Virtual Street Art

Master Thesis

VGIS 11grp1024

Proposed by:

Pierre Ducoudray

Julien Plaquevent

Cyril Schmitt

Aalborg University

Department of Electronic Systems

AALBORG UNIVERSITY

TITLE
Virtual Street Art

PROJECT PERIOD
VGIS 10th semester
February 2011 - June 2011

PROJECT GROUP
11grp1024

GROUP MEMBERS
Ducoudray Pierre
Plaquevent Julien
Schmitt Cyril

SUPERVISOR
Jakob Schou Pedersen

COPIES
3

TOTAL PAGES
84

Abstract

This report documents the creation of a system to virtualize street art. Street art is any image or lettering painted, marked or drawn on any surface.

The idea is to virtualize the process of making a street art. Each tools is replaced by a technological device. The user can use his/her iphone as a spray can to paint on a wall. To satisfy that need, the system integers an iphone application to simulate a spray can and a computer side application to render the street art on the wall. Obviously, a projector is used to project the render on the wall.

The system is designed a way that the user does not need any technical knowledge. The main objective is to entertain the user.

The iphone application is implemented with Objective-C as it is the only option. On the other hand, the computer side application is developed in Java and Processing to make it cross platform compatible.

This report presents the system development life cycle, from the preanalysis to the tests.

Preface

This report and all of its content are part of the 10th semester in Vision, Graphics and Interactive Systems at Aalborg University. It has been written by the project group VGIS 1024 during the spring semester of 2011.

The semester purpose is to validate all the knowledge learnt by doing a master thesis project. The main theme of the project is: How to virtualize street art?

Report outline

The report is divided into seven parts:

1. **Introduction:** Presentation of the project and all its components.
2. **Preanalysis:** Study of former projects which manage to solve similar problem.
3. **Analysis:** Information about technologies used in the project. This part is an analysis of the problem at hand. It also divides the main problem into smaller components.
4. **Design:** Explanation of how the components of our project are designed.
5. **Implementation:** Realization of the programming tasks. It explains each part of the program.
6. **Testing:** Presentation of all the tests made during the project and their results.
7. **Conclusion:** Brief overview of the project and presentation of the final solution.

Authors

Ducoudray Pierre　　Plaquevent Julien　　Schmitt Cyril

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

All the members of the group are interested in arts such as music, cinema, and painting. As the project is technological, the first step is to find an art which can take advantage of the technology. Graffiti is chosen for two main reasons. First, all the members are interested in it. You can find it anywhere: street, train, buildings... The idea to try it and understand the process of making one seems fun. Graffiti is also a way to express yourself. Second, it really corresponds to the study program. It is clearly an interactive system that deals with vision and graphic.
In order to define the scope of the project it is important to define the word 'graffiti'.

1.2 Definition

The term 'graffiti' can be used for any images or lettering painted, marked or drawn on any surfaces[12]. According to this definition, it can be assumed that the first graffitis came from the prehistoric cave. One example is shown on figure 1.1. It is good to notice that the tools used to make those graffitis were basics and they have not stopped evolving. They were made with animal bones and some pigments.

Figure 1.1: Cave painting in Lascaux[1]

To do a bound in the history, it is possible to go directly to modern graffiti. It appears that this new art began in the 60's[13]. At the beginning, it was designated under several names: urban art, wall art, graffiti, graffiti art, and stencil art. Because modern graffiti was born in the streets of New-York City, people usually call it 'street art'. As previously mentioned in this section, first graffitis were made with really basic tools. Nowadays, street artists use a spray can to do graffiti. They have different spray cans to have different colors and sizes.

1.3 Aim of the project

The main goal of the project is to virtualize street art. To explain the meaning of virtualization in this context, a comparison with an other art (music) is relevant.
In 1887, Thomas Edison invented the cylinder phonograph[14]. It was the first device to record and play again sounds. This device has been upgraded to become the recorder. The point is that thanks to a recorder and a computer you do not need musical instruments to create a new song. So the process of creating a song has been virtualized.
To apply this method to street art several challenges have to be dealt with. The first one is to find a way to replace the spray can by a device. Because the system has to be accessible by as many people as possible, one solution is to use a mobile phone. Then, the graffiti has to be rendered on a surface. The use of the mobile phone as a spray can has to render a paint on a surface. This could be easily done by a projector. Then, a computer is needed to make the link between the mobile phone and the projector. Finally, it is important to decide how realistic the project should be.

The notion of virtualization has been explained. Why is virtualized street art interesting to be discussed? There are several reasons. First, it is important to remind that it is forbidden to paint on a public wall (at least in most countries). Indeed, you need a private wall to do it. Moreover, you can use it only once. The virtualization can solve both constraints: legality and place. Another issue is that graffitis are not eternal. Due to some environmental issues, graffitis will be deteriorated. This problem can be managed by virtualization.

The aim of the project is to develop a system which virtualizes the process of creating a street art.

1.4 Scenario

The background of the project and its aim have been explained. In order to facilitate the conception and the design of the project, a scenario has to be clearly defined. It is assumed that the system is installed in a bar. So, there is a computer, a projector and a wall dedicated to street art.

1. A customer enters the bar and sees the system.

2. The customer takes his/her mobilee phone (which replaces a spray can).

3. He starts to do a street art on the dedicated wall using the mobile phone as a spray can. The mobile phone is located in real time and when the button 'paint' is pushed a paint appears on the dedicated wall.

4. When the customer wants to stop, he/she can share the painting on a dedicated social network on the internet.
5. Finally, the wall is 'cleaned' and a new customer can use the system.

It is a simple scenario of use. Another scenario is if the customer does not finish his/her street art. In this case, the user can save it to finish it later. It also implies that the user can load his/her street art.

Thanks to the study of background, goal and scenario of the project, the main challenges of the system have been figured out. So it is time to start the preanalysis of the problem.

Chapter 2

Preanalysis

As the aim to produce virtual graffitis has already been treated in several projects, the following section exposes an overview of these former installations. They all use different processes and techniques but try to reach the same goal: to create in real time a virtual graffiti. The study of these systems is a way to notice achievements and possible improvements. The preanalysis of these existing systems is followed by a Low-Fi prototype performed according to observations made in this section.

2.1 Existing systems

2.1.1 MobiSpray

MobiSpray is an application which allows the user to do graffitis thanks to a simple mobile phone containing an accelerometer illustrated in figure 2.1. In the case of MobiSpray, the main aim is to paint on any surfaces [2].

