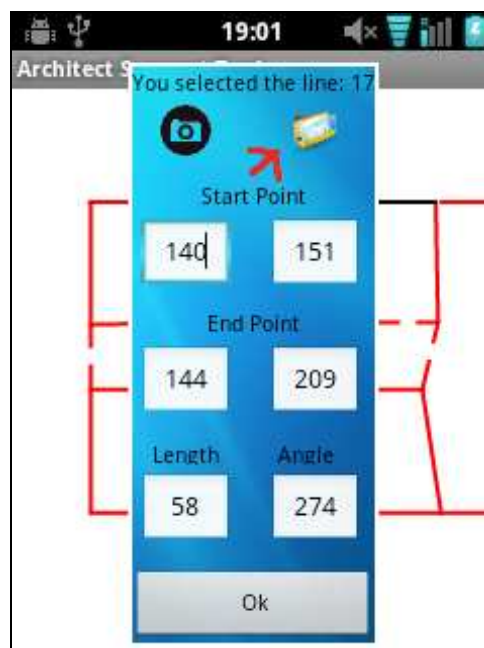


Image 3.12: Black wall after inserting pathology

And so now we can appreciate just by looking that this flat at least has two pathologies encountered at the black walls.

Sequence diagram involved:**Image 3.13: SD of Option menu****3.2.4 Gallery screen**

Now let's make the assumption that the user wishes to see the pictures taken with the camera of a pathology and (or) wants to put relevant information into it, in order to do so, he needs to do the same steps in the previous section. That is, open the Entity window of a wall that is painted in **black** (which means it has an inserted pathology).

**Image 3.14: Gallery icon in entity window**

The user then will touch the gallery icon on the top and right corner (a red cross is pointing into it), in order to open the Gallery window.



Image 3.15: Gallery window



Image 3.16: Navigating between photos



Image 3.17: Navigating between photos

The user then can see the pictures and in order to see more of them (if they exists) the only thing he needs to do is to touch the picture and move it into the right or left, as an standard gallery motion. In this example, we can see a wall with humidity problems.

Now he wants to input relevant information of this pathology, such a description and the elements that are affected. If he scrolls down the window (with the movement of the finger) he will find all the possible text fields to fill.

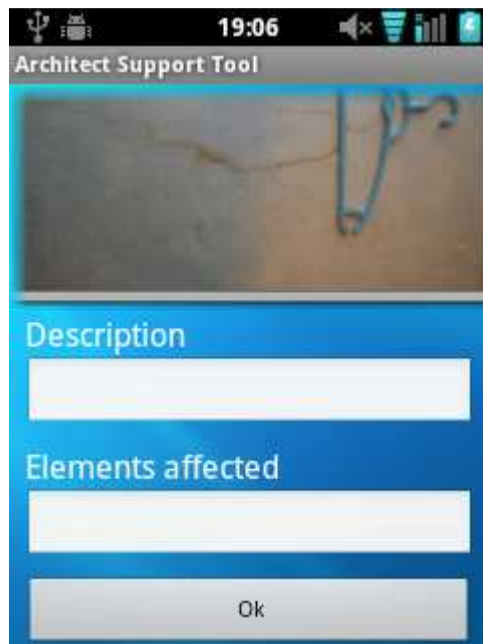


Image 3.18: Gallery text fields



Image 3.19: Gallery input method

So, when he touches one of the fields, the virtual keyboard will appear (if there is no physical keyboard in the smartphone) allowing the user to write a description of the detected pathology.

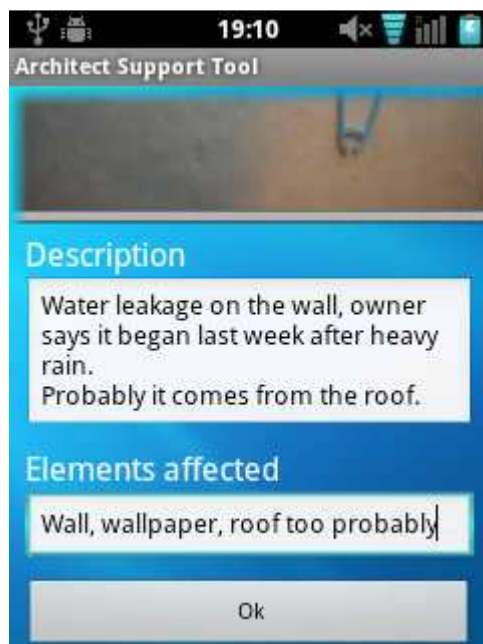


Image 3.20: Gallery filling info

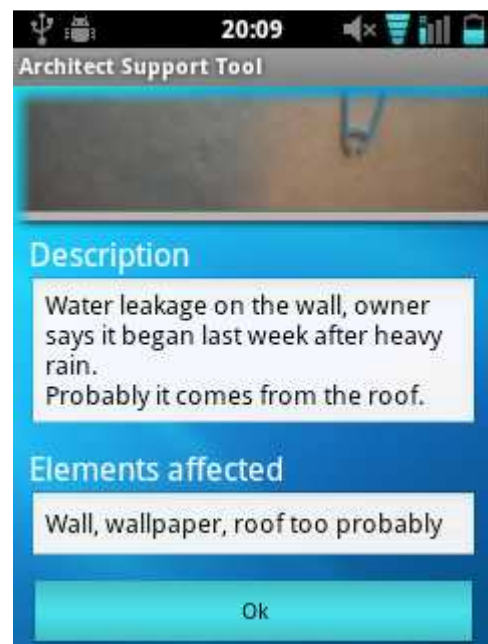
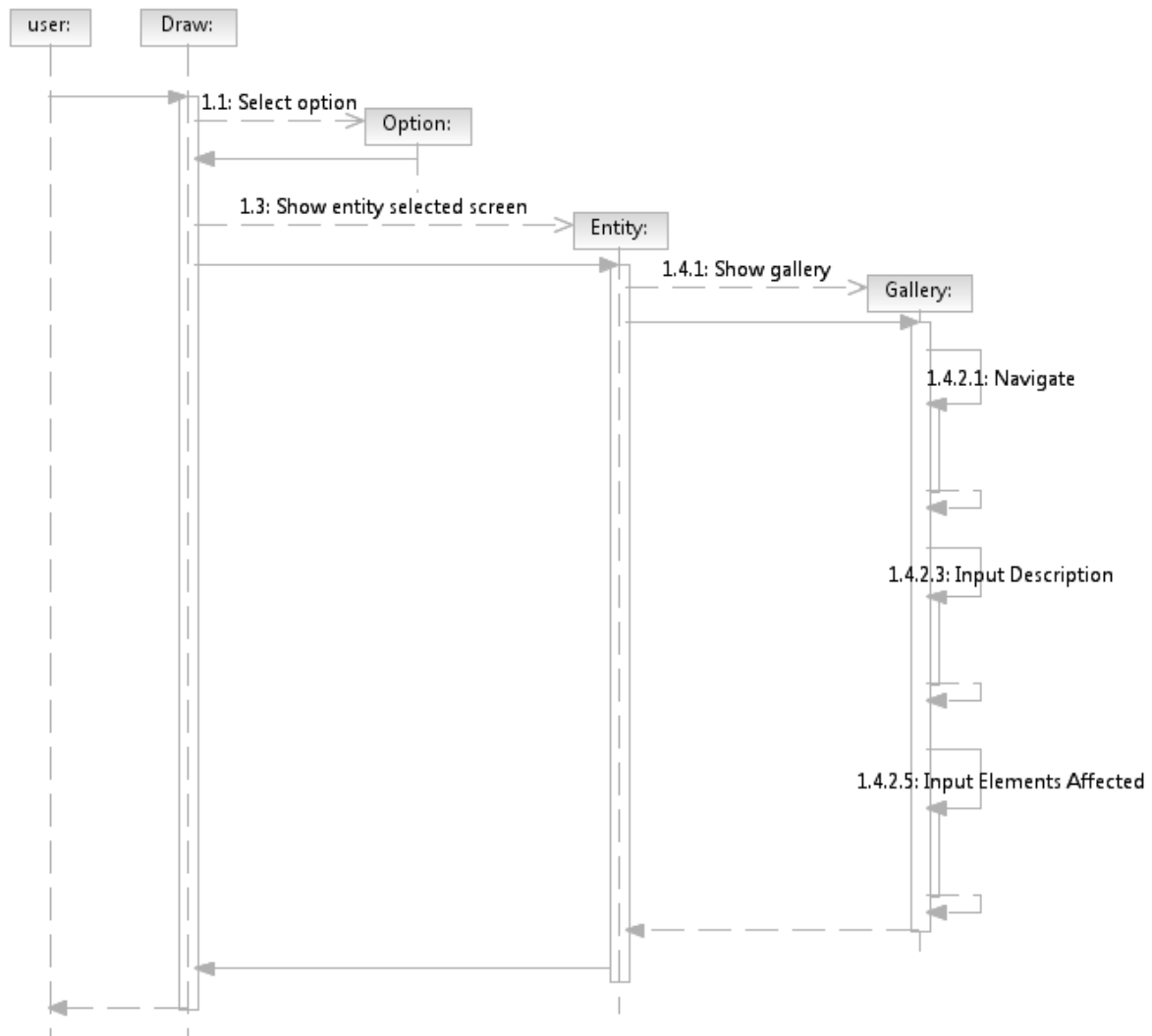


Image 3.21: Gallery accepting changes

When the user inserted all the desired information, he only needs to push the "Ok" button to save it. If he does not want to save it then he needs to push the back button.

Sequence diagram involved:**Image 3.22: SD of Gallery Window**

3.2.5 Scrolling

This feature wasn't described on the initial Analysis of requirements but soon it was clear that it would be needed in order to see the entire map.

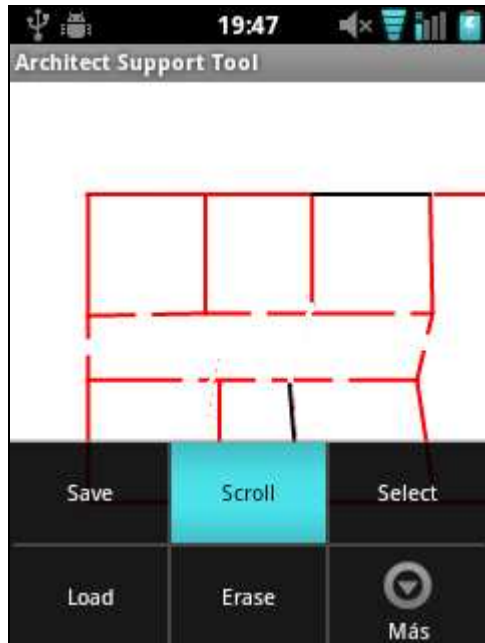


Image 3.23: Option menu "Scroll"

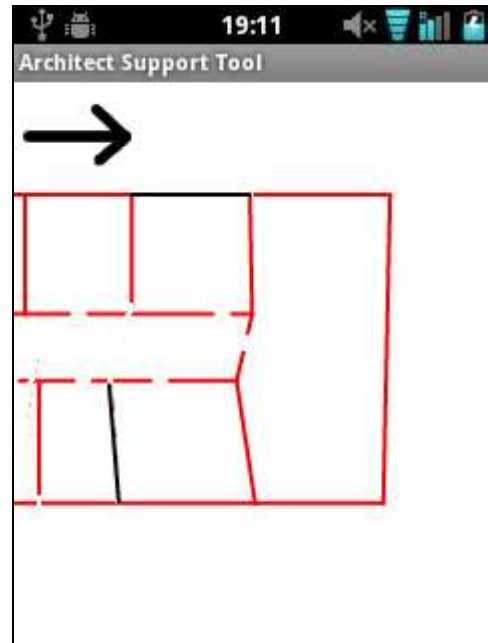


Image 3.24: Scrolling map

After when the user selects the Scroll button of the option menu, he will be on "Scroll Mode", now he can touch the screen and move his finger in order to move the map with its movement at the desired place. When he will finish he will need to push the scroll button option again to leave from the "Scroll Mode".

Sequence diagram involved:

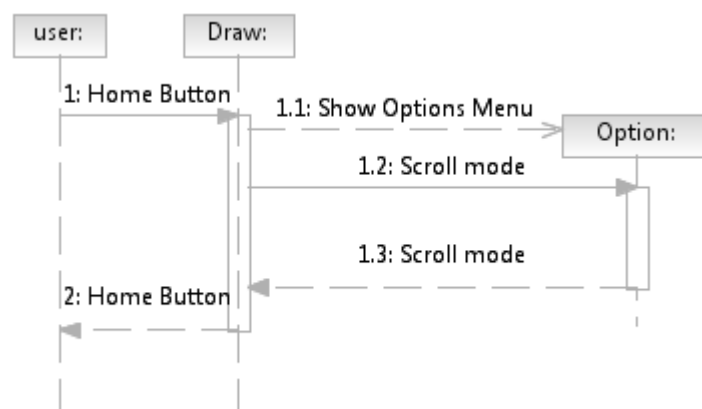


Image 3.25: SD of Scrolling

3.2.6 Saving

A must needed functionality for any application, in this prototype is easy as simple it is, the only thing that the user needs to do is to push the button Save from the Options menu.

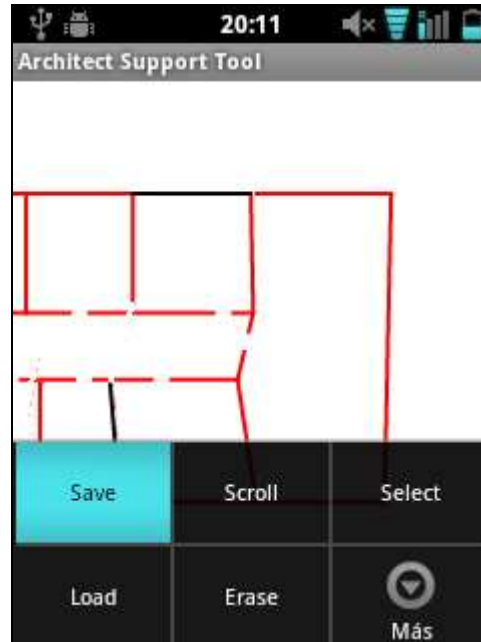


Image 3.26: Option menu "Save"

The save option will save the current map in the SQLite Database for data and into SD card for the current map (treated as a bitmap). Originally was thought to save everything on the SQLite database after converting the bitmap map into a serialize byte[] object, but it consumed too much memory and was slow.

Every time that the user will push the save button, the system will try to find this current map in the database, if exists, it will replace it for the new one, if not, it will create a saved new map.

In this prototype the user is not asked for a name, all maps will be saved with the notation of "House ID" followed by the ID number of the map assigned from the beginning as being unique.

Sequence diagram involved:

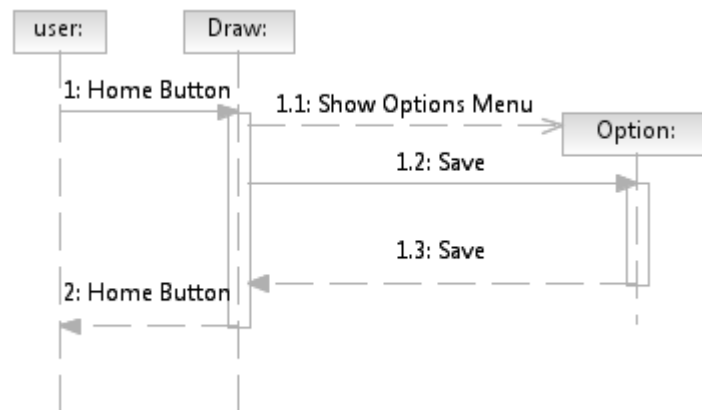


Image 3.27: SD of Saving

3.2.7 Loading

It would not make any sense if after saving the data we couldn't retrieve it so here is the load functionality of the application. We have to note that the Load screen can be reached either by touching the Load button from the Menu screen or from the Option Menu of the Draw screen.

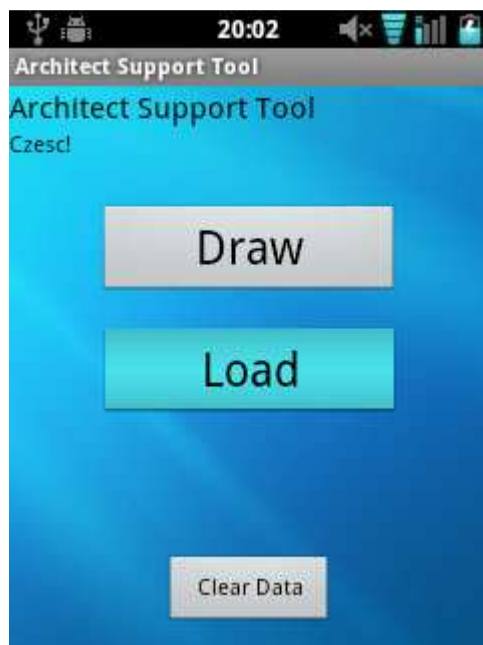


Image 3.28: Load button Menu Screen

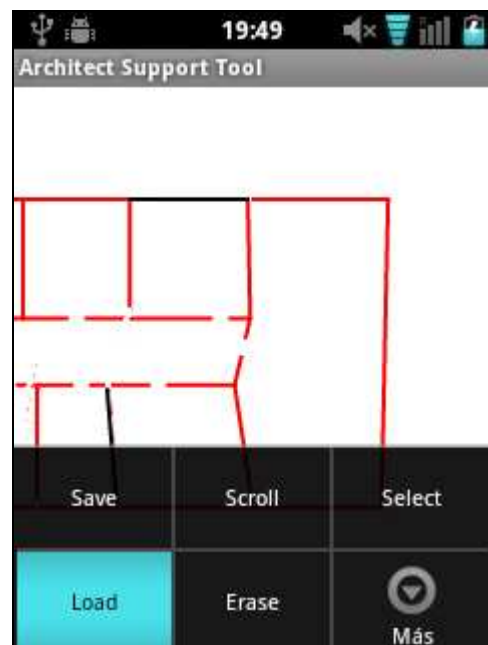


Image 3.29: Load button Option menu

When the user touches one of these buttons the Load screen will appear.



Image 3.30: Load screen



Image 3.31: Selecting from Load screen

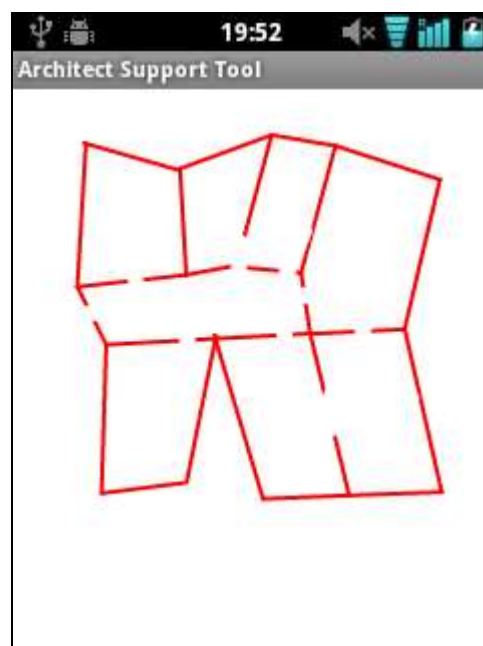
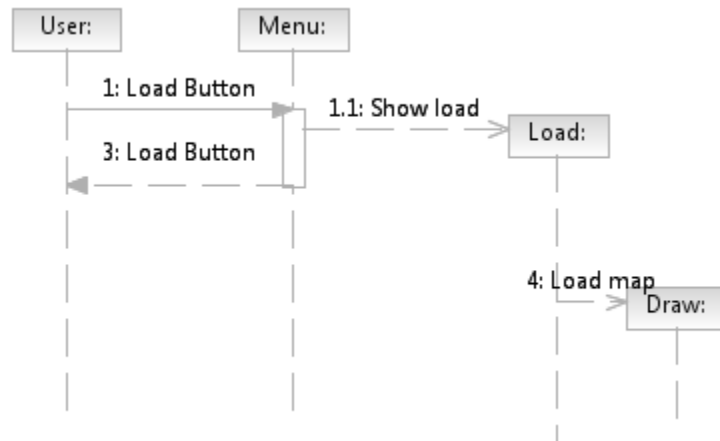
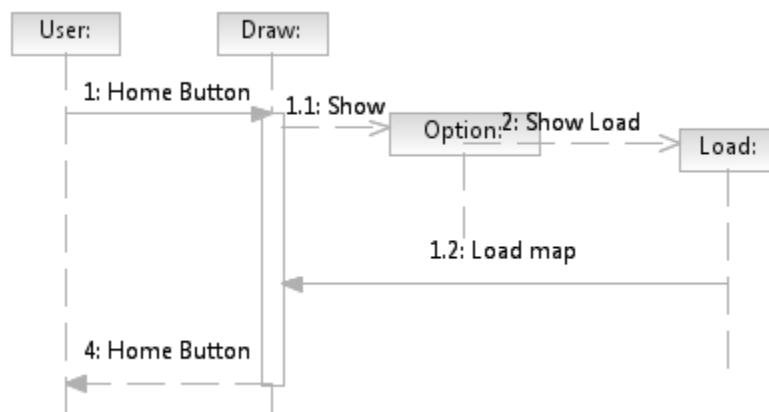


Image 3.32: After loading the selected map

In the load screen it will appear all the saved maps that exist in the current system, if there is none then this screen will be just empty expecting the user to go back pressing the back button. The user then can select any of the maps shown to be saved and load it.