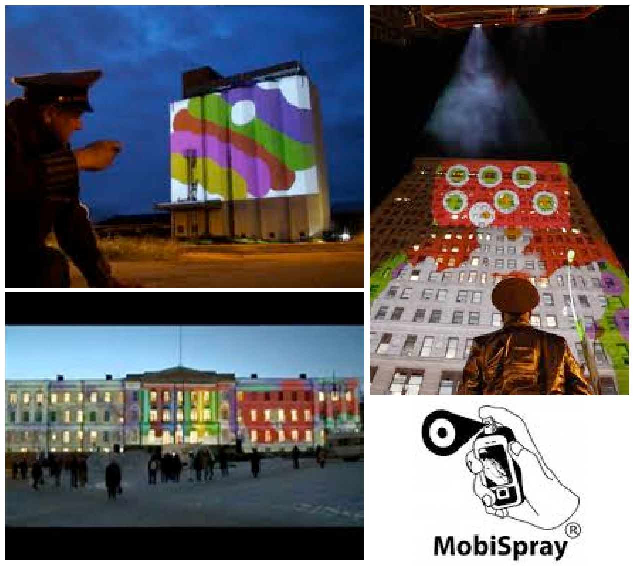


Figure 2.1: MobiSpray system[2]

The user interface shown in figure 2.2 is on the cellphone, using the keyboard of the phone to change the colors, intensity and other graphical options. The gesture is simple

: the user holds the mobile phone in the hand and paints the graffiti using the moves of his/her wrist like a remote control. Those two previous points make this system really easy-to-use by anyone. On the other side, the limited gesture and user interface proposed here can be discussed and considered as a possible improvement if the aim of the system is to simulate the process of graffiti painting. [15]. Indeed, a gesture more appropriate would simulate the action of painting with a spray can. The wrist could be used but the real gesture will include some moves with the entire arm. The question of the fingers can also be studied. The fact to press a button like a remote control could be replaced by an action closest to the way of pressing on a spray can.

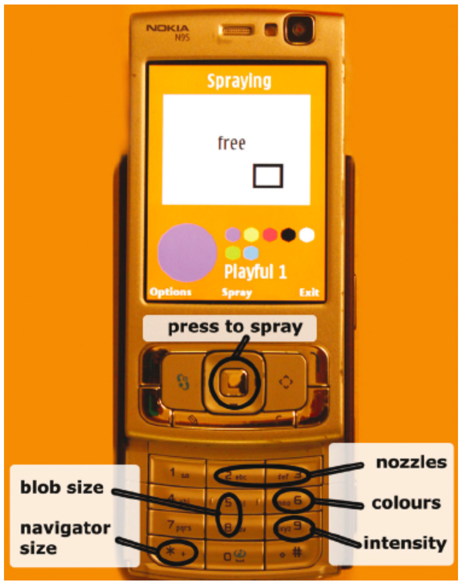


Figure 2.2: MobiSpray - User Interface[3]

The language used to code this project is Python, using some libraries such as Mobile Python, PyS60 and PyGame [2] in order to be compatible with mobile phones or to process the graphical rendering. These libraries are only compatible with some models of Nokia mobile phones [16]. The communication protocol used is UDP through a wifi network. The wireless technology gives the user more freedom performing the painting and fits the real time constraint [3]. This system also proposes an interesting feature which is the multi-user mode. MobiSpray is built for collaborative painting and 4 users maximum can paint at the same time. Concerning the rendering, the graphical aspect stays basic. There is no special texture or graffiti-like display, the system uses simple drawing functions that can be found in the software Paint for example. It is possible to use some stencils during the creation. The advantage of this tool is that a user with low skills in design can perform a satisfying graphical result, helped by the already shaped stencil.

2.1.2 Virtual Graffiti and WiiSpray

Other applications with the aim of doing virtual paintings exist[4]. Virtual Graffiti and WiiSpray are among them, illustrated by figure 2.3. This time, the principle is different. Some wiimote sensors are placed on a real spray can [5].

The development team used then a wiimote camera to receive x/y position and the distance between the user and the screen. This augmented wiimote is a good way to

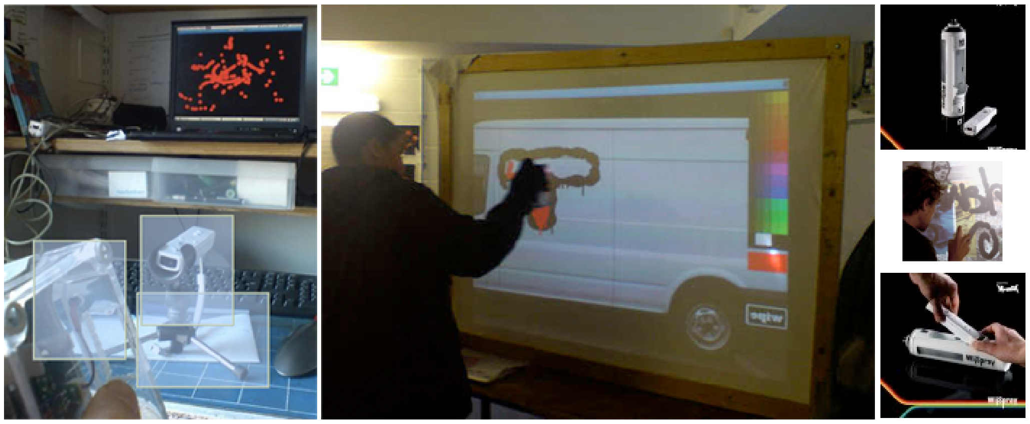


Figure 2.3: Virtual Graffiti and WIISpray systems[4][5]

simulate a real spray can but it implies that the user has a wiimote with him. It is thus possible to see the difficulty with the scenario of the project and to cunclude that few people are ready to go out with their wiimote. The communication protocol between the wiimote and the computer side application is here the Bluetooth. Bluetooth can be an appropriate solution if the computer based application and the spray can are in the same area [17]. Concerning the rendering, the development team used Processing [18]. The system is based on several libraries compatible with processing in order to link the software to the wiimote sensors. The equipment necessary is a computer, a wiimote camera, a wiimote and a projector. In the case of Virtual Grafiti, the user interface is very simple and the only function available is to chose a different color.

The user interface of WiiSpray shown in figure 2.4 proposes a more developed interface rewarded by an iF communication design award [19]. The interface is inspired by a plant where each flower will be responsible for a special function. For the choice of the color, each petal of the flower is a different shade. The user selects the color by spraying on the chosen petal. Another flower is dedicated to the stencils where, in the same process, each petal is a different stencil. The last flower is for other functions such as the size of the blobs, the choice of the brush or the style of the spray.

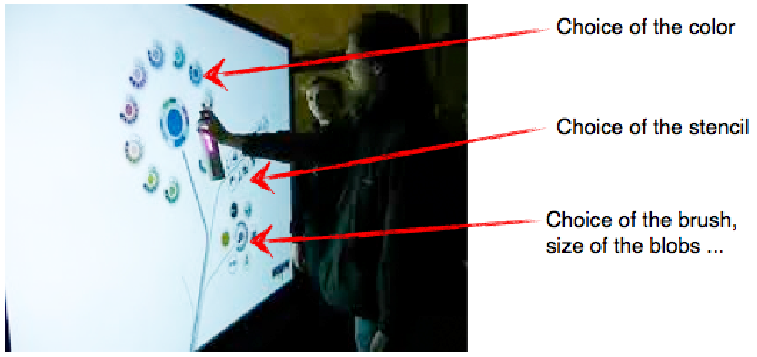


Figure 2.4: WIISpray - User Interface[5]

2.1.3 L.A.S.E.R

In this project, the aim is to paint with a laser on different surfaces, mainly buildings as can be seen on figure 2.5.

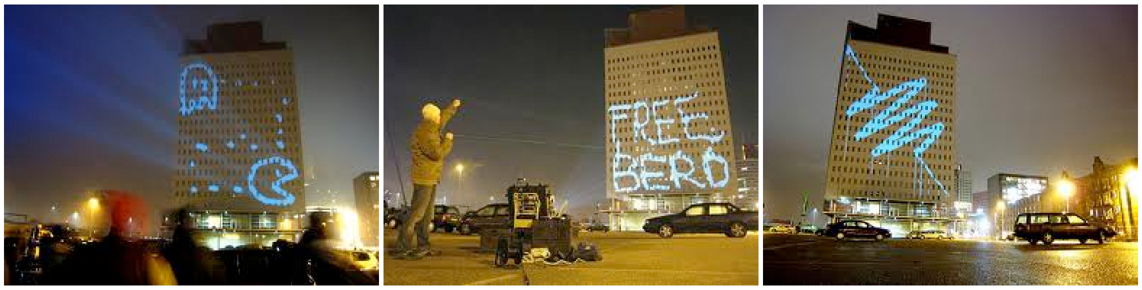


Figure 2.5: L.A.S.E.R system[6]

The software is coded in C++ and openFrameworks which is a library useful for computer vision [6]. Here the utilization is different because the user is very far from the drawing. There is no communication protocol as there is no client/server configuration. Only one computer side application is running to perform the computer vision analysis and the rendering. The gesture is different from the way to use a spray can because the laser is used like a pen. So the gesture is similar to hand writing. The equipment necessary to perform the painting is a computer, a laser, a camera and a projector.

2.1.4 Sum up of the existing systems

Table 2.1: Table of the existing systems

System	Advantages	Areas of improvements
MobiSpray	+ Uses a mobile Phone + Wireless + Stencils + Multi-user mode	- Compatibility - Non realistic rendering - Limited gesture - Limited UI - Phone dependent
WiiSpray	+ Wireless + User Interface + Stencils + Realistic gesture and rendering	- Wiimote dependant - Portability
Virtual Graffiti	+ Wireless + Realistic Gesture	- Wiimote dependant - Portability - Limited UI - Non realistic rendering
L.A.S.E.R	+ Internet independant	- No UI - Non realistic rendering - Non realistic gesture - Not really adapted to the scenario

Several useful points can be noticed from the previous survey of existing systems. Firstly, most of those systems use wireless communication protocol except L.A.S.E.R of course, which is more computer vision oriented. From a usability and gesture point of view, the more realistic projects are Virtual Graffiti and WiiSpray which use augmented real spray cans. Although these advantages, those projects are limited by the compatibility of their systems. They use a Wiimote which is not adapted to the scenario of the project. MobiSpray uses a mobile phone but the compatibility of the system is limited to certain phones of certain brands.

From the previous observations, a session of low-fi tests can be performed. It is the occasion to investigate different leads of improvements found in the existing systems.

2.2 Low-Fi testing

2.2.1 Purpose

The first point to investigate is: How will the user hold the smartphone? The second field of experiment is to size up the interest of the participants concerning particular functions and the possibilities to apply them to the scenario. This information allows the development team to have an idea of the complexity of the future user interface. Finally and from an engineer point of view, the fact to perform this kind of tests at the beginning of the project is a solution to have an iterative development cycle. This low-Fi test is a way to check the consistency of the concept and it will help to concentrate on a user centered design[20].

2.2.2 Experimental conditions and process

Table 2.2 describes the conditions of the experiment :

Table 2.2: Low-Fi Experimental Conditions

Place	Dark room
Equipement	Flat screen Mini-projector Ipod Touch Computer
Participants	Number : 4 Sex: 3 males and 1 female Age: from 21 to 25 Painting skills : none
Duration	45min

For this first test, a group of 4 volunteers is present. Due to this reduced number, a Within-Subject test design [20] [p 75] is performed. This test is divided in several tasks. Instead of having several groups of people testing one task, the four participants will test all the tasks once. The test starts with an introduction speech and a presentation by the development team where they explain the scenario and the general aspects of the project to the participants. The different tasks performed during the tests are:

The survey The first task of the tests is to fill a survey available in appendix A. The survey contains different categories of questions. Some general questions about the user are necessary to keep in touch. Then general questions about the project are asked in order to know the participants' interest about this concept. The survey contains finally some specific question in relation with the next tasks. The duration of this task is five minutes.

Usability test on the mobile application For the second phase, the participants have to face the usability test concerning the mobile application. Each participants, independently of the others tries to take spontaneously the iTouch as a spray can. The background screen of the iPod Touch is an image of a spray can, as shown in figure 2.6. The participants are asked what would be the most natural position that they would adopt. Fifteen minutes are necessary for this task.

Painting simulation - Gesture observation For the third task, the installation described in figure 2.6 is used. The participants are placed, facing the screen, stand up. An image of a graffiti is displayed on a wall using a projector. The participants have to imagine they are reproducing the graffiti using the iPod Touch. The duration of this task is fifteen minutes.

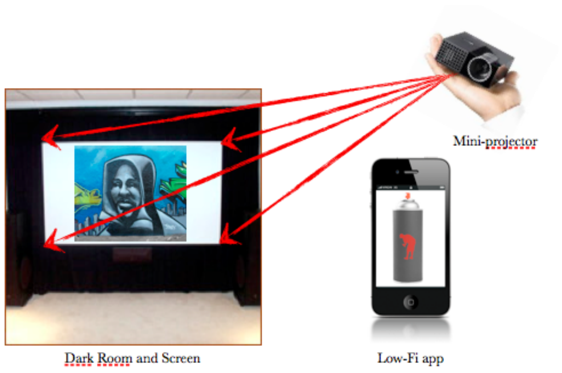


Figure 2.6: Low-Fi Installation

Multi-user mode Finally, the last point to observe is how the participants are interacting with the multi-user mode. The previous task is conducted again but with several participants illustrated by figure 2.7. This task is performed with two and three participants during ten minutes.

Figure 2.7: Low-Fi Testing

2.2.3 Results

The first point observed is that the participants were interested and motivated by the project. In the case of the scenario, they suggested that they definitely want to try a bar with such an installation. From the Usability test on the mobile device, 4 main position propositions are the result of the question: How would you take the device in your hand?