Sequence diagrams involved:**Image 3.33: SD of Loading from Menu screen****Image 3.34: SD of Loading from Draw screen**

3.2.8 Create a new map

The user has the option to whenever he wants to create a new map disposing of all the current data displayed on the screen.

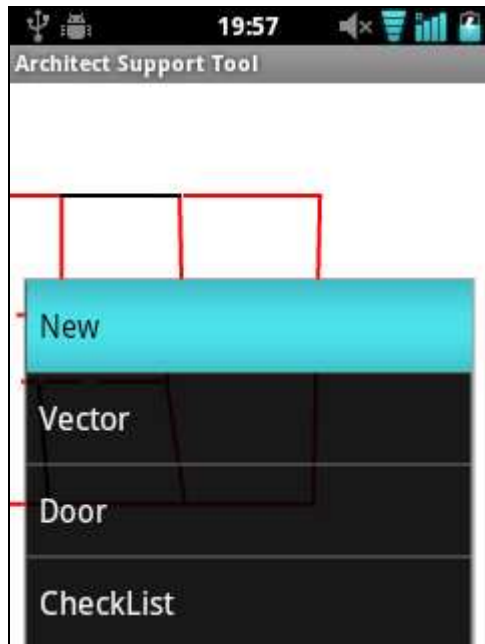


Image 3.35: Option menu "New"



Image 3.36: New blank map

Now he is able to begin a new drawing with new structures. This process is the only way to create new maps and increase the ID identifier of each map.

Sequence diagram involved:

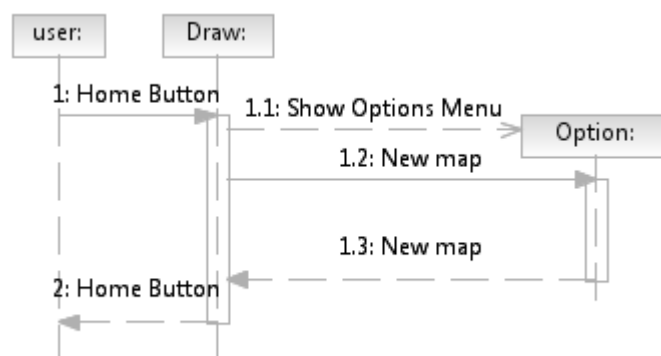


Image 3.37: SD of Loading from Draw screen

3.2.9 Erase a wall

Let's say that the user begins to draw quite a strange room, but he fails to connect the last wall to close the room.

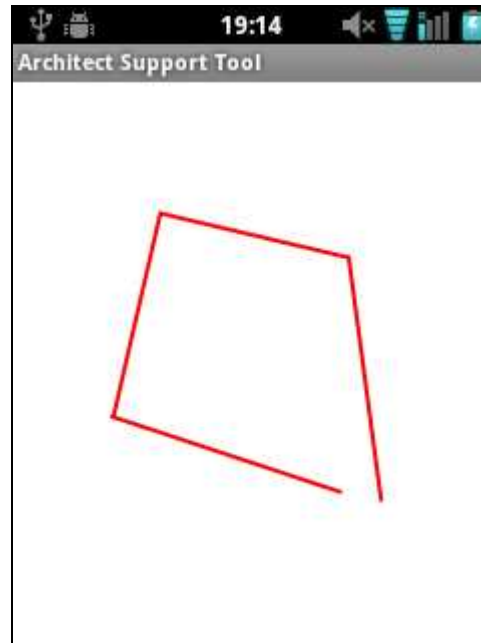


Image 3.38 Mistakenly placed wall

So he can select the erase option from the menu and select that wall to remove it.

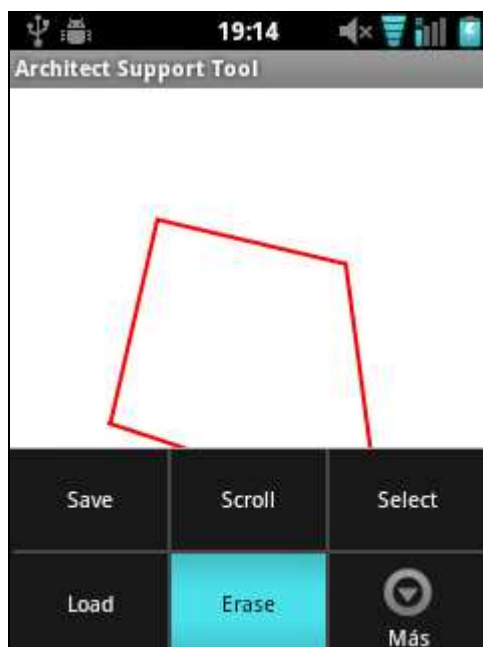


Image 3.39: Option menu "Erase"

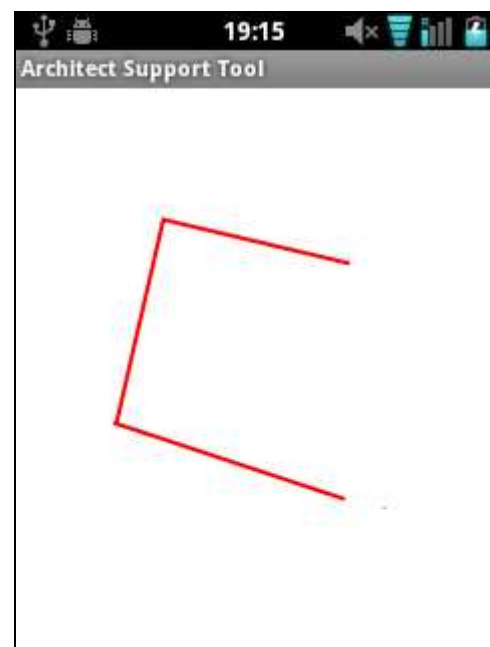


Image 3.40: Wall being removed out

When the user touches the Erase option he enters into the "Delete Mode", in this mode any entity that he will touch will be removed it. After removing the desired wall, now he can try again to draw the wall as he desires. The user needs to touch the Erase option again in order to leave the "Delete mode".

Sequence diagram involved:

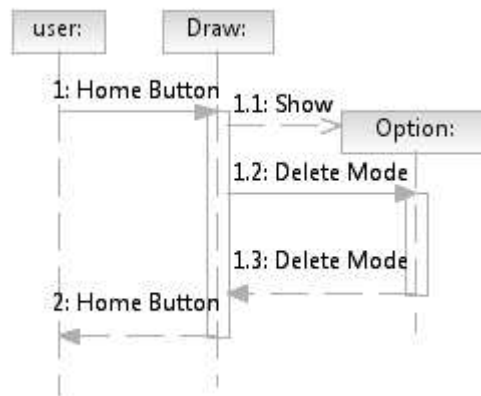


Image 3.41: SD of Erase a wall

3.2.10 Connecting Points Core

Taking the same example as before, as we stated, we cannot expect a human person to be as precise to select exactly the point where he wants to draw, that's why the Draw Core has a Connect Point Core in order to facilitate that task. It catches the point where the user touched the screen (or stopped touching the screen) and begins to search from all the entities that already exist in the system, if there is one point near to connect them.

This tolerance distance is set up as 20 pixels radius away.

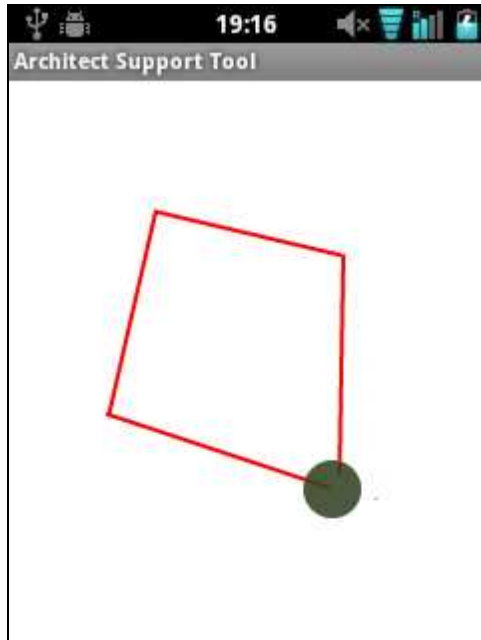


Image 3.42: Area of tolerance to connect points

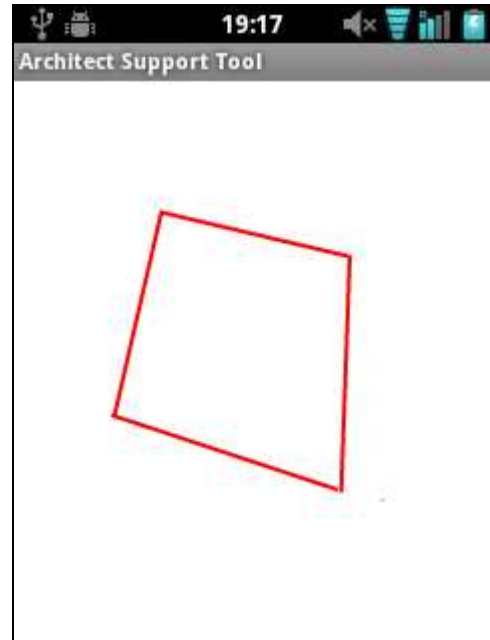


Image 3.43: Line after connecting point

For the moment, the system works out with the opposite points of the wall, but it is not capable to detect the points between them. So drawing a wall intersecting another wall will cause that wall to trespass the second one.

There is no sequence diagram for this, because it's an internal process, transparent to the user.

3.2.11 Vector mode

This is another feature in order to help the user to draw maps, until now we have seen that the user can draw straight lines with free angle (they can be 15 degrees, 25 degrees, whatever, as long the finger is moving through the screen). Sometimes this is impractical if one wants to draw rooms and flats in a faster way.

The Vector mode enables the user to only draw straight lines at 0° , 90° , 180° and 270° , as this application is not intended (for the moment, at this current prototype stage) to be a realistic representation of the world, but instead, a support tool, we think the vector mode will cover the needs of drawing rooms and flats, most of time.

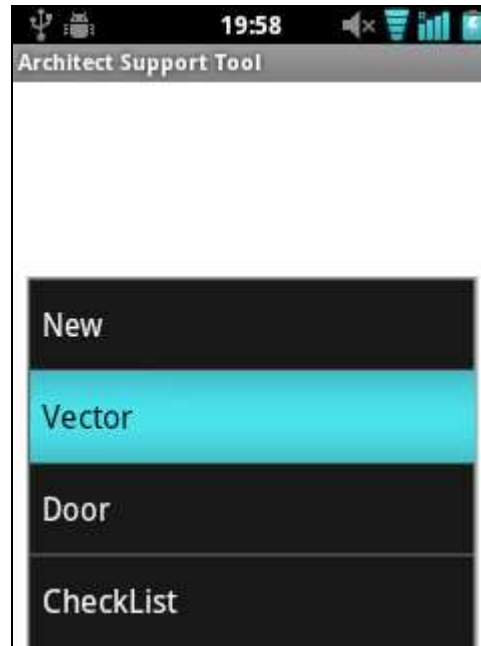


Image 3.44: Activating Vector Mode

As usual, this option is enabled through the option menu, and in order to deactivate it one has to touch it again.

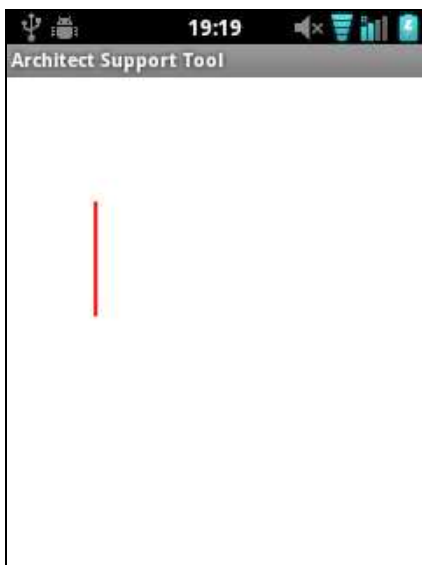


Image 3.45: Wall at 270°

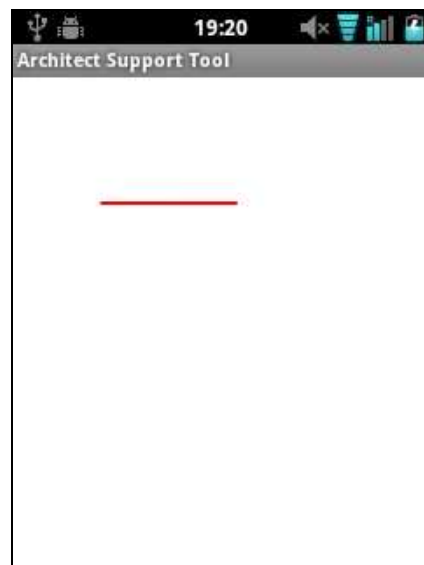


Image 3.46: Wall at 0°

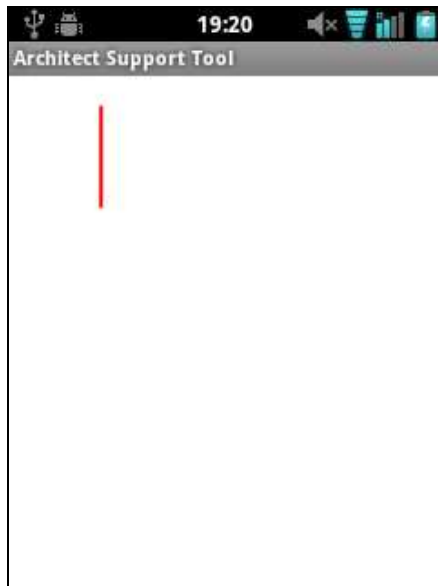


Image 3.47: Wall at 90°



Image 3.48: Wall at 180°

We have here what is represented on the screen when the user touches the screen, but maintains the touching and moves his finger around the screen, all the time the wall is represented into one of x or y axis. When the user will release the touching of the screen the last motion of wall will be finally drawn.

Sequence diagram involved:

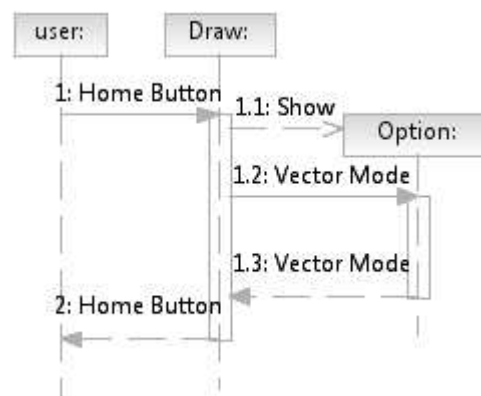


Image 3.49: SD of Vector mode

3.2.12 Practical example

We almost explained all the features except the Door option and Checklist option, that we will explain after this example.

As we already explained with these features one can draw maps quite effectively and we are going to make a real case example.

We are again with a user who is an architect and is an inspector, trying to insert pathologies in a new flat, so, with that premise he will begin to draw the map, using the vector mode to be fast and at the same time accurate.

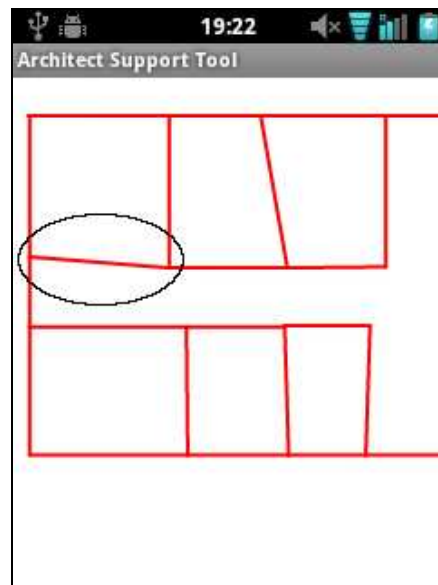


Image 3.50: Wrong drawn wall

After drawing the map, it is expected that it will not be perfect and there will be some sections that need to be improved, fortunately the application gives tools in order to fix this situations. Let's say that our user wants to correct that wall because it is not straight enough.

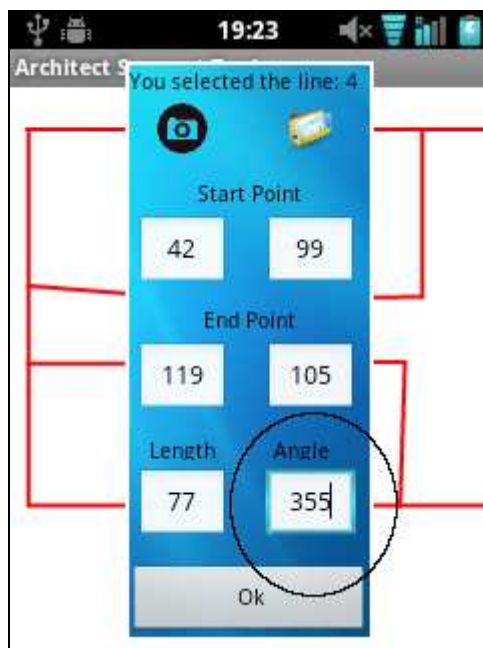


Image 3.51: Angle is 355

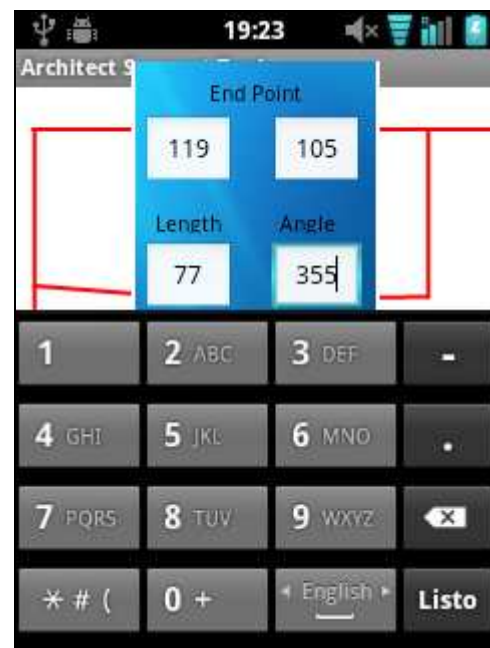


Image 3.52: Input new angle

Through the use of the Select option we can open the Entity Window with the related information of this wall, and we can see that the angle is 355° so in order to have it straight we have to input 0° .



Image 3.53: Angle at 0°

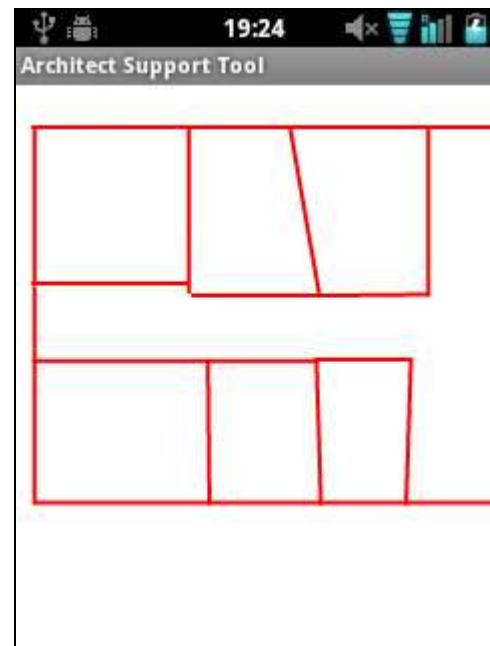


Image 3.54: New corrected wall

Ok, we manage to correct the wall to be oriented at 0° angle, but we are still not satisfied of the result. We realized that the wall was being painted from left to right, so what we need is to redraw the wall from right to left to effectively use the angle option.