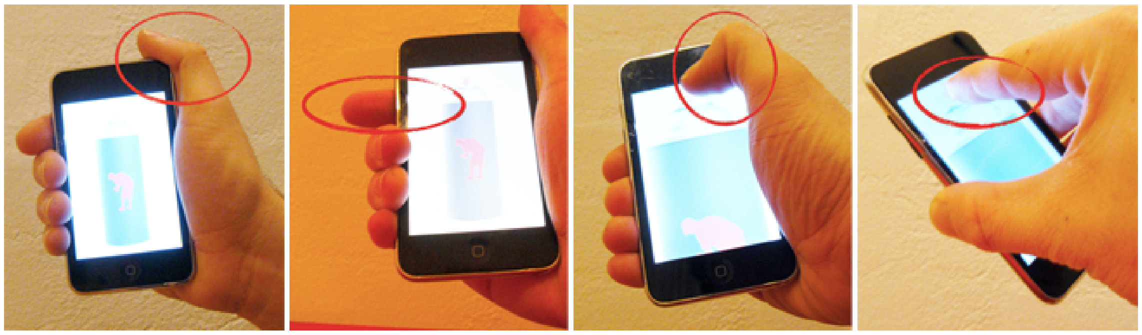


Figure 2.8: Mobile App : Usability results

As can be seen on figure 2.8, there are two main ways to hold the device. The first proposition is to use buttons of the device to interact with the installation (Volume or On/Off buttons). The second way is to use the screen. In all cases, the users preferred performing the action with the thumb and rather at the top of the device. In the case where the user interface would not use physical button, the participants made several propositions to manage functions like the change of color or size of the spray. Some of them liked the idea of sliders on the screen although others preferred an interface showing a spinning wheel that you can turn with a finger. They suggested also the possibility to interact or to action some functions with the rotation of the device. They thought that multi-touch is not necessary. The main reason is that the user already uses 4 fingers to take the device in the hand.

Concerning the rendering, all the participants desired a system as close to reality as possible. The system should have the possibility to revert the last action. It should be also possible to save a color or some settings for the user. They were quite interested in the idea of a multi-user mode. This one should be realistic as well, meaning that a painting should cover the former one, without a melt of color. While testing the multi-user mode, the participants were painting sometimes one after the other and not in the same time. They enjoyed the idea of using stencils to paint and the possibility to create or load your own one.

The participants were also enthusiastic about additional features such as the dedicated social network to share the paintings. The idea of a speech recognition module had not unanimous support. Some participants liked the idea of painting and controlling everything with the voice but others thought it would be disturbing to talk alone performing the painting. In all cases, the scenario takes place in a noisy environment and makes this function difficult to apply. The realistic aspect wished for this project could also be affected by this module. Indeed, few taggers talk to their spray can.

2.2.4 Possible functionalities

Regarding the results of the low fidelity test, table 2.3 lists each possible functionality of the system and exposes real behavior and possible solutions. These solutions will be discussed in the design section of this report. These possible solution ideas come from :

- (E)xisting systems (from preanalysis) functionalities
- (N)ew ideas from the project team
- Low Fidelity test (P)articipants

Table 2.3: Functionalities listing (E, N and P standing for the items of the preceding list)

Functionality	**Reality**	**Possible augmentations**
Change color	Take another spray can	N- Use voice to select a color P- Use a slider to select a color E- Use a wheel to select a color P- Pick a saved color
Change brush size	Change distance between the spray and the wall	N- Use voice to change the size P- Use a slider to change the size N- Use a wheel to change the size N- Pick a saved size
Change brush shape	Impossible	N- Use voice to select a shape E- Use a wheel to select a shape N- Pick a saved shape
Draw a specific shape	Use a stencil	N- Use voice to select a stencil E- Use a wheel to select a stencil P- Pick a saved stencil
Mix the paint	Shake the can before use	E- Not implemented N- Shake the device before use
	Paint sags if paint not mixed	E- Not implemented N- Paint sags if paint not mixed
Paint	Press the button on top	E- Push a hardware button P- Push a software button
Cover existing paint	Cover with no mixing	E- Cover with no mixing N- Mix the colors
Deal with can running dry	Take a new can	E- Infinite spray can N- Refill the can
Undo/redo actions	Impossible Paint over	N- Use voice to undo/redo N- Use a software button
Share the painting	Painting is only in one place	N- Share it on internet E- Project it in a public place
Reset the painting	Take a new canvas/wall	N- Use voice to reset canvas E- Use software button to reset E- Create a new canvas

Chapter 3

Analysis

The study of the existing systems introduced several projects with the same goal: to create virtual graffiti. These previous systems use different technologies, different ways to imitate the spray can. This section will present some technical possibilities to realize such a system. The goal is here to chose the technical solutions that fit the most with the current scenario.

3.1 Use cases and work flow

As a reminder for the analysis part, figure 3.1 illustrates some basic use cases ordered according to the scenario previously introduced.

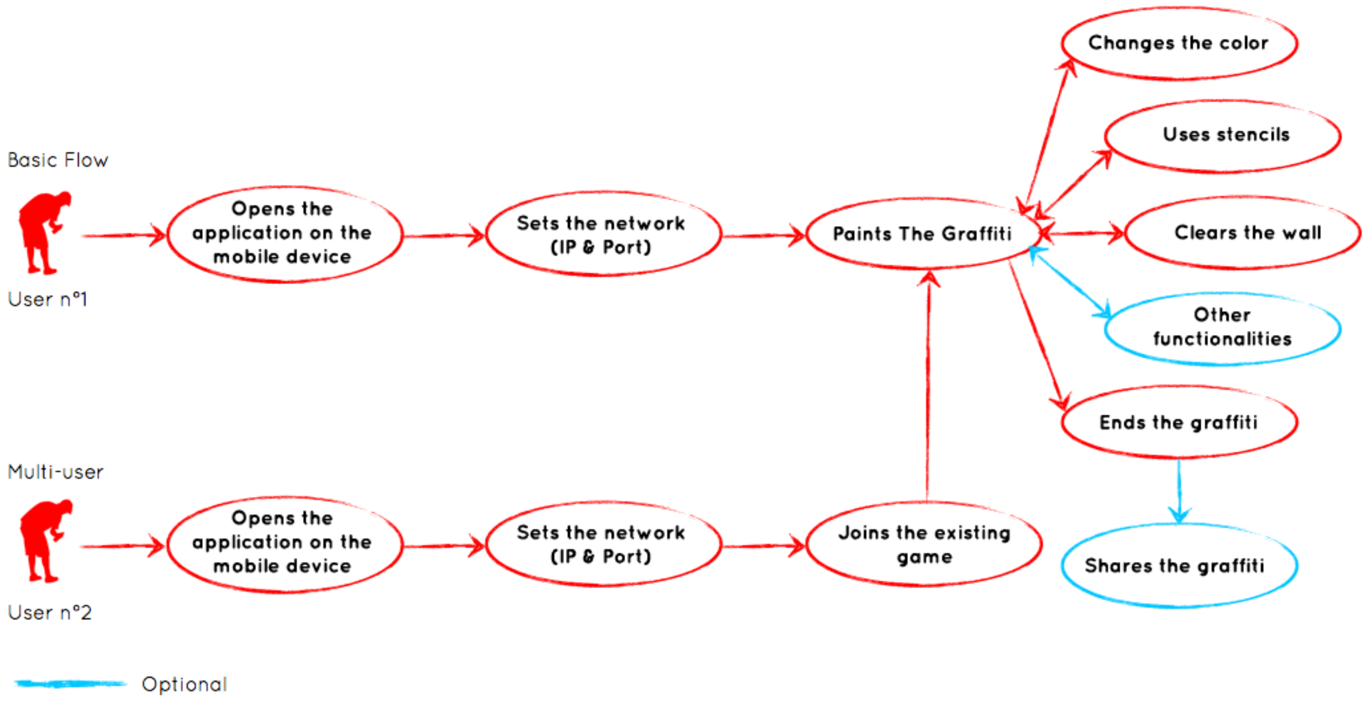


Figure 3.1: Use cases and work flow

3.2 Device movements retrieval and 2D projection computation

An important issue of the project is to detect and analyze the movement of the device which is used as a spray can. It has to be accurate enough to make the system realistic. Thus, the position of the device should be retrieved in real time. This task can be executed thanks to an accelerometer. In order to explain how, it is important to study what is an accelerometer. Some physics computations are then required and finally, it is important to point out the limits of this method.

3.2.1 Accelerometer

An accelerometer is an electromechanical device which is used to calculate accelerations [21]. The acceleration can be due to several forces. Passive forces like the gravity can be taken as an example. The movement or the vibration of the accelerometer can also be measured, in this case it is a dynamic force.
The calculation of passive forces is useful to obtain the angle of the device or the way it is hold which is not really the purpose of the system. Indeed, it focuses more on the dynamic forces caused by the movements of the device. A 3-axes accelerometer is shown in figure 3.2.

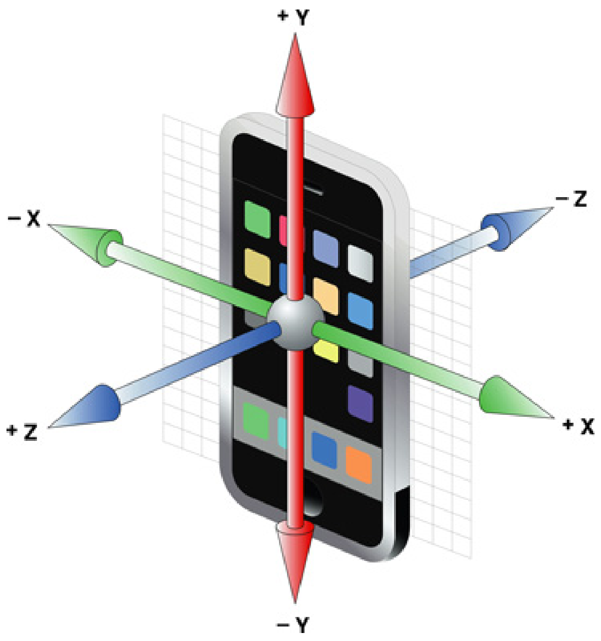


Figure 3.2: Accelerometer's axes on Iphone[7]

From the accelerations retrieved from the accelerometer, the system will have to compute a projection of the position of the device in space to the canvas plan.

3.2.2 Physics of movement

The movement of an object can be characterized by three data: position - velocity - acceleration. Those three values are linked by the simple formulas [22]:

$$a = \frac{dv}{dt} \qquad \& \qquad v = \frac{dx}{dt} \qquad \Rightarrow \qquad a = \frac{d^2x}{dt^2}$$

a = acceleration
v = velocity
x = position

The acceleration is simply the second derivative of the position. So, if you have the accelerations you can compute the position by calculating two integrals.

3.2.3 Limits

As previously mentioned, it is possible to get the position from the accelerations. But, this method needs to know the initial conditions to work. So it is important to initialize the process by knowing the position of the device at the beginning.

Another consideration is that the gravity is always applied to the accelerometer. In fact, it is important to configure a 'no movement' state. The value of the accelerations when the device is static has to be subtracted from the accelerations when the device is moving.

Finally, a digital filter has to be applied [23]. The main reason is that the signal from the accelerometer is not noise free. Several algorithms can be used to filter the signal, one of them is a moving average [24]. The principle of this method is to take several samples and calculate the average value. This algorithm is used as a preprocessing treatment.

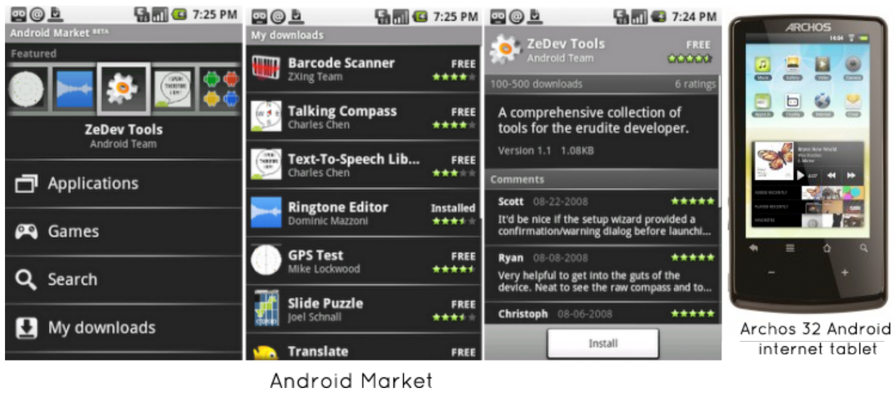
Thanks to the accelerometer the position detection issue can be solved. It is now important to find a device which contains an accelerometer to be used as a spray can.

3.3 Phone

It is important that the system can be used by the costumers painting with their own virtual spray can. The mobile phone seems to be the most appropriate tool for the current scenario. A cell phone is personal and most people bring their mobile phone to go out. The next step is to find out what kind of cell phone will be targeted for the development. A phone with several embedded equipments and an appropriate solution to distribute easily the mobile application is needed. Thus, the customer of this scenario should be able to download the application on his or her cell phone everywhere. That is why it is preferable to target smartphones which have all the required tools. This idea is in adequacy to the fact that people tend to change their simple phone for a more complex one, according to [25]. The following analysis will focus on two operating systems which could be two viable solutions for our project: Android OS and iOS.

3.3.1 Android OS

Android OS is Google's open source operating system. Many smartphones are equipped with Android since 2007. Android OS allows developers to create their own application via the Android SDK. The main advantage of Android is that the development language is Java. Every member of the group has advanced skills in Java programming. Many java libraries are compatible with the android SDK[26]. Most phones equipped with Android possess an accelerometer. The Android Market, illustrated by figure 3.3 proposes to Android's users to download applications easily from a computer or from an Android phone. Android OS cannot see and connect to ad-hoc networks. In other terms, the mobile phone is not able to connect to a customized network created by a computer[27][p2]. It can be a disadvantage for the current scenario. If no wireless internet access is available in the bar, and if the server is connected with a wire, the establishment of an ad-hoc network is not possible between the mobile and computer side application. Android OS is also the operating system of some Archos internet tablets since 2010 as shown in figure 3.3.



A comprehensive collection of tools for the erudite developer.

It'd be nice if the setup wizard provided a confirmation/warning dialog before launchi...

Very helpful to get into the guts of the device. Neat to see the raw compass and to...