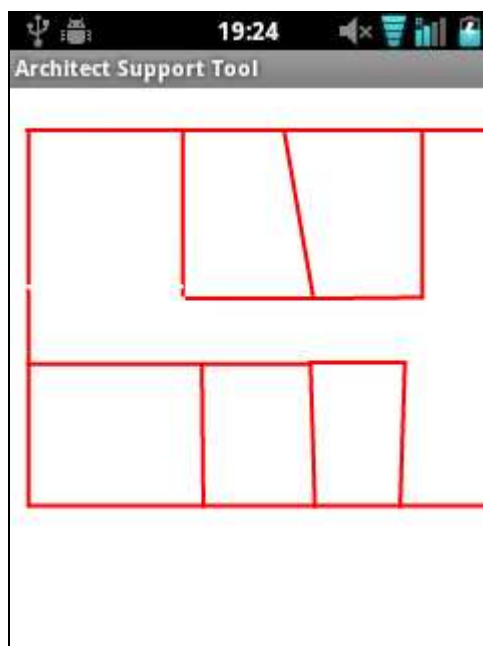


Image 3.55: Erasing wall

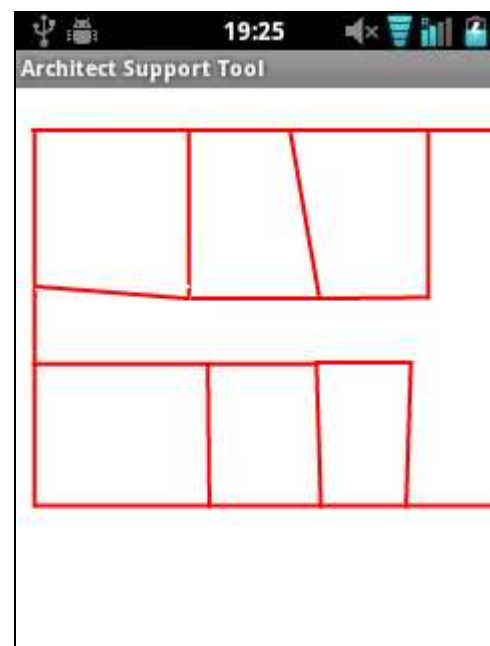


Image 3.56: Repainting wall

We have deleted the wall, and redraw it from right to left, and now we will change its angle. The explanation as to why this wall is being inclined it's because, even though we are on vector mode, and this shouldn't happen, the connect points core is always active and is detecting that there is some near point around there, preventing us to actually draw an straight line at 270° degrees. It's not a problem because we can fix it.



Image 3.57: Fixing angle

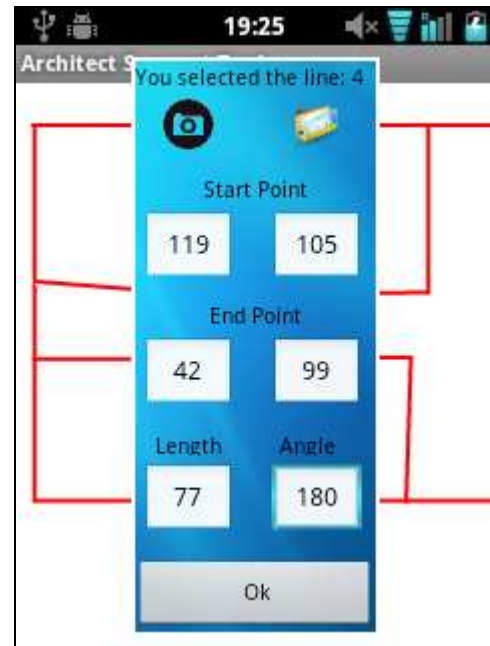


Image 3.58: New angle at 180°

We could see that it had 175° so we will change it for 180° and accept the changes, and so this is the result that we desired.

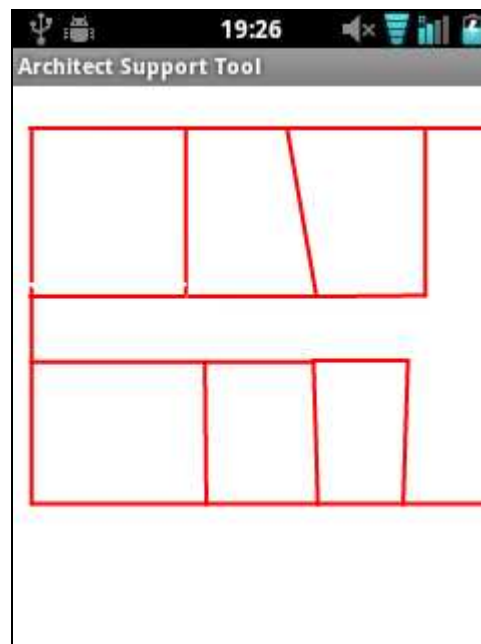


Image 3.59: New wall

Now let's say that our architect has realized that he actually didn't want to make one wall, but two walls, dividing that room in two rooms. There are several ways to accomplish this, but we are going to explain how to do it manually through the "Length" parameter of the entity window.

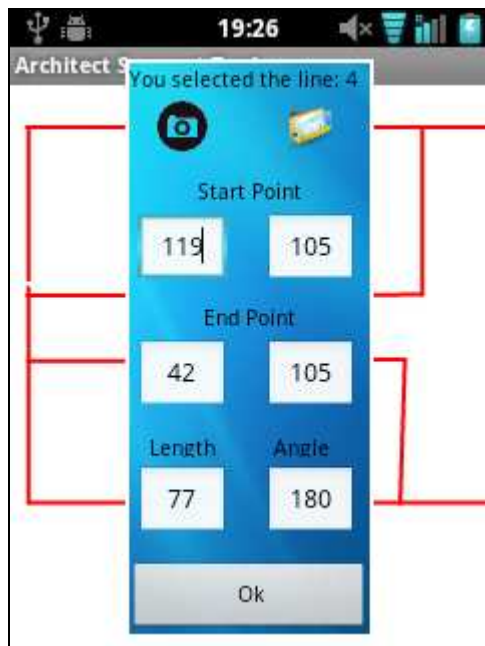


Image 3.60: Entity window

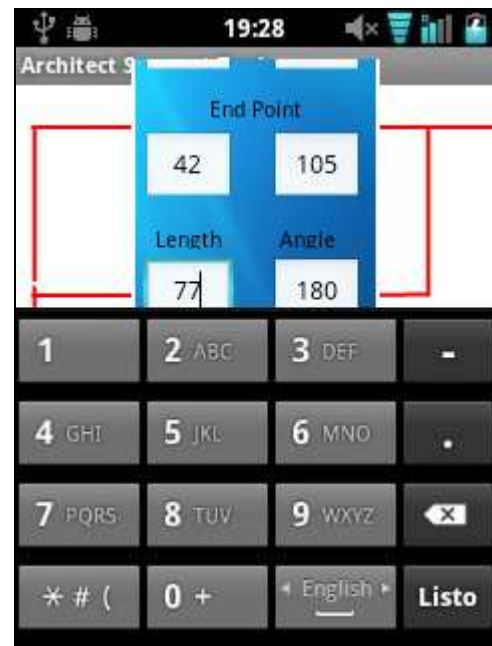


Image 3.61: Changing length

We see that the wall has 7.7 meters of length (77 decametres), in order to divide this room we have to shorten it into a half. So 3.5 meters will be good enough.

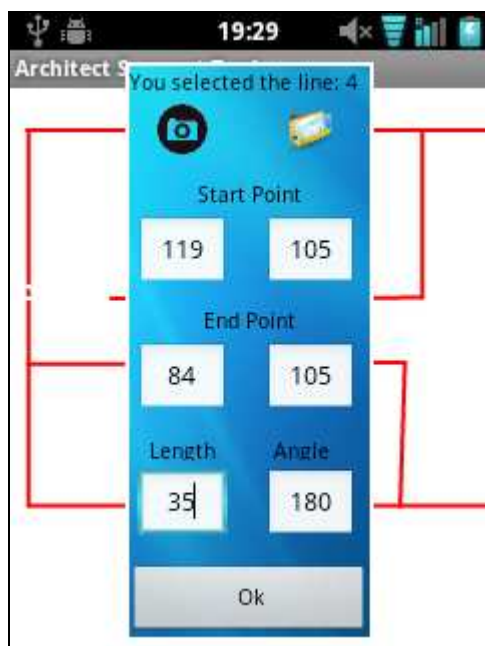


Image 3.62: Accepting changes

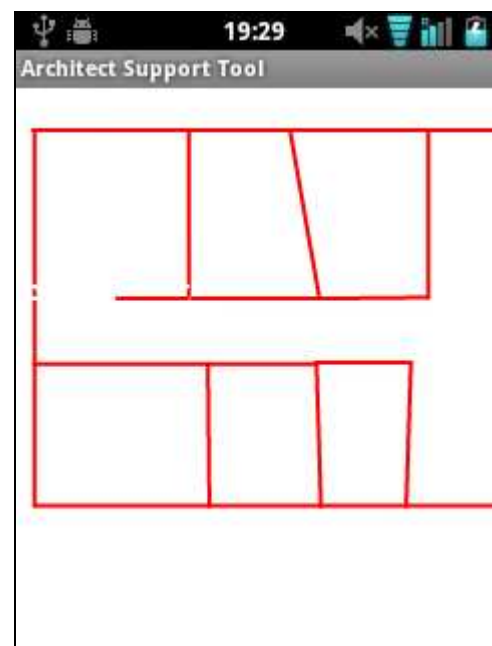


Image 3.63: Shortened wall to 3.5m

Ok, now we are going to draw another wall to prepare this first room, we will do it by finger, and we will not care so much about trespassing other walls because we will correct it through the Length parameter.

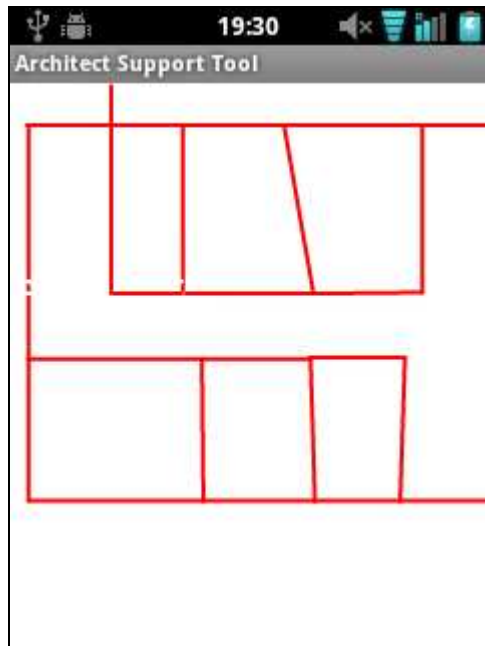


Image 3.64: New painted wall

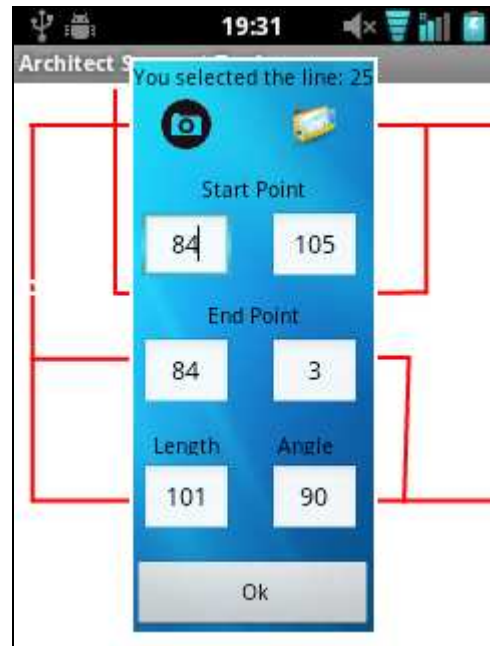


Image 3.65: Length of the wall (10.1m)

Either by trial and error or checking the length of the parallel wall or just because we already know, we correct the length size to 8.4 meters, and so we accept the changes.

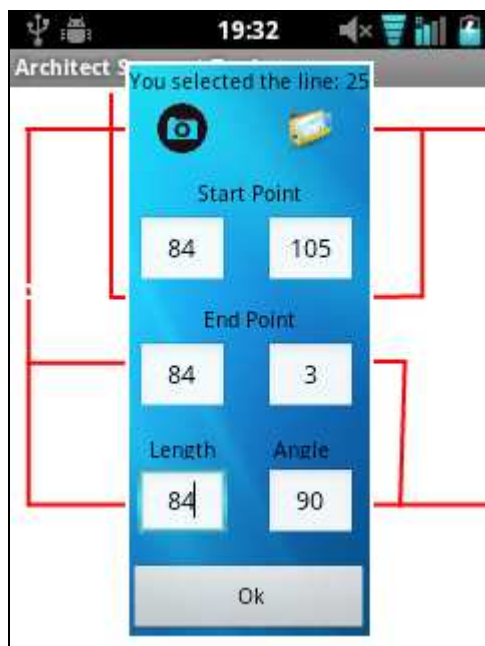


Image 3.66: New length size 8.4m

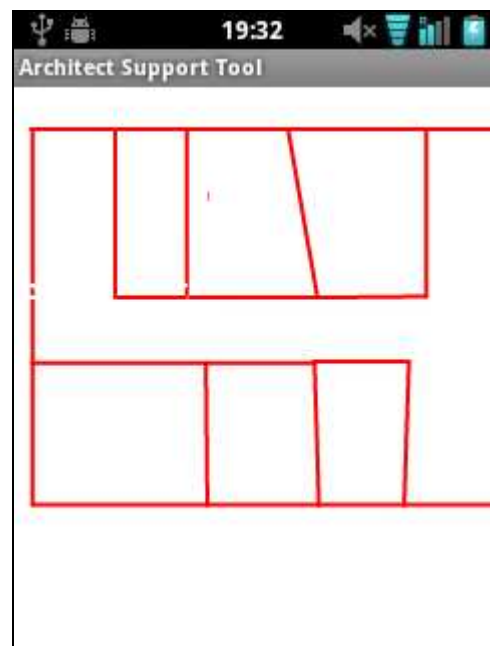


Image 3.67: New repainted wall

Now we can see how we manage to do a room inside the first one, let's go to finish this process, drawing the last wall.

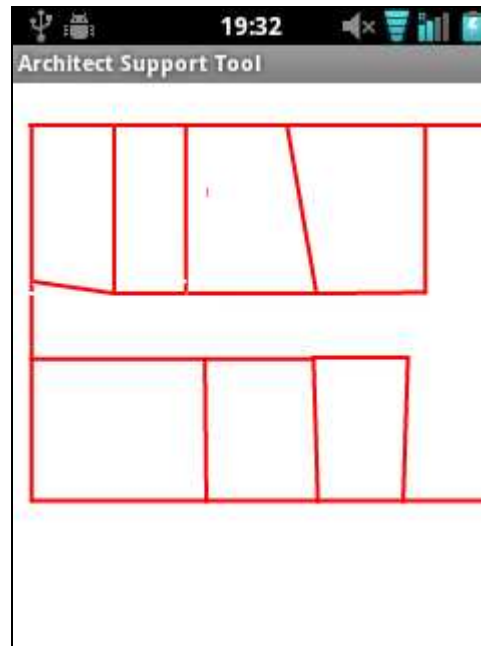


Image 3.68: New repainted wall

Again it happened the same thing as before, there is a point around there where the Connecting Points Core redraws our line into it. But, now we know how to fix that really quickly:



Image 3.69: Correcting angle to 180°

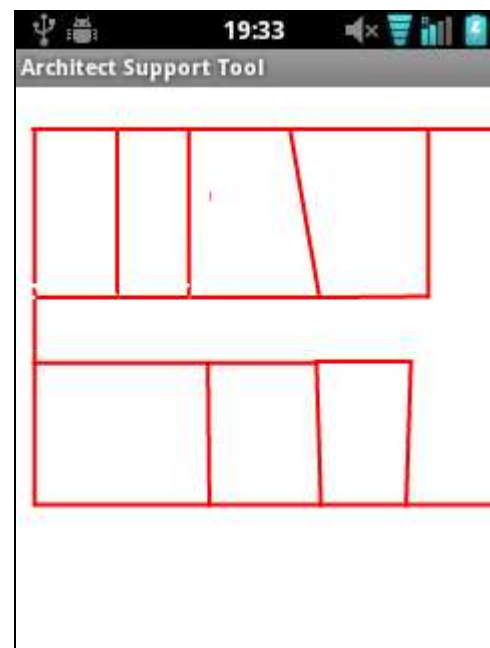


Image 3.70: New repainted wall

And finally we have our room divided in two smaller rooms. Following the same principles one can fix the entire map as much as he wants:

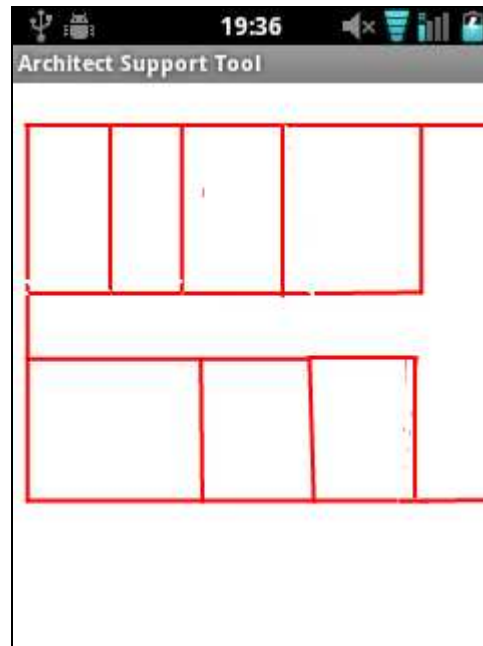


Image 3.71: Corrected map

Now as to following the same scenario, we need to put doors into this map, that guides us into the next section.

3.2.13 Door Input

The ability to put doors is important to see where the entrances of the respective rooms are, as this is a prototype there were no time to drag and drop a door icon, so we used the ability to "remove" partially smaller sections of the walls.

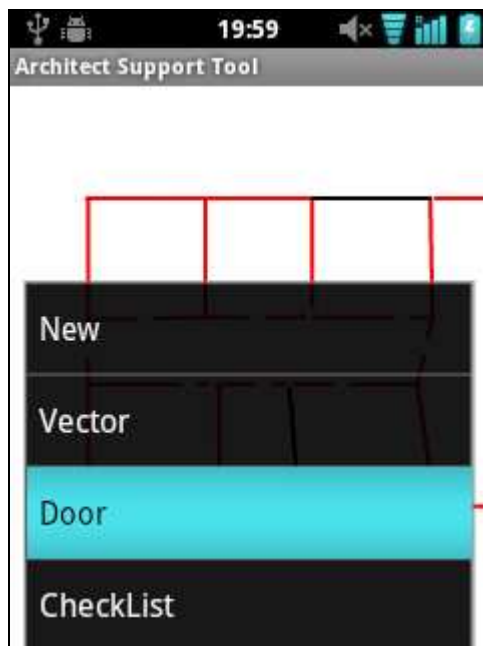


Image 3.72: Option menu "Door"

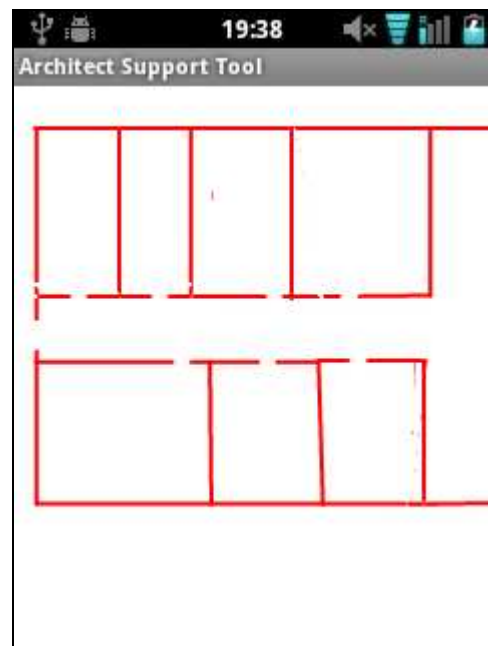


Image 3.73: Flat with doors

When the user checks the Door option menu, he enters into the Door Mode. In this mode he can draw "white" lines, effectively removing the small sections where he crosses

with the movement of his finger. These white lines does not generate any type of data, they only modify the bitmap, nothing else.

Using it in an smart way, one can give the illusion of drawing entrances into each room or he flat itself. As the picture 3.73 is shown.

The user is requested to touch again the Door option menu in order to leave out from this mode.

Sequence diagram involved:

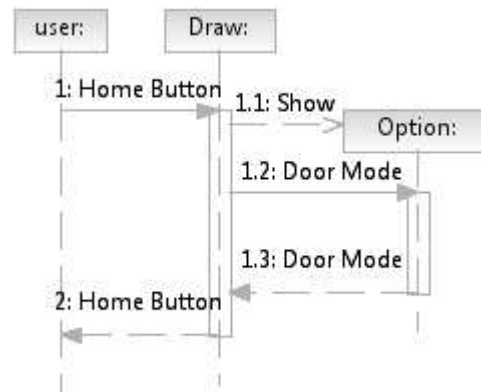


Image 3.74: SD of Vector mode

3.2.14 Checklist

This feature allows the user to check a list of elements that the flat should have. The user can check those elements for information purposes, or try to fill in the new information. It consists of a list of elements, where the user can check or uncheck what he feels is being complied with the quality standards or not.