Figure 3.3: Android OS

3.3.2 iOS

iOS is the operating system of Apple's mobile devices: iPad, iPod Touch and iPhone illustrated in figure 3.4. It is not open source. iOS programming language is Objective C, based on the C and C++ languages. Objective C is an object oriented language used mainly with the Cocoa framework, created by Apple to build Apple applications [28]. The brand was the first to introduce a platform to buy or download mobile applications via it's Application Store. Every device running with iOS is equipped with an accelerometer. iOS devices can connect to an ad-hoc network. iOS can also deal with mutli-touch. The main disadvantages of using iOS are for the developers. In order to test the code on a real device, a special license is needed. This license is not free. One member of the project team already owns an Apple developer account and can provide easily a new license for the group. Objective C programming for iOS is only possible if the developer owns a Mac and the XCode development environment. Two members of the group possess a MacBook. Although these points, using Objective C is a nice challenge for the team. The members of the group are interested in iPhone and iPod and are motivated to learn the process to create applications for these devices.

Figure 3.4: Apple mobile devices

3.4 Communication protocol

If the mobile application is responsible for collecting and sending data from the accelerometer, the project needs a computer side application to receive and transform the data into a virtual painting. A link between those two elements is necessary. The choice to use a mobile device as a spray can implies a wireless communication protocol between the device application and the computer side application. The following analysis section will focus on some wireless communication protocols which are managed by most of mobile phones.

3.4.1 Bluetooth

Bluetooth is WPAN (Wireless Personal Area Networks) technology [29][p2]. In other words, Bluetooth networks are short range wireless networks created to link and connect different devices together such as printers, computers, keyboards, cell phones, cameras. For the mobile phones, bluetooth replaced the infrared protocol to exchange data. The main advantage compared to infrared is that bluetooth devices do not need a line of sight to communicate [30][p378]. Bluetooth devices are divided into three classes according to the operating range and power of the signal. Table 3.1 describes the three bluetooth classes.

Table 3.1: Bluetooth power classes properties [29]

Type	Power	Operating range
I	100mW	Up to 91 meters
II	2.5mW	Up to 10 meters
III	1mW	Up to 1 meter

iPhones and iPod Touch belong to the bluetooth class two. Their operating range is about 10 meters which is an appropriate distance for the current scenario. One of the advantages of bluetooth networks is that you do not need any internet connection to establish the link between the devices. This protocol allows a data flow up to 1Mbps. In optimal conditions, it means 1600 messages delivered per second which can be enough for real time[29].

3.4.2 OSC

The Open Sound Control protocol was originally created as an alternative to the midi format mainly used with electronic instruments. It uses the UDP protocol to send (or receive) formatted messages to addressed hosts of the network through a special port [31]. A network connection is needed. According to the OSC specification [32], the protocol allows a time precision of 200 picoseconds and thus allows a real time communication. The performances and the accuracy depend on the bandwidth of the connection. The main advantage is here the syntax and the format of the OSC messages. One message can encapsulate several variables of any type which can be practical for example to send x, y and z positions. It is possible to work with the OSC protocol and the Objective-C language. Some libraries such as vvosc allow developers to create applications which receive and send OSC messages [33]. As the OSC protocol uses a network, information can be sent via the wifi. For the current scenario, the bar has to possess a wifi network or a computer able to share an ad hoc wireless connection. Here is the main disadvantage, the mobile device needs some configuration to communicate properly with the computer side application on a given network.

3.5 Rendering on the computer

An important part of the project is the graphical render of the painting. After the computation of the coordinates of user's movements projected on the canvas' 2D plan, the computer-side application has to print a realistic representation of the paint. The other systems studied in the pre-analysis do not use any real paint physical model, they just draw perfect points with no paint leaks and without any specific texture nor light [34]. Only WiiSpray offers a spray opacity management depending on the time the spray stays at the same coordinates. Since the pattern of the physical behavior of a liquid is a tough subject, this study will be restrained to :

- paint leaks which happen when the user projects too much paint at the same place or when he does not shake the can before using it
- paint mix or coverage depending on wetness
- paint density and opacity depending on the time the spray stays at the same place

There are different strategies to address these problems and to model the behavior of a liquid. First, a physical model can be used to compute the newtonian equations of the liquid from scratch. Second, existing software already implementing fluid simulation are also available. And third, the knowledge accumulated from real world experiments can help creating a simple model.

3.5.1 Newtonian liquids physical model

This approach is the most complicated one since it consists in computing the Newtonian equations of the paint particles from scratch. It's efficiency in terms of computational load is highly dependent on the way the model is implemented and optimized. It is based on Newtonian fluids mechanics. As an example, to deal with paint leaks, the forces applied on the paint particles by the support could be computed using the simple formula $\tau = \mu \frac{\partial u}{\partial y}$ with :

- τ ($Pa = N/m^2$) the stress exerted by the support to the fluid on a parallel direction
- μ ($Pa.s = kg/(s.m)$) the viscosity, a constant characteristic of the newtonian fluid
- $\frac{\partial u}{\partial y}$ (s^{-1}) the gradient of the velocity of the fluid on a perpendicular direction

The more complex Navier-Stokes formulas [9] which include pressure forces could also be used. This approach is used for aircraft design and very sharp applications. This project should work in real time and in the very specific context of spray painting with a viscosity-set fluid on a vertical wall. This approach is too heavy to be interesting in this case.

3.5.2 Existing fluid simulation softwares

Existing softwares already deal with this problem : 3D simulations like Blender [35] or Glu3d [36] for 3D Studio Max both implement fluid simulation modules. Blender is a good solution since it is free and open source but it can be used for any kind of fluids and has too many parameters to set to get a realistic paint simulation.

Some solutions also use the power of computer's graphics processing units -which is higher than central processing units one- to compute this kind of models in an efficient way. This is called General-purpose computing on graphics processing units (GPGPU) and ACUSIM's AcuSolve Computational Fluid Dynamics (CFD) flow solver is one example of this strategy [37]. Unfortunately, this solution is hard to deploy and is also too generic for us.

There are also more paint specific ones like IMPaSTo [8] which is a very interesting specific model for brush painting but it cannot be used in this project because of the use of a spray can instead of a brush. It takes into account the complex interactions between the canvas, the brush and the paint itself. As an example, the wetness of the paint and the moves of the brush are included in the computations as shown on figure 3.5.

Figure 3.5: Example of painting using IMPaSTo [8]

Finally, there is a solution which seems suited for this project's application : Processing [18], a java based programming language designed for interactive systems production. The communication with the mobile application would be implemented easily with the OSCP5 library and the graphical rendering of the painting could benefit from the numerous graphics and interaction libraries it offers. A simple model based on Jos Stam's Navier-Stokes solver [9] allows to compute basic fluid behaviors fast enough for the system. It would need to be adapted because it is not paint specific but could still constitute a good start for a solution. Figure 3.6 shows an example of this solver. It is also possible to use the powerful Java OpenGL and OpenCL libraries in order to get a more beautiful render (using resources from both CPU and GPU) as shown on figure 3.7 [10].