As basic as this list is, for this prototype stage, they are probably the most important to check, the current elements to check is as follows:

- **Dimensions:** If checked means that the flat has a proportionate size and addeuqate for its purpose (it's not the same a flat for a living than an study flat).
- **Ventilation:** If checked means that the flat has enough exhaust ventilations.
- **Windows:** If checked means that the flat has all its windows in good condition, and are sufficient for the flat.
- **Smoke Pipe:** If checked the smoke pipes do not have any problem, and they do exist.
- **Gas Pipe:** If checked the gas pipes do not have any problem, and they do exist.
- **Lights:** If checked, the flat has enough lights to make it a comfortable place through all the room.
- **Plugs:** If checked, the flat has enough plugs connections through all the rooms.

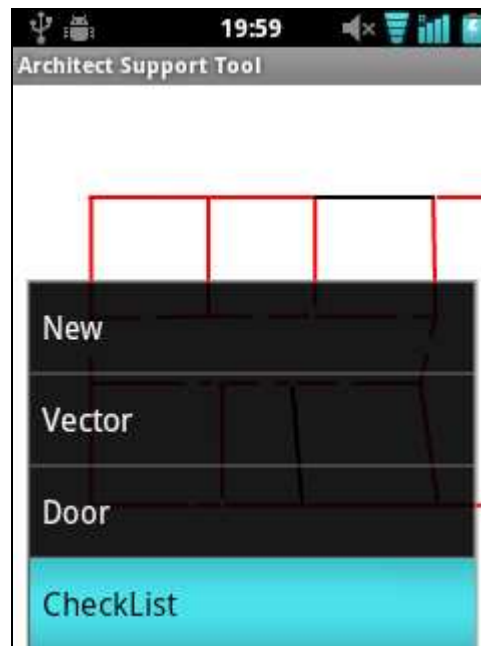


Image 3.75: Option menu "CheckList"

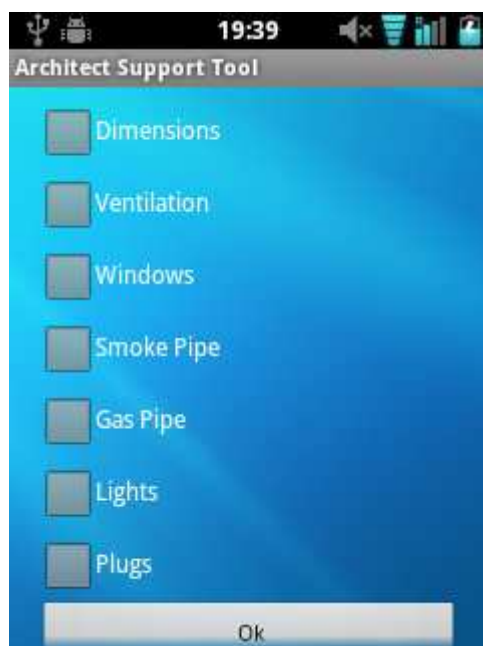


Image 3.76: Checklist screen

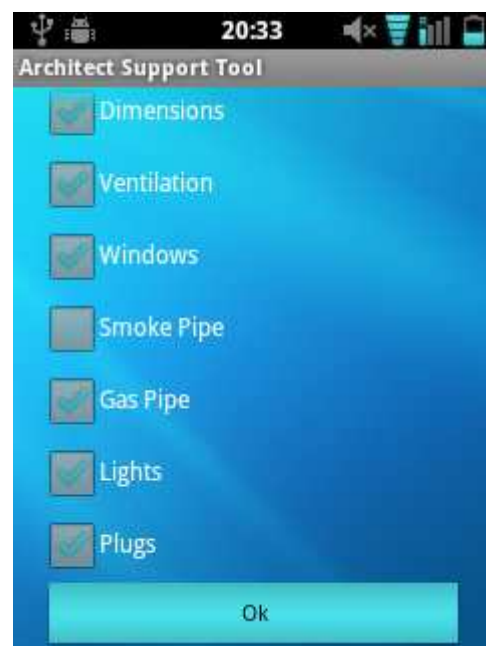
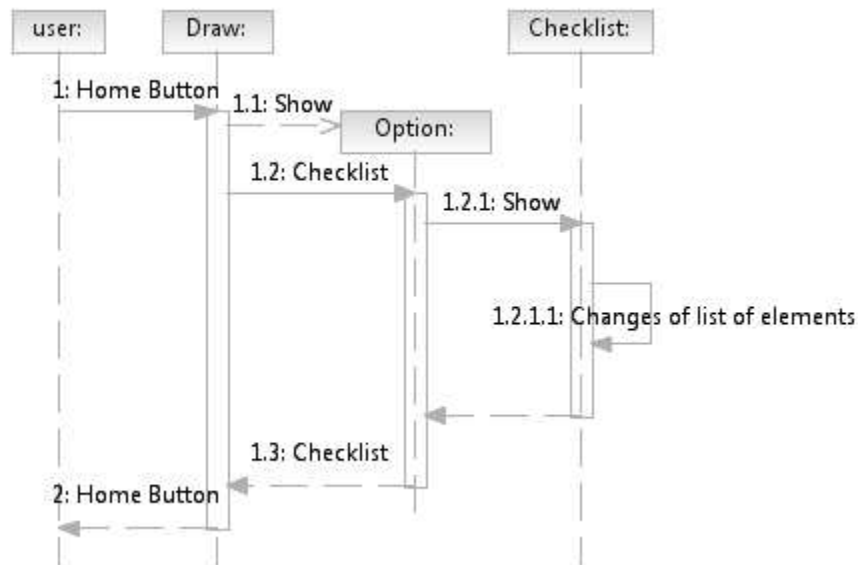


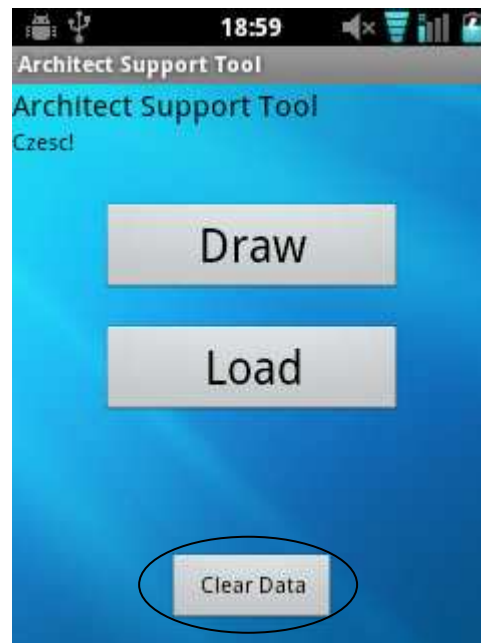
Image 3.77: Accepting changes

When the user selects the Checklist Option menu, the Checklist screen will appear with all the elements to be checked. If the user does not want to make any modification or just to read it, he will need to touch the back button, instead of the Ok button.

All maps has one checklist associated to them.

Sequence diagram involved:**Image 3.78: SD of Checklist screen****3.2.15 Clear data**

This button allows the user to delete all the information of the system, pictures, and databases (related to the application of course).

**Image 3.79: Clear Data button**

Unfortunately this is the only way to delete all the generated information in this prototype as we didn't have time to create individual delete operations for each map and pictures.

3.3 Design of the Data Base

The Data Base will be made using SQLite, in particular there will be 4 Tables in which one of them is stored a particular relevant data.

Before entering into explaining those tables, we need to remember when and what we need to store.

3.3.1 Save drawing scenario

When the user selects Save, from the application, he wants the current map to be saved.

The relevant information of a current map is:

- Current drawn **Bitmap**
- The Data array **mapCoordinates**, responsible to divide the map in sections and label them depending if on these sections there is an entity or not.
- The Data array **aEntities**, responsible to save all entities in the map.
- The **ID** of the current map
- The pictures captured from pathologies.

For the last point, as they are already saved into the SD card as per the Camera application works, we only need the name of the picture, so an string will be enough.

The first point the Bitmap, is stored in the SD using the ID of the current map for later locate it. The rest of this information is stored into an object class called "saveObject".

```
public class saveObject implements Serializable {

    private static final long serialVersionUID = 1L;

    public byte mapCoordinates[][];
    public ArrayList<DrawEntity> aEntities = new ArrayList<DrawEntity>();
    public int Id;
    public String bitmapFlnm;
}
```

Code 3-I: saveObject class

This saveObject is serialized and then inserted into the SQLite in the first "newtable".

```
private void save() {
    if (Id == 0) {
        List<byte[]> states = null;
        dh = new DataManipulator(this);
        states = dh.selectAll();
        dh.close();
        dh = null;
        Id = states.size() + 1;
    }

    saveObject so = new saveObject();
```

```

        so.aEntities = this.aEntities;
        so.mapCoordinates = this.mapCoordinates;
        so.Id = Id;
        so.bitmapFlnm = "HOUSE_ID_" + Id;

        // save bitmap to sd
        ByteArrayOutputStream bytes = new ByteArrayOutputStream();
        mBitmap.compress(Bitmap.CompressFormat.PNG, 100, bytes);

        try {
            File folder = new
File(Environment.getExternalStorageDirectory() + BASE + "HOUSE_" + so.Id +
File.separator);
            if (!folder.exists()) {
                folder.mkdir();
            }
            File f = new File(Environment.getExternalStorageDirectory() +
BASE + "HOUSE_" + so.Id + File.separator + so.bitmapFlnm + ".png");
            f.createNewFile();
            // write the bytes in file
            FileOutputStream fo = new FileOutputStream(f);
            fo.write(bytes.toByteArray());
            fo.close();

        } catch (Exception e) {
            e.printStackTrace();
        }

        byte[] object = SerializerClass.serializeObject(so);

        dh = new DataManipulator(this);
        long temp;
        temp = dh.update(so.Id, object);
        if (temp < 0)
            dh.insert(object);
        dh.close();
    }

```

Code 3-II: Saving a map

We are using the DataManipulator class, in order to communicate with the Database, the DataManipulator class has all the function calls, insert, update and select needed for the application.

3.3.2 Save pathology scenario

This is when the user inserts a new pathology (or modifies one). The relevant information to save is:

- ID of the current map
- ID of the entity selected
- String Description: Where the description of the pathology will be captured.
- String Damage: Where the elements affected will be captured.
- Pictures already taken

For the last parameter, we do not need nothing special because they are already saved in the SD card using the ID of the current map and the ID of the entity ID to identify them.

It's clear that in order to identify a Pathology we need the current ID map, and the ID of the selected entity, the combination of both fields will identify univocally the pathology and so save the description, damage and pictures of the pathology).

In this case, there is no serialized object, an SQL query of insert or update will be called, using the Data Manipulator class.

```
public long insertPathology(int houseid, byte entityid, String
description, String damage) {
    String INSERT_pathology = "insert into " + TABLE_NAME2 + "
(houseid,entityid,description,damage) values (" + houseid + "," + entityid +
",\"\" + description + "\",\"\" + damage + "\"")";
    this.insertStmt =
DataManipulator.db.compileStatement(INSERT_pathology);
    return this.insertStmt.executeInsert();
}
```

Code 3-III: Saving a pathology

This data is stored into the second table of the database, called "newtable2".

3.3.3 Save temporary scenario

When we were in the implementation phase, we encountered some problems as the when the application itself was being pushed out of the memory, or in the background in order to let pass another application in Android systems.

Everytime that happened (specially when launching the camera), when the application returned back in front, there were some data lost. In order to circumvent this issue we implemented a save temporal option, that is called when the application lose its focus, when that happens, all Android applications has a lifecycle and it's enough to know that the onDestroy callback it's used.

We override that callback in order to save a temporary file, so when it gets back, it will load the temporary file if exists.

Here is the simple code from Draw.java

```
@Override
public void onDestroy() {
    super.onDestroy();
    saveTmp();
}
```

Code 3-IV: Saving temporary map

The function saveTmp() is almost identical as save, just with minor tweaks, also it calls from the DataManipulator class another simple version of Insert (or update) call to the SQLite database.

This information is stored in the third table of the database called "newtable3".

Thanks to this implementation it allows the user to recover the last working map even if he exited from the program.

3.3.4 Save checklist scenario

All maps have a checklist, if this does not exist the checklist will appear with all its elements unchecked, but at the first time that the user saves any changes (as pushing the button Ok, on the checklist screen) then it will save the relevant information into the database.

The relevant information to store is just the boolean values of all the fields (if they are checked or not). For SQLite limitation which does not allow to store booleans, we just store integer values, being 0 false, and 1 true.

This is how, from the CheckList class, is called and passed the arguments to the DataManipulator class:

```
btnCheckOk.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        List<Integer> list = new ArrayList<Integer>();
        byte val1 = (byte)(dimensions.isChecked()? 1 : 0);
        byte val2 = (byte)(ventilation.isChecked()? 1 : 0);
        byte val3 = (byte)(windows.isChecked()? 1 : 0);
        byte val4 = (byte)(smokepipe.isChecked()? 1 : 0);
        byte val5 = (byte)(gaspipes.isChecked()? 1 : 0);
        byte val6 = (byte)(lights.isChecked()? 1 : 0);
        byte val7 = (byte)(plugs.isChecked()? 1 : 0);

        DataManipulator dh = new DataManipulator(getBaseContext());
        list = dh.selectCheckList(houseID);
        if (list.size() != 0) {
            dh.updateCheckList(houseID, val1, val2, val3, val4, val5,
val6, val7);
        } else {
            dh.insertCheckList(houseID, val1, val2, val3, val4, val5,
val6, val7);
        }
        dh.close();
        finish();
    }
});
```

Code 3-V: Saving checklist of current map

And this is the InsertCheckList of the DataManipulator class, quite similar as the insert from the case of save of draw.

```
public long insertCheckList(int houseid, byte dimensions, byte ventilation, byte
windows, byte smokepipe, byte gaspipe, byte lights, byte plugs) {
    String sql = "insert into " + TABLE_NAME4 + "
(houseid,dimensions,ventilation,windows,smokepipe,gaspipe,lights,plugs) values
(" + houseid + ",\"" + dimensions + "\",\"" + ventilation + "\",\"" + windows +
 "\",\"" + smokepipe + "\",\"" + gaspipe + "\",\"" + lights + "\",\"" + plugs +
```

```

"\");
    this.insertStmt = DataManipulator.db.compileStatement(sql);
    return this.insertStmt.executeInsert();
}

```

Code 3-V: InsertCheckList from DataManipulator class

The checklist data is stored into the fourth and last table of the database called "newtable4"

3.3.5 Creation and purpose of each table

So finally, we have 4 tables in our SQLite database:

- newtable: Responsible to store the data structure of a map, it stores serialized objects.
- newtable2: Responsible to store the pathologies, it stores ID of the map, ID of the entity, description and damage.
- newtable3: Responsible to store the save temporal file, (just one entry)
- newtable4: Responsible to store the checklist information (1 ID field for current map, and the other 7 extra fields representing the elements to be checked).

We are sorry for having these names, as we didn't have time to change them and test it properly, so we prefer not to touch them. Obviously in future stages they would be changed into something more describing of what is their purpose.

This is the code used to create these tables in the DataManipulator class

```

public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + TABLE_NAME + " (id INTEGER PRIMARY KEY, data BLOB)");
    db.execSQL("CREATE TABLE " + TABLE_NAME2 + " (id INTEGER PRIMARY KEY, houseid INTEGER, entityID TINYINT UNSIGNED, description VARCHAR, damage VARCHAR)");
    db.execSQL("CREATE TABLE " + TABLE_NAME3 + " (id INTEGER PRIMARY KEY, data BLOB)");
    db.execSQL("CREATE TABLE " + TABLE_NAME4 + " (id INTEGER PRIMARY KEY, houseid INTEGER, dimensions TINYINT UNSIGNED, ventilation TINYINT UNSIGNED, windows TINYINT UNSIGNED, smokepipe TINYINT UNSIGNED, gaspipe TINYINT UNSIGNED, lights TINYINT UNSIGNED, plugs TINYINT UNSIGNED)");
}

```

Code 3-VI: Table structures of our SQLite database

4 Testing and performance

Unfortunately we didn't have time to test this application extensively outside of its development process, but we can say that at least we improved from the ever first version.

The early version, as we commented at some point before had 2 big problems regarding usage of memory and performance.

First problem - Saving Data structures into SQL

The early version used a Connecting Points Core who consumed lots of resources, in order to detect if the finger of the user when it touches the screen were near of another point, the first implemented solution was to have 2 arrays of points, the size of each array was equal to all pixels that a screen could have. In our particular case 240*320 pixels (as per our android smartphone X10 Mini). Each position of this array represented a pixel of the screen.

When an user touched the screen, we saved into the position of the array the new id entity that was being created, and also the next 40*40 pixels near to it (20 pixels of radius tolerances was equal as an square of 40*40 pixels being the touched one its centre), so when later the user were about to touch the screen around that particular position, we just had to check into the array if it was filled with an entity id, if it was we knew that it had a near point to be connected.

And we were using 2 arrays, one for integer positions, and the other for float positions. Obviously when we begin to implement the save function and the database, this proved to be too much memory consuming, actually was calculated that by an average flat size, it was required 0.5 Mega Bytes. With just 2 flats we were exceeding the virtual memory of java, and also it was slow.

Solution:

The solution was simple and we were surprised to not think about it before. We just do not create any structure at all, just when an entity is created we save its starting point and ending point, in the same entity object, and that's all, and the most interesting thing is that we were already doing this, so no big modifications were needed to do.

Then, when the user touches the screen we call a function to search from the array structure containing all entities, in order to find a point that it is inside the 20 pixels radius. As the array of entities usually will contains few elements (from the order of the 10th or 20s as much), the search is instantly, and we manage to resolve all the memory problems.

With this and other tweaks we passed from using 0.5MB as average for each flat into something no more bigger than 10KB, the amount of flats that the system can store, at least into its SQL database has been increased by fifty.

Second problem - Saving photos for pathologies

At the beginning the default option for a camera when it takes a photo is to take it at full resolution as much the camera of the smartphone can handle. This was a problem,

because when we tried to load these photos internally inside the application, after only two pictures we were running out of virtual memory.

Solution:

There is an option when calling the internal camera application to take a low resolution picture and store it into a bitmap, that saved all the virtual memory.

5 Planning

Finally the original planning has suffered some changes of redistribution and dedicated tasks.

TASK	START	END	HOURS SPENT
Investigation and Documentation	1/03/12	30/04/12	140
Determining architect requirements	1/03/12	12/03/12	
Investigating architect terminology	13/03/12	14/03/12	
Preliminary analysis	15/03/12	19/03/12	
Learning Android environment	20/03/12	22/04/12	
Preparing framework	23/04/12	30/04/12	
Specification and Design	1/05/12	31/05/12	110
Analysis requirement	1/05/12	7/05/12	
Designing components	8/05/12	15/05/12	
Draw module strategy	16/05/12	19/05/12	
Interface module strategy	20/05/12	22/05/12	
Task planning	23/05/12	30/05/12	
Implementation	10/06/12	28/08/12	340
Main screens and basic navigation	1/06/12	13/06/12	
Draw Core	14/06/12	24/06/12	
Connect points Core	21/06/12	24/06/12	
Menu Option Interface	25/06/12	26/06/12	
Menu Option Select	27/06/12	11/07/12	
Popup Screen	27/06/12	29/06/12	
Length module	30/06/12	7/07/12	
Angle module	8/07/12	11/07/12	
Menu Option Erase	12/07/12	14/07/12	
Menu Option Scroll	15/07/12	19/07/12	
Menu Option Save	20/07/12	24/07/12	
Menu Option Load	6/08/12	10/08/12	
Menu Option New	11/08/12	12/08/12	
Improvement Connect points Core	13/08/12	14/08/12	
Camera Interaction	15/08/12	20/08/12	
Gallery Module	21/08/12	24/08/12	
Improvement Draw Core	25/08/12	27/08/12	
Vector Mode	26/08/12	27/08/12	
Menu Option Door	28/08/12	29/08/12	
Pathology Module	30/08/12	5/09/12	
Checklist Module	6/09/12	10/09/12	
End of Implementation	10/06/12	10/09/12	
Testing	11/09/12	16/09/12	20
Fixing code	11/09/12	16/09/12	
Preliminary documentation	1/03/12	24/09/12	
Final Thesis Memory	24/03/12	28/09/12	67
Total hours:			677

About the first 2 phases, there were no changes because at the moment of doing the original plan we were at 1/6/12, but the implementation have been much longer than expected. Originally the plan was not to use August month, but was there in order I was being late. And that was the case, all the implementation process has taken much longer than what I thought, and it was because I am new on Android environment, as every task that I needed I have to document myself in order to know how to do it, and that was an investing time that's not directly represented. It has to be noted that the period from 25/7/12 to 5/8/12 I was at my home country, on holidays, so I was not working on the project on those days.

Second, sometimes I need feedback from my architect colleague and I couldn't locate him fast, and after locating him I had to revise and modify some functionality. (That's why is a project based on the prototype lifecycle).

But I wanted to maintain as far as I could the original tasks without dropping any one, so in the end the process that suffered from it was the Testing phase, reduced just 20 hours and probably inside these hours 15 are from fixing code, so it has been only 5 hours for testing.

I preferred to have a prototype with much functions possible be done, and working reasonable than to have a solid proof stable prototype but less functionalities. If this works and it is good, the stabilization of this prototype will be the next step.

Also in the beginning at the Investigation and documentation, and also at the specification and design, I was setting up myself here at Wroclaw with this new experience being erasmus student, so I need some time in order to properly focus on thesis.

So far this summer has been great in order to advance a lot of work on this project.

6 Budget

As all projects there have to be a section speaking about the budget or overall cost that would represent this application if it was about to put on commercial status. In this particular case, perhaps in the future may be a commercial application, but obviously not as it is like now, it needs more improvements yet to be on commercial level, but anyway for the moment is not so difficult to compute the cost.

First there are no proprietary licenses in this code; the advantage for programming for android is that there are no additional costs on licenses.

Second, the smartphone cost it cannot be counted as the cost of the application, because that's outside of our control.

Third, the only costs here is the number of hours developing on this application, which as the present case it have been only one person (me), and one can just multiply the total number of hours by an average price per hour of programming.

In Spain would be like $677 * 8 = 5.416\text{€}$ here at Wroclaw would be less, I am not educated enough to know the average income for an IT specialist but I suppose around $677 * 19 = 12.863\text{PLN}$. If our most directly competitor charges like 80€ for a similar application, if we charge let's say just 40€, after 136 downloads we will recover the costs.

But to think in those terms is unreal, as stated, the application as it is now cannot be sold (even is not usable enough for a free release, unfortunately), so this case of study should be done in the future when the prototype would cease to be a mere prototype.

But as this is only personal costs, I sincerely do not treat that as important as long the application is popular and successful, in the future. Sincerely, I am just thinking to make it as free as possible.

7 Possible improvements and upgrades

This is the first usable prototype to see how helpful an application like this can be for architects, but obviously it needs to be improved. These are all the elements that need improvement:

- The Length measure of the entity window should be changed in order to reflect Meters and not Decametres, this should be fairly easy to accomplish.
- Delete option for Pathologies, or saved maps. This should be easy to accomplish too.
- Every time that the user wants to save the application should ask if he wants to rewrite an over existing save or create a new one (and ask also the name of this save state). This may need a bit of work but shouldn't be any problems in doing so.
- Zoom option, for zooming out or in the entire bitmap. This is difficult to implement because it touches a lots of areas, but should be done too.
- Icon doors, instead of the actual system of deleting pieces of a line, it should be like dragging and dropping a door icon into the wall. This is difficult because it needs logic in order to intersect the wall and put the corrected orientation of the door.
- Ability to draw circles or ellipses, for columns. A bit difficult it needs some thinking process, but at least the entity class is already created to store it.
- Upgrade the checklist window with much more detail, perhaps even flexible for the architect to desire. In the future there should be like an overall checklist of the flat and then another independent checklist for each room. This is difficult to do, it needs a designing process.
- A third option on the menu screen to read some local laws as what a property should accomplish. Not difficult but tedious, it needs investigation of the current laws of the selected country.
- Improve the drawing method, for example, the connecting points core should be also to detect when a line is crossing another line and so cut it and connect them, challenging but reasonable to implement with some time. There may be a major problem; depending on the solution the performance could be severely affected, as to try to guess all the points of the current lines.
- An Undo option, to undo whatever have been drawn previously. (it will improve the usability of the application).
- Export option to Cad, this would be really useful for architects but it is also the most difficult to do in my opinion. One needs to know how Cad works internally and as far as I know it's a proprietary system... so a lot of investigation should be required to make it compatible.
- Improve the databases and stability of the system.

As one can see, there are a lot of room for improvement, some of this can be difficult but almost all of them can be achieved with reasonable time, improving the application much more.

8 Conclusions

This project is quite ambitious, from the beginning it's clear that there is an empty space in the market to cover this necessity from architects, but to develop this application has been really challenging for me, as it was my first time developing for an smartphone, in this case for Android systems, and also living outside my country.

That caused serious problems during the implementation phase, as I didn't have enough knowledge (and still I don't think that I am good enough on it) but so far the knowledge that I have learnt regarding that is invaluable and indisputable. A kind of knowledge that will be useful for my future, and probably for the future of the informatics, as the trending is to have more embedded systems in small devices.

Being said that, this application is not trying to substitute any previous routine that architects has, for lack of time it couldn't be finished as I think it should be in order be completely usable. But, as a prototype and to test it it's a good beginning. After this I will see how much I can improve this solution, and I am eager to show it to my friend in Spain Jaume Casadevall Puig, the actual architect who gave me the idea and supported me giving me all the required information from an architect point of view.

As for my personal learning and enjoyment, I already loved to do this thesis in such a beautiful country as Poland, and especially in such a multicultural city like Wrocław. People were surprised when I asked to stay here for more time in order to work, they were thinking that I could do that back at my home town, but, it wouldn't be the same. I am really grateful to my home university, Universitat Politècnica de Catalunya and my hosting university, Politechnika Wrocławska to allow me to stay here more time in order to finish this thesis and being so comprehensible to me.

Finally thanks to my tutor Krzysztof Waśko, who even always busy and having to cope with Erasmus people that sometimes do not even know how to speak proper English (I hope I am not the case!), never let me out and supported me.

9 Annex A: Images index

IMAGE 2.1: USE CASE DIAGRAM OF THE DRAW CORE	- 18 -
IMAGE 2.2: USE CASE DIAGRAM OF THE PATHOLOGY CORE	- 20 -
IMAGE 2.3: USE CASE DIAGRAM OF THE CHECKLIST CORE	- 21 -
IMAGE 2.4: USE CASE DIAGRAM OF THE PERSISTENT CORE	- 23 -
IMAGE 2.5: 3-LAYER ARCHITECTONIC PATTERN	- 24 -
IMAGE 2.6: UML CONCEPTUAL DATA MODEL OF LAYER PRESENTATION	- 25 -
IMAGE 2.7: UML CONCEPTUAL DATA MODEL OF DRAW CORE DOMAIN LAYER	- 26 -
IMAGE 2.8: UML CONCEPTUAL DATA MODEL OF PATHOLOGY CORE DOMAIN LAYER	- 27 -
IMAGE 2.9: UML CONCEPTUAL DATA MODEL OF CHECKLIST CORE DOMAIN LAYER	- 27 -
IMAGE 2.10: UML CONCEPTUAL DATA MODEL OF PERSISTENCE LAYER	- 28 -
IMAGE 3.1: MENU SCREEN OF AST	- 29 -
IMAGE 3.2: SD OF DRAW BUTTON	- 30 -
IMAGE 3.3: SD OF DRAW BUTTON	- 30 -
IMAGE 3.4: DRAWING SCREEN	- 30 -
IMAGE 3.5: OPTION MENU	- 30 -
IMAGE 3.6: OPTION MENU "MORE"	- 31 -
IMAGE 3.7: OPTION EXTENDED MENU	- 31 -
IMAGE 3.8: SD OF OPTION MENU	- 31 -
IMAGE 3.9: PRACTICAL SELECT EXAMPLE	- 32 -
IMAGE 3.10: PRACTICAL SELECT EXAMPLE	- 32 -
IMAGE 3.11: POPUP ENTITY WINDOW	- 32 -
IMAGE 3.12: BLACK WALL AFTER INSERTING PATHOLOGY	- 33 -
IMAGE 3.13: SD OF OPTION MENU	- 34 -
IMAGE 3.14: GALLERY ICON IN ENTITY WINDOW	- 34 -
IMAGE 3.15: GALLERY WINDOW	- 35 -
IMAGE 3.16: NAVIGATING BETWEEN PHOTOS	- 35 -
IMAGE 3.17: NAVIGATING BETWEEN PHOTOS	- 35 -
IMAGE 3.18: GALLERY TEXT FIELDS	- 36 -
IMAGE 3.19: GALLERY INPUT METHOD	- 36 -
IMAGE 3.20: GALLERY FILLING INFO	- 36 -
IMAGE 3.21: GALLERY ACCEPTING CHANGES	- 36 -
IMAGE 3.22: SD OF GALLERY WINDOW	- 37 -
IMAGE 3.23: OPTION MENU "SCROLL"	- 38 -
IMAGE 3.24: SCROLLING MAP	- 38 -
IMAGE 3.25: SD OF SCROLLING	- 38 -
IMAGE 3.26: OPTION MENU "SAVE"	- 39 -
IMAGE 3.27: SD OF SAVING	- 40 -
IMAGE 3.28: LOAD BUTTON MENU SCREEN	- 40 -
IMAGE 3.29: LOAD BUTTON OPTION MENU	- 40 -
IMAGE 3.30: LOAD SCREEN	- 41 -
IMAGE 3.31: SELECTING FROM LOAD SCREEN	- 41 -
IMAGE 3.32: AFTER LOADING THE SELECTED MAP	- 41 -
IMAGE 3.33: SD OF LOADING FROM MENU SCREEN	- 42 -
IMAGE 3.34: SD OF LOADING FROM DRAW SCREEN	- 42 -
IMAGE 3.35: OPTION MENU "New"	- 43 -
IMAGE 3.36: NEW BLANK MAP	- 43 -
IMAGE 3.37: SD OF LOADING FROM DRAW SCREEN	- 43 -
IMAGE 3.38: MISTAKENLY PLACED WALL	- 44 -
IMAGE 3.39: OPTION MENU "ERASE"	- 44 -
IMAGE 3.40: WALL BEING REMOVED OUT	- 44 -
IMAGE 3.41: SD OF ERASE A WALL	- 45 -
IMAGE 3.42: AREA OF TOLERANCE TO CONNECT POINTS	- 45 -
IMAGE 3.43: LINE AFTER CONNECTING POINT	- 45 -
IMAGE 3.44: ACTIVATING VECTOR MODE	- 46 -
IMAGE 3.45: WALL AT 270°	- 46 -
IMAGE 3.46: WALL AT 0°	- 46 -

IMAGE 3.47: WALL AT 90°	- 47 -
IMAGE 3.48: WALL AT 180°	- 47 -
IMAGE 3.49: SD OF VECTOR MODE	- 47 -
IMAGE 3.50: WRONG DRAWN WALL.....	- 48 -
IMAGE 3.51: ANGLE IS 355	- 48 -
IMAGE 3.52: INPUT NEW ANGLE.....	- 48 -
IMAGE 3.53: ANGLE AT 0°	- 49 -
IMAGE 3.54: NEW CORRECTED WALL.....	- 49 -
IMAGE 3.55: ERASING WALL	- 49 -
IMAGE 3.56: REPAINTING WALL.....	- 49 -
IMAGE 3.57: FIXING ANGLE	- 50 -
IMAGE 3.58: NEW ANGLE AT 180°	- 50 -
IMAGE 3.59: NEW WALL	- 50 -
IMAGE 3.60: ENTITY WINDOW	- 51 -
IMAGE 3.61: CHANGING LENGTH	- 51 -
IMAGE 3.62: ACCEPTING CHANGES	- 51 -
IMAGE 3.63: SHORTENED WALL TO 3.5M	- 51 -
IMAGE 3.64: NEW PAINTED WALL	- 52 -
IMAGE 3.65: LENGTH OF THE WALL (10.1M).....	- 52 -
IMAGE 3.66: NEW LENGTH SIZE 8.4M.....	- 52 -
IMAGE 3.67: NEW REPAINTED WALL	- 52 -
IMAGE 3.68: NEW REPAINTED WALL	- 53 -
IMAGE 3.69: CORRECTING ANGLE TO 180°	- 53 -
IMAGE 3.70: NEW REPAINTED WALL	- 53 -
IMAGE 3.71: CORRECTED MAP	- 54 -
IMAGE 3.72: OPTION MENU "DOOR"	- 54 -
IMAGE 3.73: FLAT WITH DOORS	- 54 -
IMAGE 3.74: SD OF VECTOR MODE	- 55 -
IMAGE 3.75: OPTION MENU "CHECKLIST"	- 56 -
IMAGE 3.76: CHECKLIST SCREEN	- 56 -
IMAGE 3.77: ACCEPTING CHANGES	- 56 -
IMAGE 3.78: SD OF CHECKLIST SCREEN	- 57 -
IMAGE 3.79: CLEAR DATA BUTTON	- 57 -

10 Annex B: Code index

CODE 3-I: SAVEOBJECT CLASS.....	- 58 -
CODE 3-II: SAVING A MAP	- 59 -
CODE 3-III: SAVING A PATHOLOGY.....	- 60 -
CODE 3-IV: SAVING TEMPORARY MAP.....	- 60 -
CODE 3-V: SAVING CHECKLIST OF CURRENT MAP.....	- 61 -
CODE 3-V: INSERTCHECKLIST FROM DATAManipulator CLASS	- 62 -
CODE 3-VI: TABLE STRUCTURES OF OUR SQLITE DATABASE	- 62 -

11 Bibliography and references

Official Android developer page:

<http://developer.android.com/index.html>

Stack Overflow:

<http://stackoverflow.com/>

Android Forums:

<http://androidforums.com/>

Android & Eclipse tutorials on youtube:

<http://www.youtube.com/>

AutoCad - Wikipedia Foundations

<http://en.wikipedia.org/wiki/AutoCAD>

OrthoGraph for iPad

<http://www.orthograph.net/orthograph-architect-ipad.html>

2012

Architect Support Tool Resume

Alexandre Uroz Làzaro

Title: Development of a software tool to support architects

Volume: 1/1

Student: Alexandre Uroz Làzaro

Tutor: Dr. Krzysztof Waśko

Department: Systemy multimedialne i mobilne

Date: 24 of September 2012

1	PREPARATION.....	- 1 -
1.1	PROJECT GOALS.....	- 2 -
1.2	ENVIRONMENT DESCRIPTION.....	- 3 -
2	PROGRAMMING.....	- 4 -
2.1	MAIN AND MENU CLASS.....	- 4 -
2.2	DRAW CLASS.....	- 5 -
2.3	DRAWENTITY CLASS.....	- 8 -
2.4	SAVEOBJECT CLASS.....	- 9 -
2.5	DATAMANIPULATOR CLASS.....	- 9 -
2.6	INSERTING PATHOLOGIES	- 10 -
3	DESCRIPTION	- 13 -
3.1	DRAWING SCREEN	- 13 -
3.2	ENTITY SCREEN.....	- 14 -
3.3	GALLERY SCREEN.....	- 15 -
3.4	SCROLLING	- 16 -
3.5	SAVING	- 16 -
3.6	LOADING.....	- 16 -
3.7	CREATE A NEW MAP.....	- 17 -
3.8	ERASE A WALL.....	- 17 -
3.9	CONNECTING POINTS CORE.....	- 18 -
3.10	VECTOR MODE	- 18 -
3.11	DOOR INPUT.....	- 19 -
3.12	CHECKLIST.....	- 19 -
3.13	CLEAR DATA.....	- 20 -

1 Preparation

The idea of doing this project began back at 2010, in Spain, I have a friend who is an architect and in a casually chat we were having he commented me of how, at the moment, there was no significant application to help architects like him when they were working in the streets, analyzing, reviewing and studying the buildings.

In this particular case, his job consisted of visiting flats and buildings in order to find pathologies, circumstances that appear on buildings that diminish their security, value and/or comfort. They need all this information for several purposes, sometimes because the client requested a check of the security of the building and if it is needed to do some fixings, other times they need it in order to issue the certificate of occupancy, other times it might be just simple as to document the state of a building during its lifetime.

Usually all this information is gathered by hand making notes and taking photos so as later introduce them in a computer using their main popular architect program like AutoCad. He told me it would be really useful to have some sort of application that could be launched through a mobile device, like a smartphone or a tablet pc and could assist them in this work.

I thought it was an interesting concept to further develop so when finally I got my change to do it I proposed it here in Wrocław. And I am happy that my tutor found it interesting too.

It is no secret that an application like this, though small, merges several areas of informatics, like graphical design, graphical interface, user interaction and databases. And finally it is based from a real necessity which encourages doing its development.

As up today there are no significant applications on the market I decided to try to do it by myself and this is the result.

After some deliberation, I intended for this application to be for Android systems, the reason is quite simple, first I really like the Open Source environment that surrounds Android Operative System, and second, I had an Android smartphone, and could not afford to buy another smartphone, being said that I am not closing the possibility to develop it for other devices and OS (Operative System's) in the future if it is needed. But for the moment and for starters like me in this Android world, this would be good enough.

First thing to notice in the world of architects is that there are several roles in their work, for example:

- *Valuer*: One who is responsible to evaluate properties and give an approximate price.
- *Issuer of occupancy*: One who is responsible to evaluate properties and issue the certificate of occupancy.
- *Inspector*: One who is responsible to evaluate the legal conditions of the property.
- *Interior designer*: One who tries to maximize or rearrange or decorate the available space inside of a property to meet the expectations of the customer.

All of them, in order to do their job needs the representation of the property into a 2D representation (sometimes in 3D), usually called a map. The most basic thing is to be able to have a map of the property from a point of view from above to bottom.

They usually achieve this taking pictures, drawing in papers, and sometimes using notebooks (or netbooks) with AutoCad (the main architect's software).

But the process is quite "manual", and later they need to put all this information together in order to use it.

So, architects are forced to carry notebooks or netbooks if they want to fully take advantage of the informatics in their job at the "streets", which lots of them do not bring them. In some countries to have a notebook may be a normal thing, but for another countries isn't.

AutoCad is a heavy application that usually needs a quite powerful computer, depending on the severity of the property that one wants to represent. Usually architects have desktop computers which are powerful to do that job. There is also the explanation that not everywhere in the world they can afford for a notebook or netbook, and finally, when they need to work almost all day in the streets usually the battery autonomy of the notebooks restricts them.

So in the end, they usually take pictures and use pencil/pen on paper.

But that can be improved if they would be able to gather all this information in one place. This tool could support them in order to check further information readily available from Databases.

And so, here is the scope, and the main goal of this project. For the moment, this tool is not pretending to substitute any previous methodology; it is still far from it. But as a first prototype that can be fully expanded it's a good beginning.

1.1 Project goals

Good, we have a necessity to fulfil and motivation to develop but now the first problem I encountered was that I did not have previous knowledge of Android, so taking into consideration that, here are the goals of this project.

- 1) Learn Android language and be proficient developing for it
- 2) Develop a useful tool for architects that allow them to at least represent maps and introduce information.
- 3) Start a prototype which may be expanded in the future

In order to fulfil these goals the minimum application's requirements in order to be useful should be as follows:

1. Ability to represent maps
2. Ability to annex photos into sections of these maps
3. Ability to save and load these maps
4. Ability to write descriptions of the pathologies encountered
5. Ability to check a list of requirements of these maps
6. Ability to use this application on smartphone or tablet pc through touch screen

These have been the goals that I manage to accomplish to a certain degree, but this project has so much potential that it can be improved much further.

1.2 Environment description

The environment in which this project has been developed is as follows:

- Eclipse
- Sony Ericsson Xperia X10 Mini with Android version 2.3
- Android 2.1 Framework

First, there is no particular reason to choose Eclipse over another development software, just that it is perhaps one of the most used for programming Android, so that's why we choose it too in order to follow the general trend. So if sometimes one have to look for information on Internet, it will be easier to find as being more popular. The time in this project has been a critical factor so there was no time to lose trying to figure out which software would be the best but just to be practical.

Second, the reasons for using the smartphone Sony Ericsson Xperia X10 Mini are two, first, it was the system I usually have at my disposal to test everything, and second, this is one of the smallest smartphone on the market, so if my application can work in this small smartphone its guaranteed it will work much better on a bigger resolution screen of a bigger smartphone. Just to be sure we used emulators for testing this application on other devices.

Finally, Android version 2.1 is old, but it is still a quite distributed version between android users¹, as there were no problems to make this application for that version it was a good idea to do it in order to maintain as much as possible the compatibility of older devices.

¹ ¹ Android version 2.1 is the third most used version as of the day this thesis was written according to <http://developer.android.com/about/dashboards/index.html>

2 Programming

The programming was done using prototype methodology, which consisted in try to acquire a readily testing application to receive feedback and going back in previous steps to upgrade and modify the relevant parts.

Following this principle, we have applied a facade software pattern because it's still small enough to contain almost all the logic in few files.

Our first step was to create the interface and the navigation between windows, so our Main and Menu class were created.