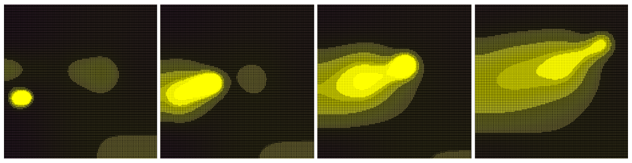


Figure 3.6: Fluid dynamics under Processing using Navier-Stokes solver [9]

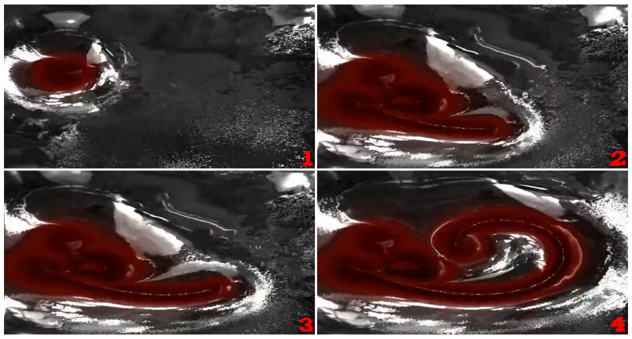


Figure 3.7: Graffiti Viscosity project using vitamin library for Processing [10]

3.5.3 Specific model from experiments

In order to make a realistic model adapted to the specific use of this project, real spray painting tests were performed. The results will help adapting the Processing model and optimizing it by implementing only the observed behaviors. The shapes drawn on figure 3.8 are the most common one the user could think about :

- 4 red dots on top, made with the same can but from different distances
- 2 black horizontal lines made at different speeds
- 1 red vertical line showing wet paint melting
- 1 red dot with paint leak zoom on the right picture

The 4 red dots are the most basic shapes which can be drawn. There is already a specific behavior to observe : paint leaks happen when the spray is too close of the canvas. The third dot, which looks better and has no leak, has been made from a distance of 20cm. The two black lines are the second step in user's first moves. It shows how the speed of the movement influences the opacity of the paint and can cause paint leaks : the first line was made slowly while the second one was drawn fast. The red vertical line allowed to determine that different layers of paint should mix together if painted while still wet and

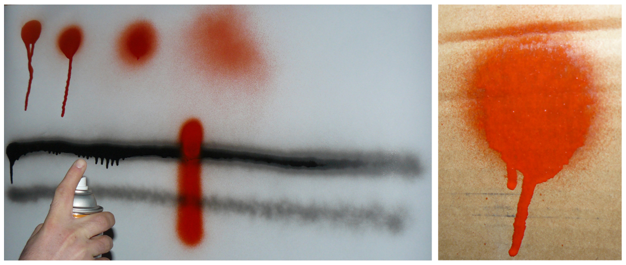


Figure 3.8: Real spray paint tests

cover each other if the former ones are dry. The red dot on the right demonstrates that paint leaks also occur when the can is not shaken properly before being used. In this case the paint is more liquid.

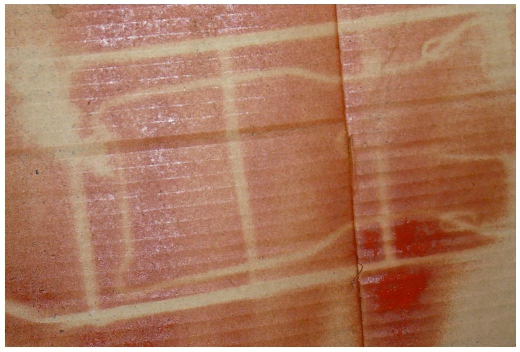


Figure 3.9: Real spray paint stencil use

Figure 3.9 shows an attempt at using a stencil. The stripes created by the stencil are clearly delimited and there is no special effect on the borders.

Finally, figure 3.10 illustrates the first real multiuser graffiti test. From this experiment, it is possible to notice that it is hard to work together without interfering. The two spray paint cones (represented in black and red on the figure) can easily mix due to the small work space. The green area shows an example of this kind of interference : the two paints are mixing a bit but the result is not very impacted. This is due to the collision occurring between the particles which make them lose most of their speed. The paint which reaches the canvas is mostly composed by the spray which was targeted on this specific area. Painting each other's hand is also a risk in multiuser action.

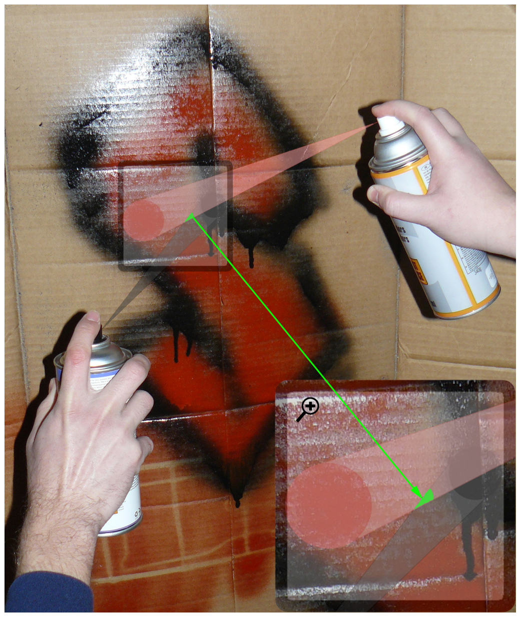


Figure 3.10: Real spray paint multiuser test

Chapter 4

Design

The previous chapter demonstrates that several technologies are needed to build a system of virtual graffiti according to the scenario. The design chapter will present the architecture of the different modules: the mobile and computer-side applications. It will also introduce the different steps of the elaboration of the User Interface.

4.1 Global architecture

Figure 4.1 introduces the global architecture of the system. The first module is the mobile application. The chosen device is the iPhone and the language of development is Objective-C. The mobile application is responsible for transforming the accelerations from the accelerometer into 2d positions. It sends to the computer-side application these coordinates. This second module is the computer side application. It performs the rendering according to the data received from the mobile application.