2.1 Main and Menu Class

Main.java is the first class to be called by the program and it's only a bypass step, this class just waits 0.1 seconds and then invokes the Menu activity, which will be the one responsible to show the buttons.

```
public void run(){
    try{
        sleep(100);
        Intent menuIntent = new Intent("com.pwroc.ArchTool.MENU");
        startActivity(menuIntent);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    finally{
        finish();
    }
}
```

Code 2-1: Main class invoking Menu class

Menu.java is just a graphical layout with three buttons that allows the user to choose what to do.

We will focus on the important button that invokes the Draw.java activity and it's where almost all our application is written on.

```
Button tut1 = (Button) findViewById(R.id.Draw);
Button tut2 = (Button) findViewById(R.id.Load);
Button tut4 = (Button) findViewById(R.id.cleardata);

tut1.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        //buttonSound.start();
        startActivity(new Intent("com.pwroc.ArchTool.DRAW"));
    }
});
```

Code 2-2: Menu class declaration of buttons and call to Draw class

2.2 Draw Class

After we have done the basic screen navigation then we begin to create the Draw class which is the most important class of the program.

We need first to create the ability to draw objects with the user's finger on the screen; this is done creating a View class called MyView where we are overriding its onDraw method. This method is called all the time the screen needs to refresh its contents.

In order to paint, we need a Canvas, and a Bitmap, the Bitmap is the image that is being generated after drawing in the Canvas (the canvas is no more than the screen of the smartphone).

```
@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.WHITE);
    (...)
    canvas.drawBitmap(mBitmap, src, dst, mBitmapPaint);
}
```

Code 2-3: OnDraw method responsible to update the content of the screen

We are drawing into the current canvas the generated Bitmap that the user has drawn previously, the arguments src, dst are used for the scrolling and mBitmapPaint is just the color configuration of our canvas. We omitted all the previous logic needed to calculate src and dst.

And also we need to capture various events, when the user touches the screen, when the user is drawing on the screen and when the user stops touching the screen. Each one of them is linked to a specified function responsible to capture these events and act accordingly called *touch_start*, *touch_move* and *touch_up*.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    (...)
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            if (scrollMode) {
                fEndX = fStartX = touchX;
                fEndY = fStartY = touchY;
            } else {
                offsetX = 0;
                offsetY = 0;
                touch_start(touchX, touchY);
            }
            invalidate();
            break;

        case MotionEvent.ACTION_MOVE:
            touch_move(touchX, touchY);
            invalidate();
            break;

        case MotionEvent.ACTION_UP:
            if (!scrollMode && !deleteMode && !selectMode) {
                if(doorMode) {
```

```

        touch_up_door();
    } else {
        touch_up();
    }
    invalidate();
}
break;
}
return true;
}

```

Code 2-4: Managing the input events of the user when drawing

As one can see, there are some variables that its purpose is to control in which mode we are, those were implemented later, those modes gives the user the option to scroll the map, to delete a wall, to select a wall and to insert a door.

The *touch_start* function responsible to draw is as follows:

```

fEndX = fStartX = x;
fEndY = fStartY = y;

Point pI = new Point(iStartX, iStartY);

// Tolerance to connect points between lines
PointF newPointF = computeTolerance(pI);
if (newPointF != null) {
    fStartX = x = newPointF.x;
    fStartY = y = newPointF.y;
}

mPath.reset();
mPath.moveTo(x, y);

mBitmap2 = mBitmap.copy(mBitmap.getConfig(), true);

```

Code 2-5: Touch_start function when drawing

Its main purpose is to save that touching point into the global variables *fEndX*, *fStartX* (idem for Y axis), that later will be used.

Also it needs to make a copy of the current bitmap.

We can see a new function called *computeTolerance()*, it is responsible to detect if there is another point near of the one that the user touched, and if it exists use that point instead.

The *touch_move* function responsible to show how the user is drawing a line in the screen is as follows:

```

float dx = Math.abs(x - fEndX);
float dy = Math.abs(y - fEndY);
// If the movement is too small ignore it
if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {
    // Retrieve original bitmap
    mBitmap = mBitmap2.copy(mBitmap2.getConfig(), true);
    mCanvas.setBitmap(mBitmap);
    // Draw the line
    mCanvas.drawLine(fStartX, fStartY, x, y, mPaint);
    // Update global variables
    fEndX = x;
}

```

```

        fEndY = y;
        iEndX = (int) x;
        iEndY = (int) y;
    }

```

Code 2-6: Touch_move function when drawing a line

It is reading the new point where the user is touching and draws a line into that point, but every time it destroys that line through the use of the bitmap copy. The original one that we copied in the Touch_start function, so creating the illusion that the user is just drawing a shadow of a line.

The final line will be drawn when the user will stop touching the screen, in the touch_up function as follows:

```

PointF fEndP = new PointF(fEndX, fEndY);
Point iEndP = new Point(iEndX, iEndY);

// Delete any action from move event
mBitmap = mBitmap2.copy(mBitmap2.getConfig(), true);
mCanvas.setBitmap(mBitmap);

float distx = Math.abs(fEndP.x - fStartX);
float disty = Math.abs(fEndP.y - fStartY);
double dist = Math.sqrt(distx * distx + disty * disty);

// Do not draw too small lines
if ((dist < 20) ) {
    mPath.reset();
    return;
}

// Connect end Point with current existing points if possible
PointF newPointF = computeTolerance(iEndP);
if (newPointF != null) {
    fEndX = newPointF.x;
    fEndY = newPointF.y;
    fEndP = new PointF(fEndX, fEndY);
}

// Mark all the coordinates of the points of the entity being
// drawn
byte entityId = findAvailableEntityID();
byte[][] entityCoordinates = new byte[MAP_RESOLUTION][MAP_RESOLUTION];
entityCoordinates = calcMapCoordinates((int) fStartX, (int) fStartY, iEndX,
iEndY, entityId);
updateMapCoordinates(entityCoordinates, entityId);

// Create entity object
DrawEntity oDrawn = new DrawEntity(entityId);
PointF fStartP = new PointF(fStartX, fStartY);
int length = computeDistance(fStartP, fEndP);

oDrawn.SetStartPoint(fStartP.x, fStartP.y);
oDrawn.SetEndPoint(fEndX, fEndY);
oDrawn.SetLength(length);

int angle = computeAngle(fStartP, fEndP);
oDrawn.SetAngle(angle);

```

```
// Add entity object to the list of entities
aEntities.add(oDrawn);

mPath.lineTo(fEndX, fEndY);

// commit the path to our canvas
mCanvas.drawPath(mPath, mPaint);
mPath.reset();
```

Code 2-7: Touch_up function finally drawing a wall

We put comments on the code to explain what it does, but at this point we have to explain our data structure that we are using to keep all the drawings and are mentioned in this code snippet. We want to comment that what we call an *Entity* is a line being drawn into the map (or a wall in the current state of the prototype, in the future it could be a column, or a door, etc)

For that we created the DrawEntity Class.

2.3 DrawEntity Class

Is just a class that holds all the relevant information of a wall.

```
public class DrawEntity implements Serializable {
    (...)
    public byte id;
    private float fStartX, fStartY, fEndX, fEndY;
    private short length = 0;
    private short angle = 0;
    boolean pictures = false;
    (...)
}
```

Code 2-8: DrawEntity Class definition

We omitted all the getters and setters.

And so finally we are ready to speak what is the relevant information that defines a map:

- Current drawn **mBitmap** (holds the drawn map)
- The Data array **mapCoordinates**, responsible to divide the map in sections and label them depending if on these sections there is an entity or not.
- The Data array **aEntities**, responsible to store all entities in the map.
- The **ID** of the current map
- The pictures captured from camera application and linked to entities.

mapCoordinates is used to know if the user selects a wall or deletes it.

For the pictures, as they are already saved into the SD card as per the Camera application works, we only need the name of the picture so a string will be enough.

The first point, the Bitmap, is stored in the SD aswell, using the ID of the current map for later locate it. The rest of this information is stored into an object class called "saveObject".

2.4 saveObject Class

This class holds the relevant data to be saved into our SQLite database.

```
public class saveObject implements Serializable {
    public byte mapCoordinates[][];
    public ArrayList<DrawEntity> aEntities = new ArrayList<DrawEntity>();
    public int Id;
    public String bitmapFlnm;
}
```

Code 2-9: saveObject Class definition

This saveObject is serialized and then inserted into a SQL database when the user triggers the save option.

2.5 DataManipulator Class

This is the Database class that holds all the calls to communicate with the SQL database, called DataManipulator.class

```
public class DataManipulator {
    (...)
    static final String TABLE_NAME = "newtable";
    (...)
    private static Context context;
    static SQLiteDatabase db;

    private SQLiteStatement insertStmt, updateStmt;

    private static final String INSERT = "insert into " + TABLE_NAME + " (data) values (?)";

    private static final String UPDATE = "UPDATE " + TABLE_NAME + " SET data = (?) WHERE id = (?) ";

    public DataManipulator(Context context) {
        DataManipulator.context = context;
       OpenHelper openHelper = newOpenHelper(DataManipulator.context);
        DataManipulator.db = openHelper.getWritableDatabase();
    }
    public long insert(byte[] object) {
        this.insertStmt = DataManipulator.db.compileStatement(INSERT);
        this.insertStmt.bindBlob(1, object);
        return this.insertStmt.executeInsert();
    }
    (...)
    public long update(int id, byte[] object) {
        this.updateStmt = DataManipulator.db.compileStatement(UPDATE);
        this.updateStmt.bindDouble(2, id);
        this.updateStmt.bindBlob(1, object);
        return this.updateStmt.executeInsert();
    }
    (...)
    public List<String> selectPathology(int houseid, byte entityid) {

        String sql = "SELECT * FROM " + TABLE_NAME2 + " WHERE houseid = ? AND
```



```

entityid = ?";

    List<String> list = new ArrayList<String>();
    Cursor cursor = db.rawQuery(sql, new String[] { Integer.toString(houseid),
Integer.toString(entityid) });

    if (cursor.moveToFirst()) {
        do {
            String description = cursor.getString(3);
            String damage = cursor.getString(4);
            list.add(description);
            list.add(damage);
        } while (cursor.moveToNext());
    }

    return list;
}
(...)
public void close() {
    db.close();
}

private static class OpenHelper extends SQLiteOpenHelper {
    OpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE_NAME + " (id INTEGER PRIMARY
KEY, data BLOB)");
        (...)
    }
}
(...)
}

```

Code 2-10: DataManipulator Class responsible to communicate with the SQLite Database

With this we can save and load, for the load we just need to create a screen with the list of the saved maps.

2.6 Inserting Pathologies

At this point the next step was to take pictures and link them to the walls, in order to do that we created a mode called "select mode", this logic is in the *touch_start* function and what it does is to check if the user has touched an entity (thanks to *mapCoordinates* we know about that). And then create a popup window with the information of that wall.

```

} else if (selectMode) {
    if (mapCoordinates[cellx][celly] > 0) {
        final PointF pStart, pEnd;
        int length, angle;
        byte id = mapCoordinates[cellx][celly];

        // Gather entity information
        final DrawEntity oDrawn = getEntity(id);
    }
}

```

```

        pStart = oDrawn.GetStartPoint();
        pEnd = oDrawn.GetEndPoint();
        length = oDrawn.GetLength();
        angle = oDrawn.GetAngle();

        // Create popup window
        LayoutInflater inflater = (LayoutInflater)
getBaseContext().getSystemService(LAYOUT_INFLATER_SERVICE);
        View popupView = inflater.inflate(R.layout.popup, null);
        final PopupWindow popupWindow = new PopupWindow(popupView,
LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT, true);

        (... declaration of buttons and its listeners ...)

        // This launches the camera application
        camButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                dispatchTakePictureIntent(ACTION_TAKE_PHOTO, oDrawn.id);
                oDrawn.SetPictures(true);
                repaint(oDrawn, oDrawn.GetLength(), "Length");
            }
        });

        // This opens the Gallery Window
        galButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                Intent loadGallery = new
Intent("com.pwroc.ArchTool.BROWSERGALLERY");
                loadGallery.putExtra("houseID", Id);
                loadGallery.putExtra("entityID", oDrawn.id);
                startActivity(loadGallery);
            }
        });

        // This is the Ok button to save changes
        Button btnOk = (Button) popupView.findViewById(R.id.btnOk);
        btnOk.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                if (saveLength || saveAngle) {
                    int iLength = 0;
                    int iAngle = 0;
                    // The user has modified the angle
                    if (saveAngle) {
                        try {
                            iAngle =
Integer.parseInt(editAngle.getText().toString());
                        } catch (NumberFormatException nfe) {
                            System.out.println("Could not parse " + nfe);
                        }
                        repaint(oDrawn, iAngle, "Angle");
                        oDrawn.SetAngle(iAngle);
                        saveAngle = false;
                    }
                    // The user has modified the length
                    if (saveLength) {
                        try {
                            iLength =
Integer.parseInt(editLength.getText().toString());
                        } catch (NumberFormatException nfe) {

```

```
        System.out.println("Could not parse " + nfe);
    }
    repaint(oDrawn, iLength, "Length");
    oDrawn.SetLength(iLength);
    saveLength = false;
}

// Update MapCoordinates with the new changes
byte entityCoordinates[][];
PointF pStartTmp = oDrawn.GetStartPoint();
PointF newpEnd = oDrawn.GetEndPoint();
entityCoordinates = calcMapCoordinates((int) pStartTmp.x,
(int) pStartTmp.y, (int) newpEnd.x, (int) newpEnd.y, oDrawn.id);
updateMapCoordinates(entityCoordinates, oDrawn.id);
}
```

Code 2-11: Logic of selecting a wall and changing its attributes

We have lots of thins happening in this window, for one side the user can change the length and angle of the wall, so it needs to be repainted again with the new attributes. That's the purpose of *repaint* function on this code snippet.

Later, *camButton* launches the camera application to take a picture, and *galButton* launches another window called *BrowserGallery*, where the pictures of the wall has been stored and also it contains its description. The class responsible of the *BrowserGallery* is called *PicSelectActivity.java*

At this point, the basic things of the application were done, but then we developed other functionalities as to help the user as:

- **EraseMode:** Allows the user to delete a wall, reusing the select logic we explained before.
- **ScrollMode:** Allows the user to scroll the map if it's too big to hold in the screen.
- **VectorMode:** It helps the user to draw maps, restricting the drawing walls to be oriented into one axis's coordinates.
- **DoorMode:** Allows the user to draw a sort of entrance for each room.
- **CheckList:** Allows the user to input from a checklist the minimum requirements that the flat should comply.

And then we began with the testing phase, during this process we manage to make really good improvements on memory efficiency by 500% because at the beginning we were using data structures that were consuming too much resources. (Related of what things to store in the SQL database and the logic to detect and connect points when the user is drawing).

3 Description

We called our application "*Architect Support Tool*" (AST) and this is the main screen presented to the user, where he can choose to Draw a map, Load a previous saved map, or Clear all the data of the application.

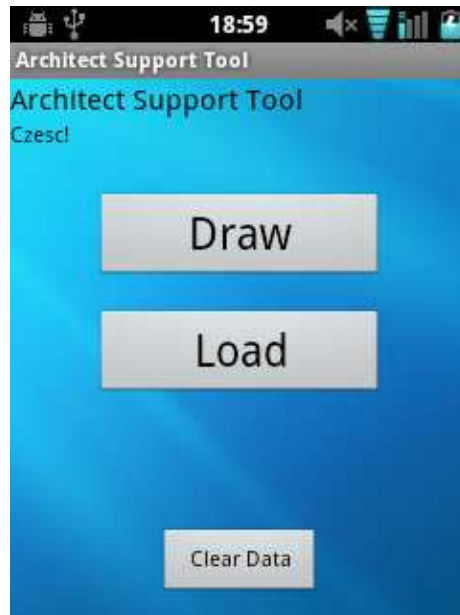


Image 3.1: Menu screen of AST

3.1 Drawing screen

This is the drawing screen and the option menu to select all the options of the application.

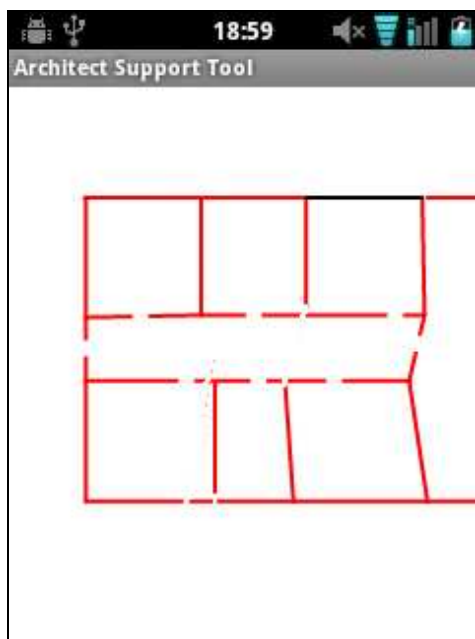


Image 3.2: Drawing screen

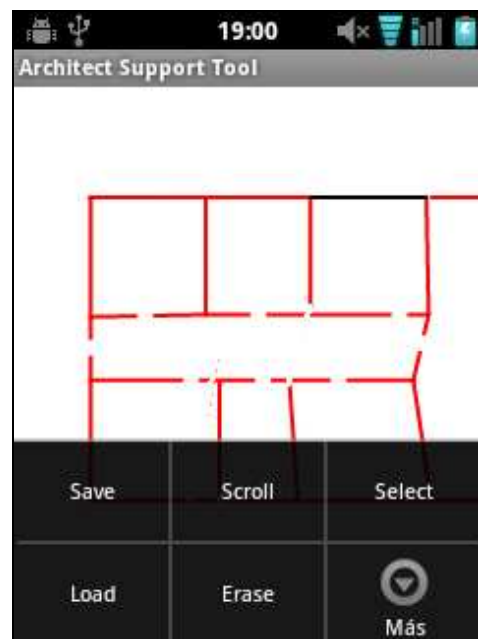


Image 3.3: Option menu

3.2 Entity screen

The entity window is a popup window that appears after selecting a wall (through the option menu "Select"). Let's say that the user have found a pathology in this encircled wall.

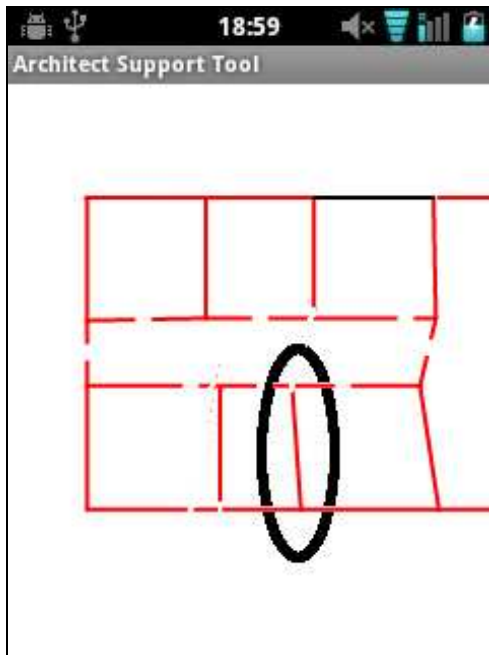


Image 3.4: Wall that the user selects

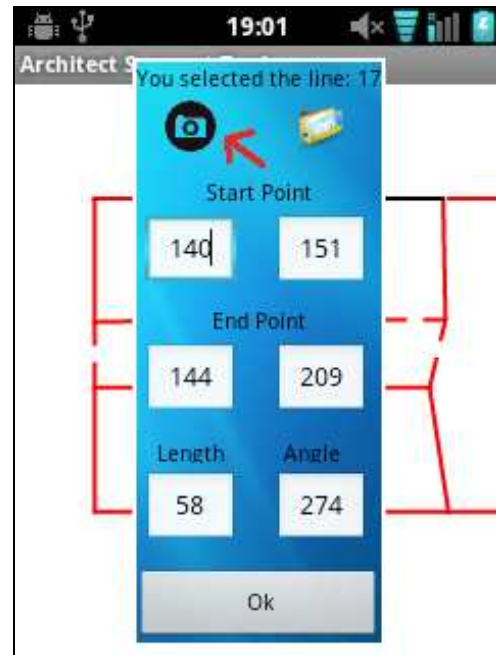


Image 3.5: Popup entity window

We take notice of the camera icon; it will allow the user to take a picture of the pathology using the internal camera and save it into this wall.

The parameters **Length**, and **Angle**, can be modified by the user to meet new criteria, and the system will redraw the new wall as the desired result.

After the user has made a picture the resulting map will be displayed like this, note the change of colour of the wall, **red** means no pathology inserted, **black** means that wall has a pathology.

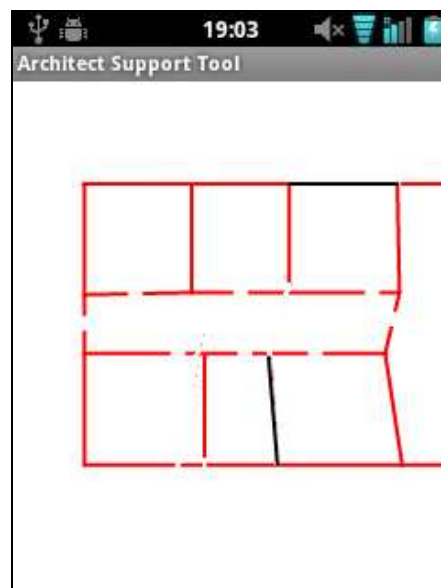


Image 3.6: Black wall after inserting pathology

3.3 Gallery screen

This screen is accessed through the folder icon in the Entity screen. The user can navigate through all the pictures (pathologies) that the selected wall has. And he can write a description to it.



Image 3.7: Gallery window



Image 3.8: Navigating between photos

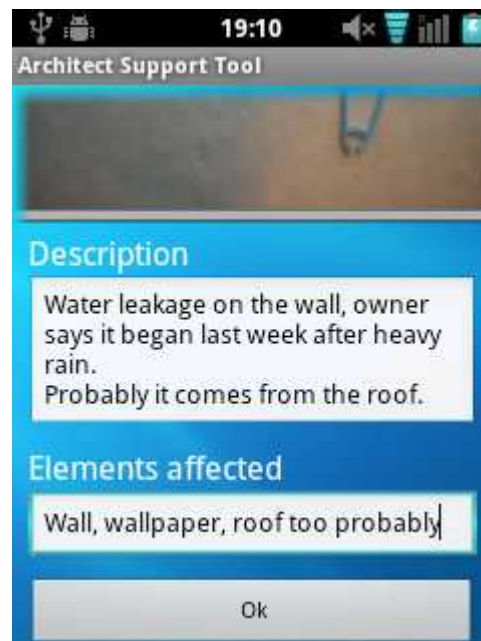


Image 3.9: Gallery introducing info

When the user inserted all the desired information, he only needs to push the "Ok" button to save it. If he does not want to save it then he needs to push the back button.

3.4 Scrolling

After when the user selects the Scroll button in the option menu, he will be on "Scroll Mode", now he can touch the screen and move his finger in order to move the map with its movement at the desired place. When he will finish he will need to push the scroll button option again to leave from the "Scroll Mode".

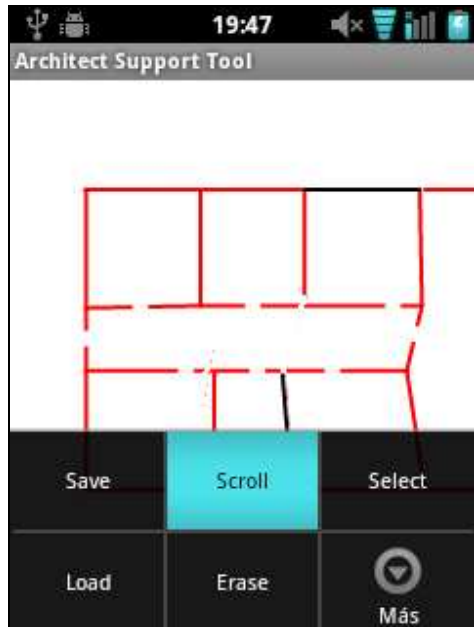


Image 3.10: Option menu "Scroll"

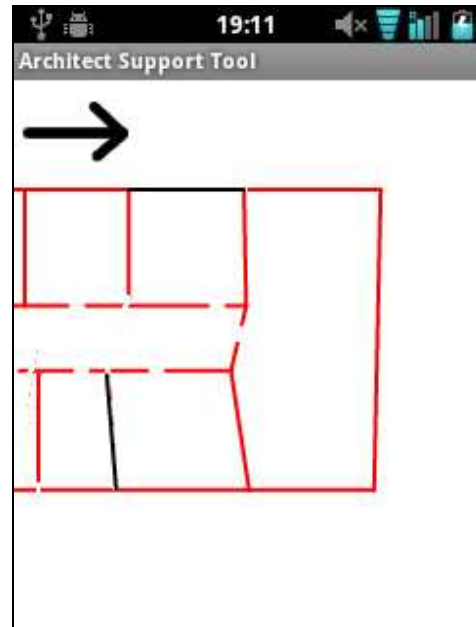


Image 3.11: Scrolling map

3.5 Saving

The save option is accessed through the Options menu and will save the current map and all its relevant information. It will rewrite the existing file if it exists, and if not, it will create a new entry.

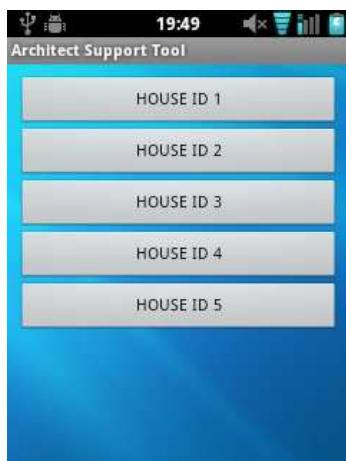


Image 3.13: Load Option

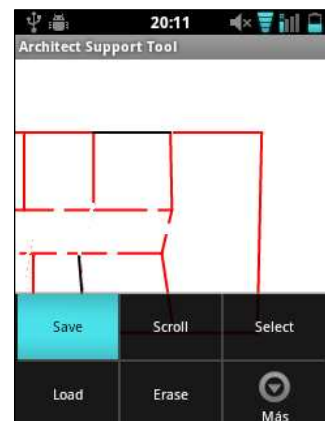


Image 3.12: Save Option

3.6 Loading

Accessed through the Option Menu this window will show us all the current saved maps in order to load them.

3.7 Create a new map

This option does nothing more than to create a new blank map.

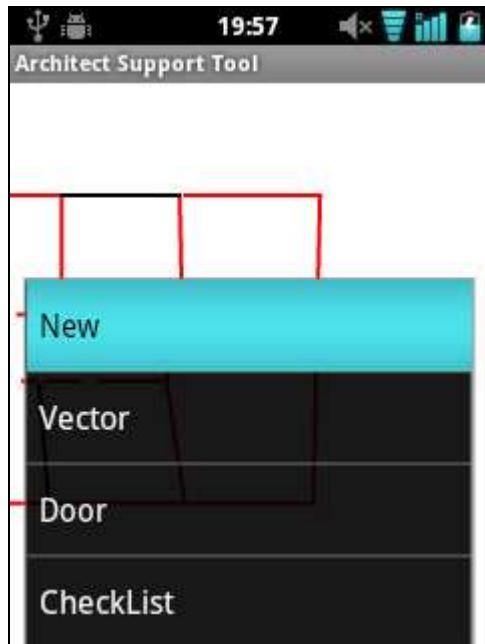


Image 3.14: Option menu "New"



Image 3.15: New blank map

3.8 Erase a wall

This option allows the user to touch a wall and remove it, to activate it the user has to touch the "Delete" option menu.

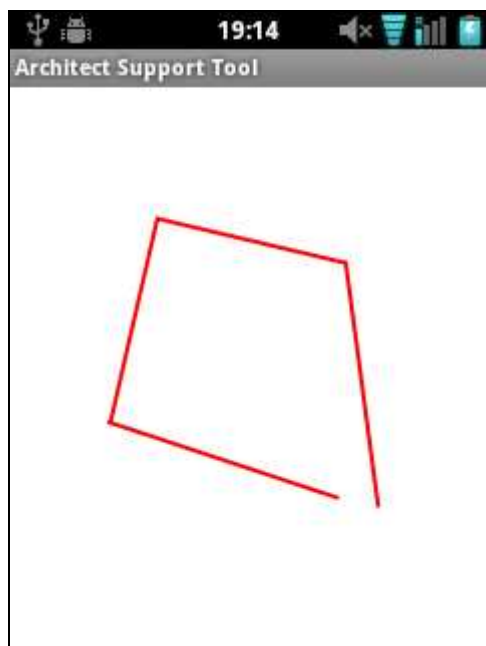


Image 3.16: Mistakenly placed wall

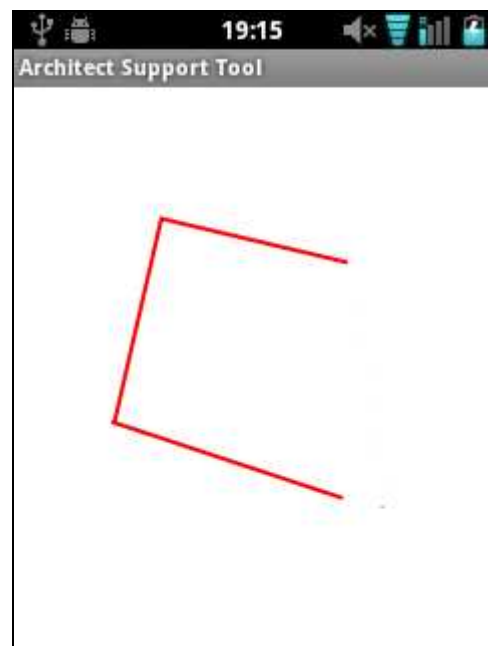


Image 3.17: Wall being removed out

3.9 Connecting Points Core

We cannot expect a human person to be as precise to select exactly the point where he wants to draw, that's why this system catches the point where the user touched the screen (or stopped touching the screen) and begins to search if there is a nearby wall to connect it.

The tolerance distance for connection is set up to 20 pixels radius away.

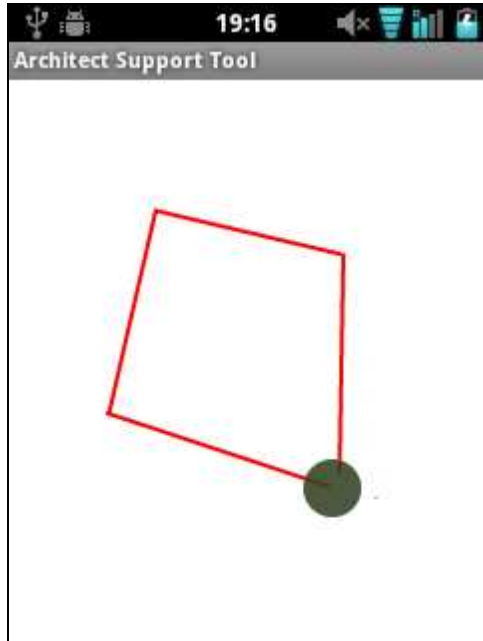


Image 3.18: Area of tolerance to connect points

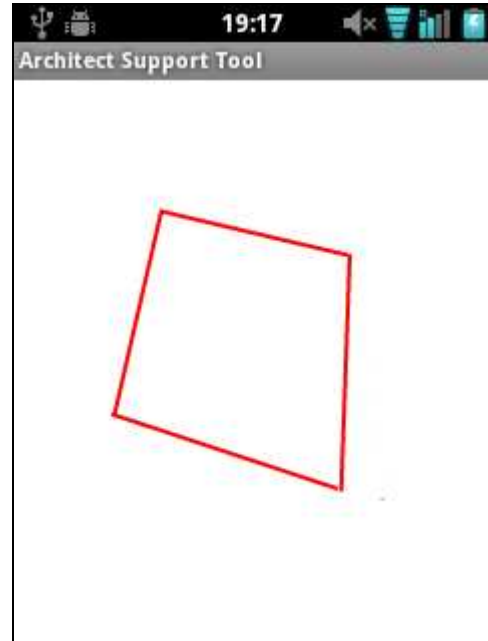


Image 3.19: Line after automatically connecting the walls

3.10 Vector mode

This is another feature in order to help the user to draw maps, until now we have seen that the user can draw straight lines with free angle (they can be 18 degrees, 173 degrees, whatever, as long the finger is moving through the screen). Sometimes this is impractical if one wants to draw rooms and flats in a faster way.

The Vector mode enables the user to only draw straight lines at 0° , 90° , 180° and 270° , as this application is not intended (for the moment, at this current prototype stage) to be an exact representation of the world, but instead, a support tool, we think the vector mode will cover the needs of drawing rooms and flats most of the time.

As usual, this option is enabled through the option menu, and in order to deactivate it one has to touch it again.

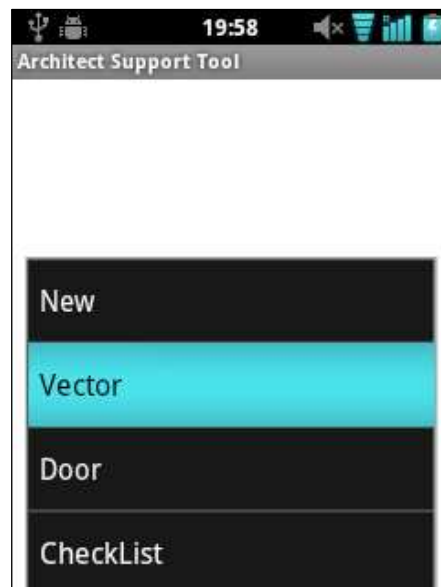


Image 3.20: Activating Vector Mode

3.11 Door Input

The ability to put doors is important in order to see where the entrances of the respective rooms are, as this is a prototype there were no time to drag and drop a door icon, so we used the ability to "remove" partially smaller sections of the walls.

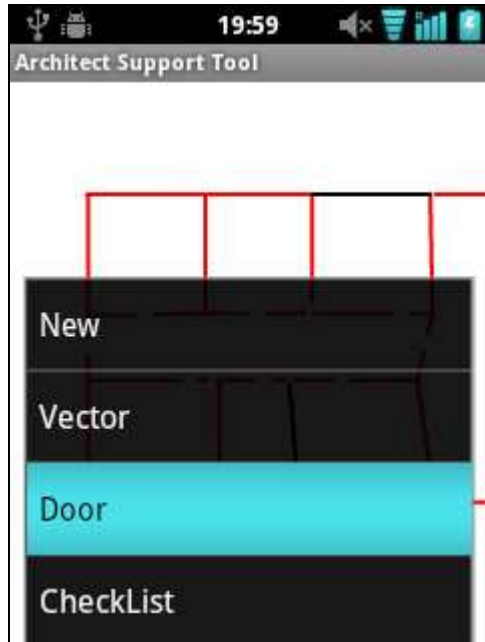


Image 3.21: Option menu "Door"

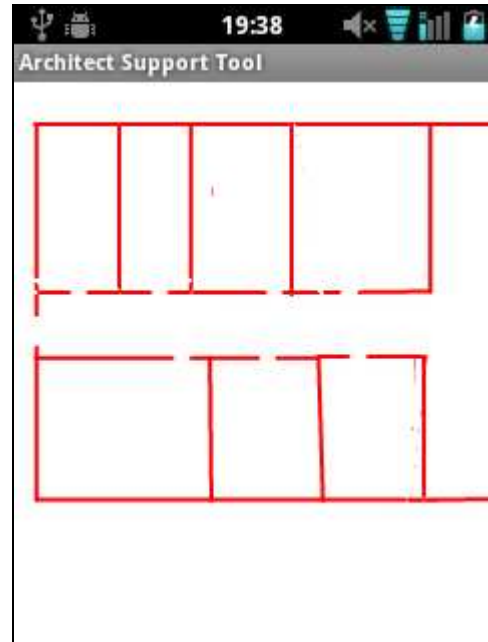


Image 3.22: Flat with doors

When the user checks the Door option menu, he enters into the Door Mode. In this mode he can draw "white" lines, effectively removing the small sections where he touches with the movement of his finger. These white lines does not generate any type of data, they only modify the bitmap, nothing else.

Using it in a smart way, one can give the illusion of drawing entrances into each room or the flat itself, as shown in the picture 3.22.

The user is requested to touch again the Door option menu in order to leave out from this mode.

3.12 Checklist

This feature allows the user to check a list of elements that the flat should have. The user can check those elements for information purposes, or try to fill in the new information. It consists of a list of elements, where the user can check or uncheck what he feels is being complied with the quality standards of the flat or not.

As basic as this list is, for this prototype stage, they are probably the most important to check, the current elements to check is as follows:

- Dimensions: If checked means that the flat has a proportionate and adequate size for its purpose (it's not the same a flat for a living than an study flat).
- Ventilation: If checked means that the flat has enough exhaust ventilations.

- Windows: If checked means that the flat has all its windows in good condition, and are sufficient for the flat.
- Smoke Pipe: If checked the smoke pipes do not have any problem and they do exist.
- Gas Pipe: If checked the gas pipes do not have any problem and they do exist.
- Lights: If checked, the flat has enough lights to make it a comfortable place through all the room.
- Plugs: If checked, the flat has enough plugs connections through all the rooms.

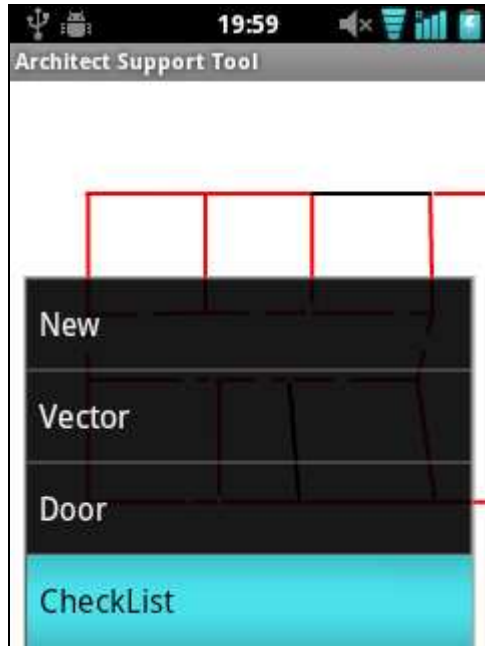


Image 3.23: Option menu "CheckList"

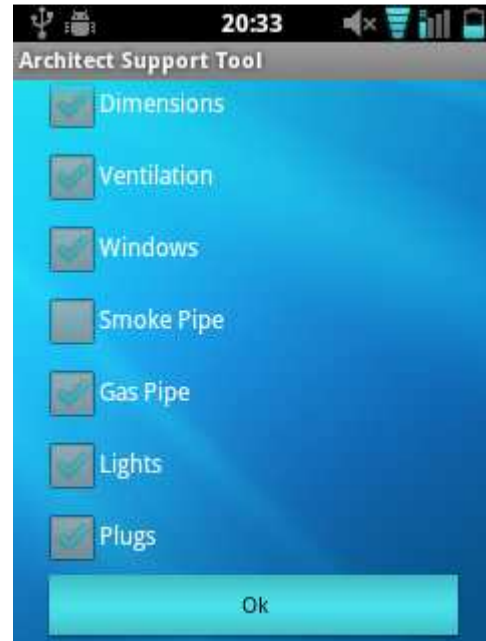


Image 3.24: Accepting changes

When the user selects the Checklist Option menu, the Checklist screen will appear with all the elements to be checked. If the user does not want to make any modification or just to read it, he will need to touch the back button, instead of the Ok button.

All maps have one checklist associated to them.

3.13 Clear data

This button allows the user to delete all the information of the system, pictures, and databases (related to the application of course).

And with all of this, the general description of the Application is done.



Image 3.25: Clear Data button

2012

Architect Support Tool

User Manual

Alexandre Uroz Làzaro

Title: Development of a software tool to support architects

Volume: 1/1

Student: Alexandre Uroz Làzaro

Tutor: Dr. Krzysztof Waśko

Department: Systemy multimedialne i mobilne

Date: 24 of September 2012

Minimum Requeriments

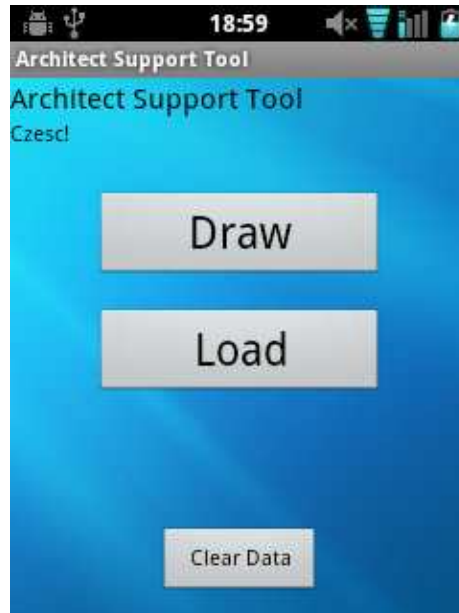
Smartphone with android version 2.1 system with integrated camera and sd card.

Installation

For now, just copy the ArchTool.apk in your smartphone and execute it to install it, in the future it will be published on Play Store.

AST User Manual

Architect Support Tool user manual for architects.



All information in this manual is based on a prototype, so the appearance of errors, bugs, or instability is, as undesirable it is, expected. We ask for the user reading this manual to understand these limitations, and try to use it as best as his possibilities while it is being improved.

Welcome

The Architect Support Tool (from now on refereed simply as AST) is an application to support and help architects in his daily duties. We hope that this prototype is being focused on the right path and it will be useful for you.

Being said that we are glad that you have deposited your trust in us and we are going to fully explain how AST can help you.

As an architect involved with building maintenance it is usual to enter in buildings to do inspections and then create reports about their status. The way this is being done is using a scientific method; to do a visual inspection and take notes and photos and fast drawings of the building and its problems.

So in the end, the architect ends up being some kind of Christmas tree with the camera on one hand, the notebook (yes one of those made of paper), few pencils and pens at hands and pockets, a torch, a tape measure and a laser meter. So this application for a cellphone, with a good camera capable of doing everything an architect do with pencils and paper, will be of a great help in many ways. It will be a simplification of the working time because with a couple of tools the architect will be capable to do his work. Moreover it will be a way to win time, because the information is clearly clean and in order.

And we hope AST will manage to accomplish this goal, though for the moment is just a prototype.

1	DRAWING SCREEN	- 1 -
1.1	OPTIONS MENU	- 1 -
1.1.1	<i>Selecting</i>	- 1 -
1.1.2	<i>Scrolling</i>	- 2 -
1.1.3	<i>Saving</i>	- 3 -
1.1.4	<i>Loading</i>	- 3 -
1.1.5	<i>Create new map</i>	- 3 -
1.1.6	<i>Erasing walls</i>	- 3 -
1.1.7	<i>Vector mode</i>	- 4 -
1.1.8	<i>Door Input</i>	- 4 -
1.1.9	<i>Checklist</i>	- 4 -
1.2	PATHOLOGY SCREEN	- 5 -
2	CLEAR DATA	- 5 -
3	REAL CASE SCENARIOS	- 6 -
3.1	EVALUATING LIVING CONDITIONS	- 6 -
3.2	PATHOLOGY INSPECTION	- 7 -
3.3	CONCLUSION	- 7 -
4	EPILOGUE	- 7 -

1 Drawing screen

This is the drawing screen and the option menu to select all the options of the application. To draw you only need to paint lines using your finger on the screen.

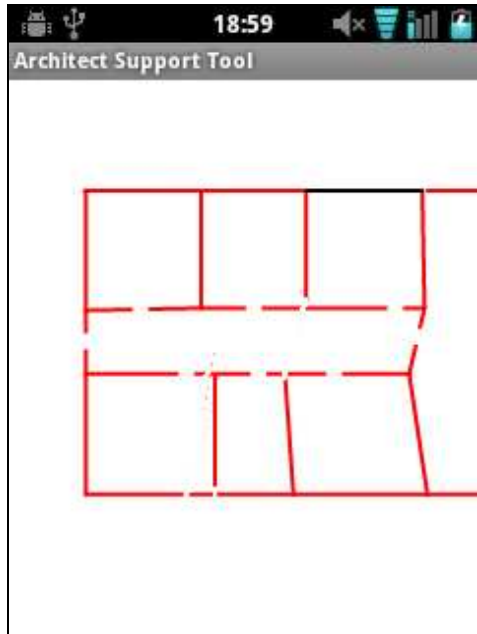


Image 1.1: Drawing screen

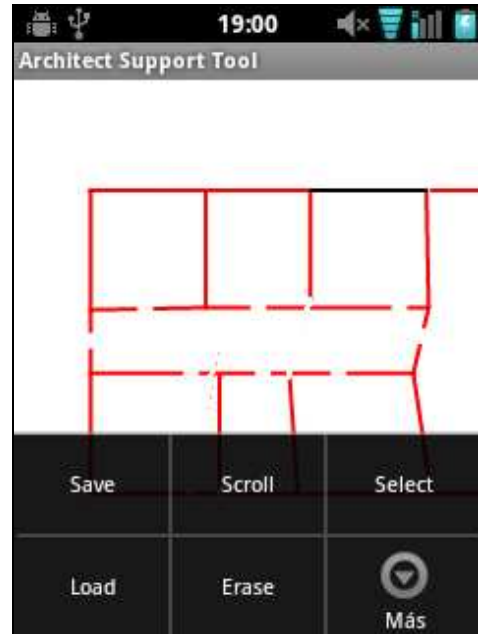


Image 1.2: Option menu



1.1 Options Menu

1.1.1 Selecting

Press the *Select* button and touch the wall that you want to see all the information related to it. You will see in this screen:

- Starting Point
- Ending Point
- Angle (Degrees)
- Length (In Decametres)

You will be able to modify the Angle and Length parameter, to redraw the wall as you wish.

Through this screen you will be able to take pictures of **pathologies** through the Camera icon  and view and insert descriptions to them through the Folder icon .

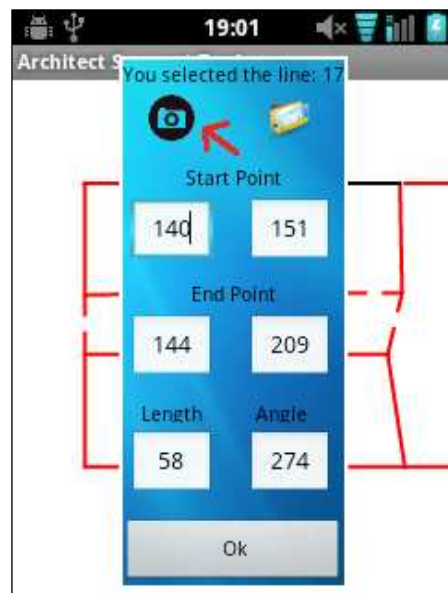


Image 1.3: Selection window

When you will take a picture the wall will change its colour from **red** (means no pathology inserted) to **black** (means that wall has a pathology).

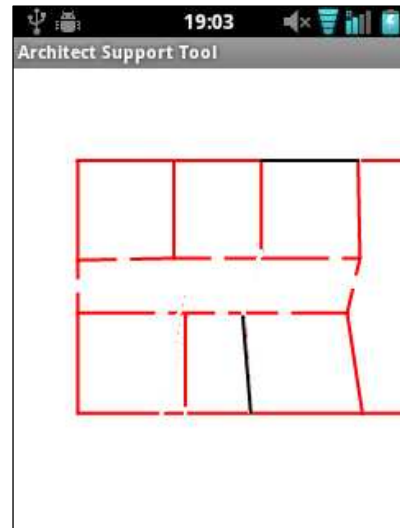


Image 1.4: Black wall after inserting pathology

1.1.2 Scrolling

Press the *Scroll* button to enter into the scroll Mode, now you can touch the screen and scroll the map to see all of its contents. Press again the Scroll button to leave from this mode.

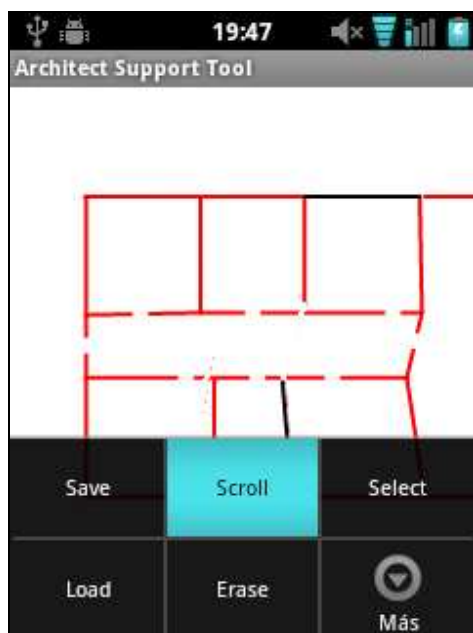


Image 1.5: Option menu "Scroll"

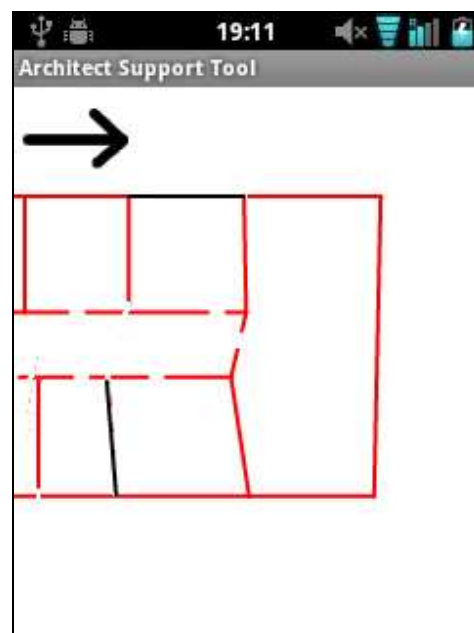


Image 1.6: Scrolling map

1.1.3 Saving

Press the *Save* button to save your current map.

1.1.4 Loading



Image 1.8: Load Option

Press the *Load* button to load any previously saved map.

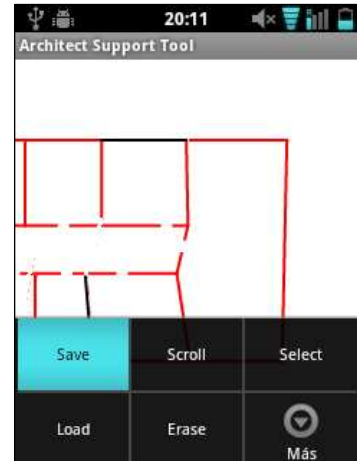


Image 1.7: Save Option

1.1.5 Create new map

Press the *New* button to create a new blank map.

1.1.6 Erasing walls

Press the *Erase* button to enter into the erase Mode, in this mode any wall you touch will be deleted. Press the *Erase* button again to leave from this mode.

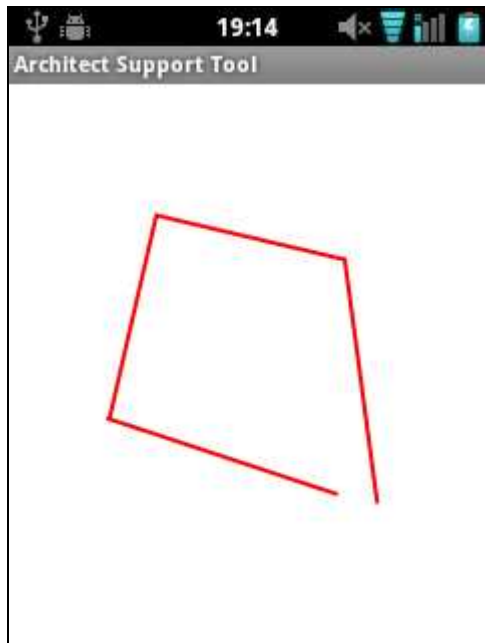


Image 1.9: Mistakenly placed wall

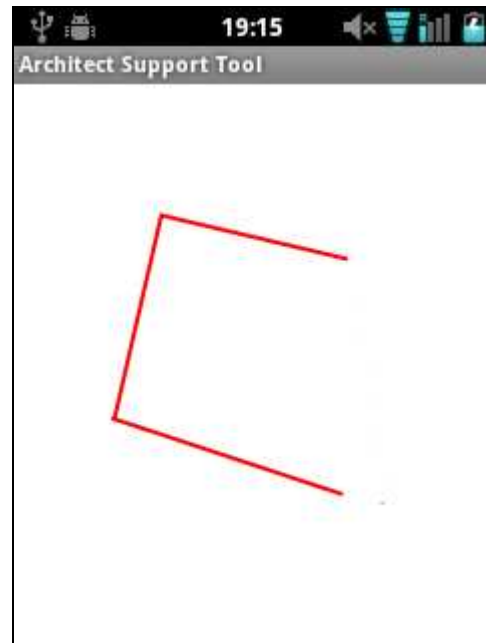


Image 1.10: Wall being erased

1.1.7 Vector mode

Press the *Vector* button to enter into vector Mode drawing. In this mode you can only draw walls at 0°, 90°, 180° and 270° degrees. Use it in order to draw faster precisely maps. Press the *Vector* button again to leave from this mode.

1.1.8 Door Input

Press the *Door* button to draw doors in the map. This will allow you to remove small portions of walls like if you were using an eraser.

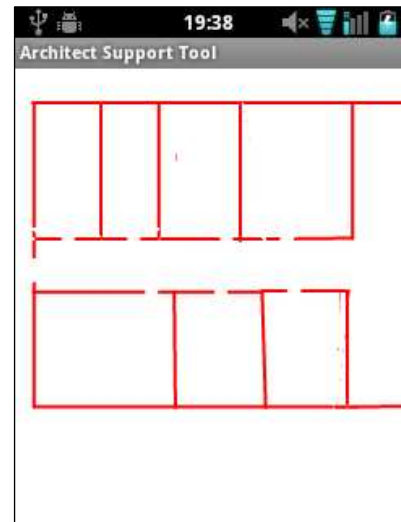


Image 1.11: Flat with doors

1.1.9 Checklist

Press the *Checklist* button to see the list of elements this flat complies with. You can verify all this qualities:

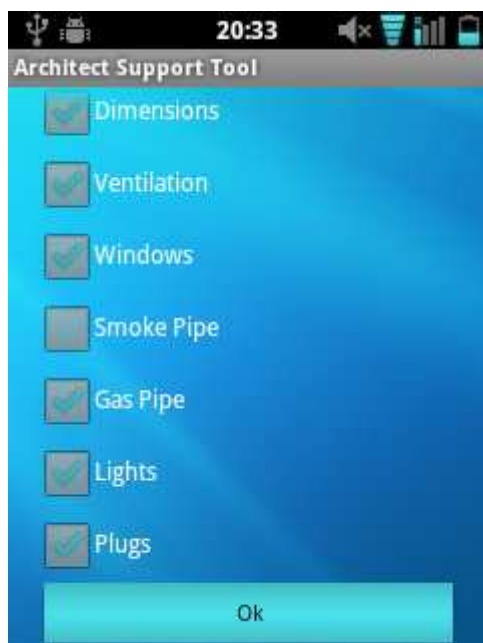


Image 1.12: Checklist window

- Dimensions: If checked means the flat has a proportionate and adequate size for its purpose.
- Ventilation: If checked means that the flat has enough exhaust ventilations.
- Windows: If checked means that the flat has all its windows in good condition, and are sufficient for the flat.
- Smoke Pipe: If checked the smoke pipes do not have any problem and they do exist.
- Gas Pipe: If checked the gas pipes do not have any problem and they do exist.
- Lights: If checked, the flat has enough lights to make it a comfortable place through all the room.
- Plugs: If checked, the flat has enough plugs connections through all the rooms.

1.2 Pathology screen


Through the select screen (point 1.1.1) press the *folder* icon  to see the pathologies this wall has and insert, modify or read its **description** and **elements affected**



Image 1.13: Gallery window

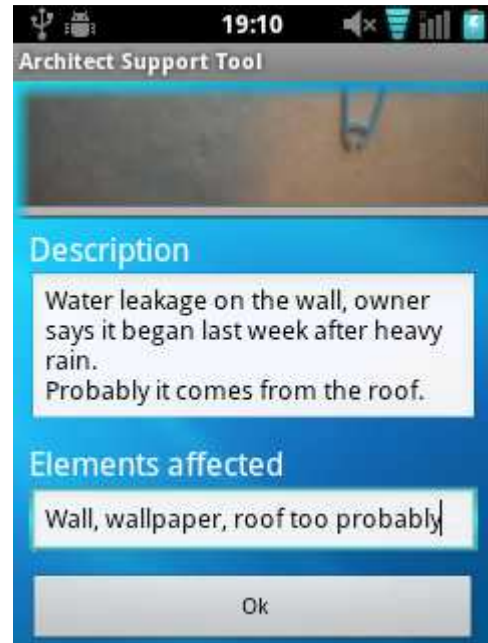


Image 1.14: Description of the pathology

2 Clear data

Press the *Clear Data* button from the main screen to delete **all the application data** from your mobile device. **Be sure to do that!**



Image 2.1: Clear Data button

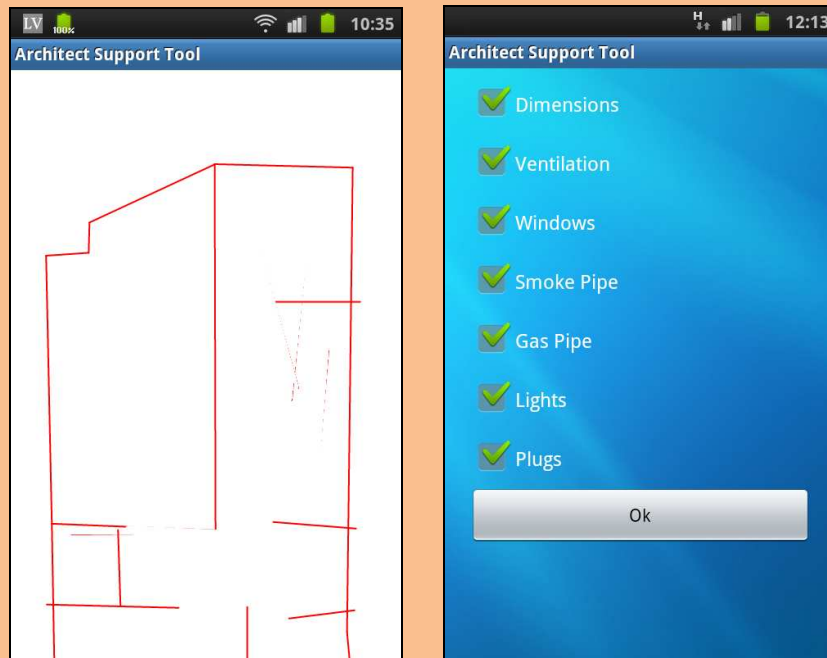
3 Real case scenarios

These are real examples of use of this application from an architect point of view, Jaume Casadevall Puig from Catalonia let him to speak directly to you of how to use AST from his experience.

3.1 Evaluating living conditions

Here at Catalonia and surely on other countries, there is a specific law about the living conditions for houses. There are some requirements I have to check. Some are the dimensions of the pieces and others are about installations (how many plugs are on the house, lights on all the pieces, etc.)

I've made a test. I've made a fast drawing of the flat and also used the app checklist to check everything is all right.

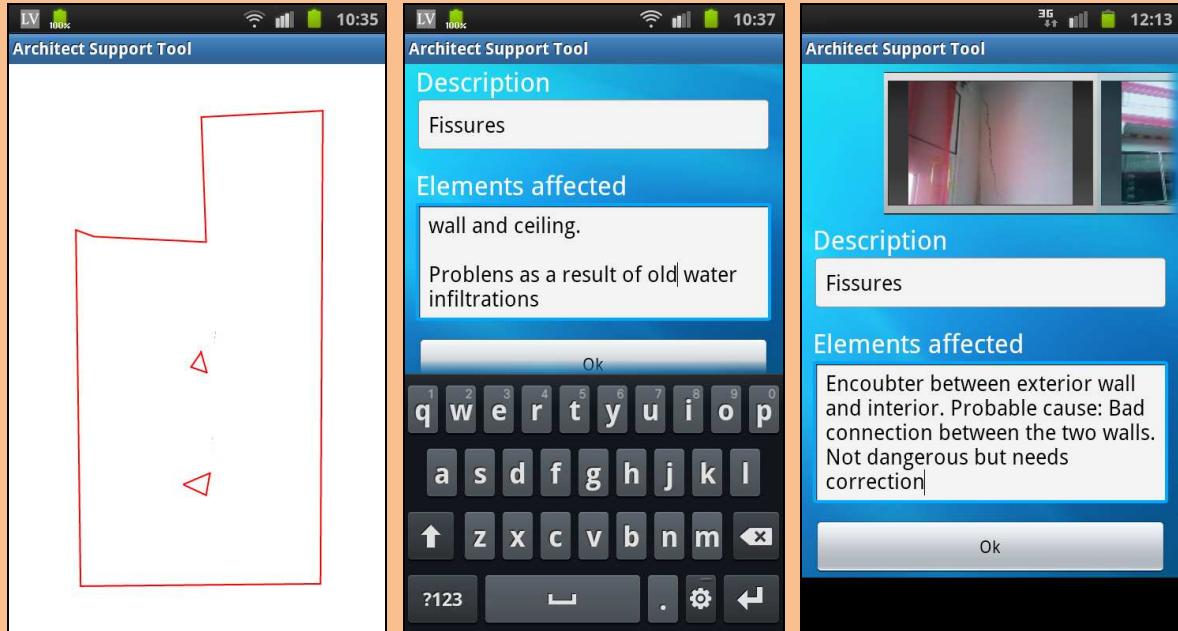


Here we can see the fast sketch drawing and also the checklist

The conclusion is that it was a good way to speed up all the process and simplify the way I do field work.

3.2 Pathology inspection

Another one of my jobs is to perform inspections on old buildings searching for problems. In that case I have to do fast drawings too and create notes about the pathologies I found.



Here you can see the fast notes the app is capable to produce

3.3 Conclusion

This app sure is a prototype but at the actual point it is already a powerful app that in many ways earns me time on my daily work. It's a fast way to do things I already do and few more that I don't, and moreover it is a way to do it all in a better way.

4 Epilogue

And so this concludes the purpose of this manual, explaining all the options that the application offers, and how to use them even from an architect's point of view. We hope you will be glad to use it. And we cannot end this manual without giving special thanks to Jaume Casadevall Puig who, kindly, offered himself to help us in order to write this manual.