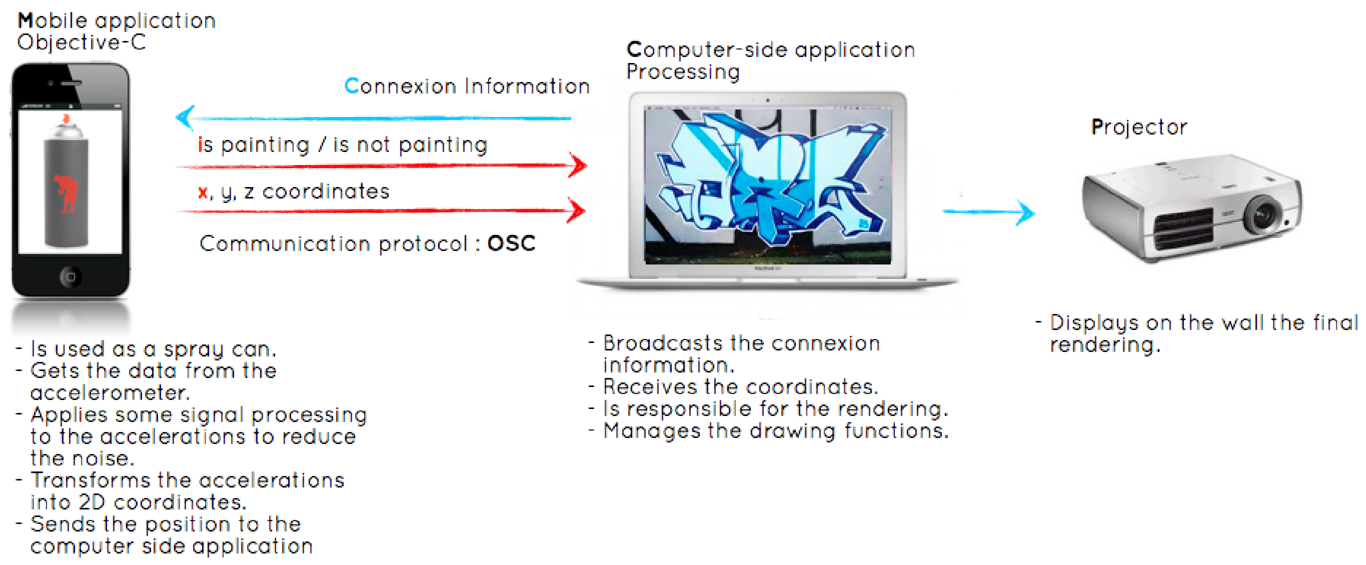

Mobile application
Objective-C
Connexion Information
is painting / is not painting
x, y, z coordinates
Communication protocol : OSC
Computer-side application
Processing
Projector
- Is used as a spray can.
- Gets the data from the accelerometer.
- Applies some signal processing to the accelerations to reduce the noise.
- Transforms the accelerations into 2D coordinates.
- Sends the position to the computer side application
- Broadcasts the connexion information.
- Receives the coordinates.
- Is responsible for the rendering.
- Manages the drawing functions.
- Displays on the wall the final rendering.

Figure 4.1: Global architecture of the system

4.2 Selected functionalities

Table 2.3 page 24 shows a list of possible functionalities for the system extracted from the pre-analysis. After the analysis of the project, table 4.1 was drawn listing each feature and how it will be implemented in the actual system. The selection of these solutions have been made to make the system as close as possible of the real world painting conditions except for the possibility to share the painting on the internet. The size of the brush will not be dealt with because it implies working on the "z" axis and would make the system much more complicated for a minor improvement.

Table 4.1: Selected functionalities listing

Functionality	**Reality**	**Selected solution**
Change color	Take another spray can	Pick another spray can in the "bag" menu
Change brush size	Change distance between the spray and the wall	Not implemented because of technological issues
Change brush shape	Impossible	Not implemented (realism)
Draw a specific shape	Use a stencil	Pick a stencil from the "bag>stencils" submenu
Mix the paint	Shake the can before use Paint sags if paint not mixed	Shake the device before use Paint sags if paint not mixed
Paint	Press the button on top	Press a software button on top
Cover existing paint	Cover with no mixing	Cover with no mixing
Deal with can running dry	Take a new can	Take a new can in the "bag"
Undo/redo actions	Impossible	Not implemented (realism)
Share the painting	Painting is only in one place	Share it on internet (augmentation)
Reset the painting	Take a new canvas/wall	Use the paint pot from the "bag" menu to reset the wall

4.3 Mobile application

From the selected functions and the global architecture of the system, the design of the mobile application can begin. To do so, the mobile application has to be divided into several modules responsible for the functionalities selected. To keep in touch with the User Centered Design, tests are performed. This section will introduce those steps and present the elaboration of the Graphical User Interface.

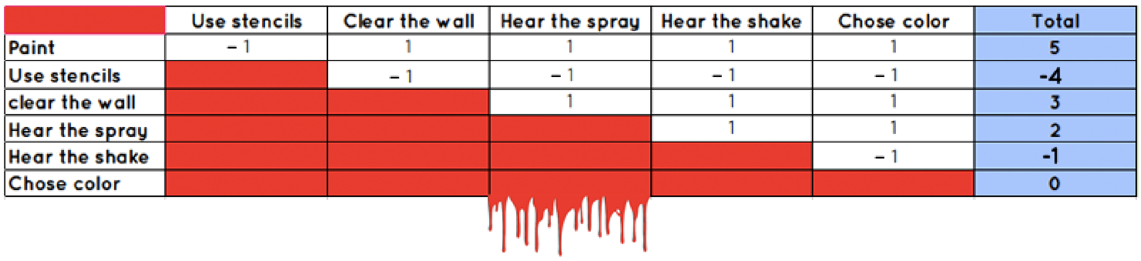
4.3.1 Value Analysis

In order to focus the future implementation on user's needs, a value analysis is performed [38]. This analysis gives an idea of the essential functions of the system from a user's point of view. It helps the developers to save time and energy during the software implementation, targeting the essential features. This information is also very useful to create the Graphical User Interface. This value analysis focuses on the main features of the mobile application. This sublist of features represents the basic actions which can be done using a spray can.

Table 4.2: Value Analysis conditions

Place	Laboratory
Equipement	Result grid
Participants	Number: 6 Sex: 3 males and 3 females Age: from 21 to 25
Duration	10min

The aim is to compare and determine the rank of importance of a feature according to the user's point of view. For this, a table shown in figure 4.2 has been created. The participants compare each row with the columns giving a mark from -1 to 1. -1 means the feature in the row is less important than the feature in the column. 0 implies both features have the same importance in the system. Finally, a 1 is given if the feature compared in the row is more important than the feature in the column. The sum of the user's results will give the final rank where the more important feature is the one with the highest score.



	Use stencils	Clear the wall	Hear the spray	Hear the shake	Chose color	Total
Paint	− 1	1	1	1	1	5
Use stencils		− 1	− 1	− 1	− 1	-4
clear the wall			1	1	1	3
Hear the spray				1	1	2
Hear the shake					− 1	-1
Chose color						0

Figure 4.2: Value analysis

Results The value analysis shows that it is important for the user that the system keeps the realistic properties of a spray can. Participants are attached to the basic functions and behavior of a spray can as shown in figure 4.3. The most important feature pointed by this analysis is to be able to paint. The second one is to hear the noise during the spraying action.

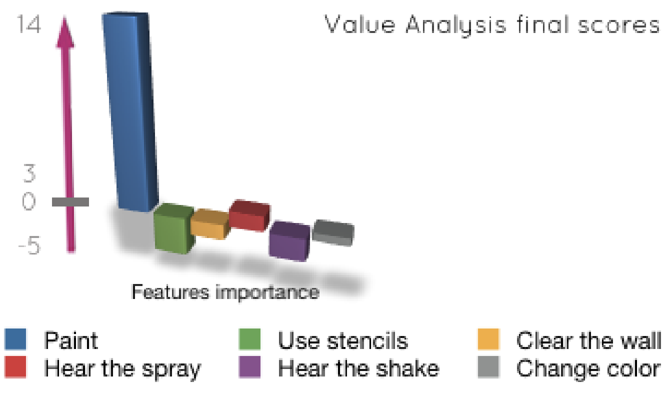


Figure 4.3: Value analysis - Results

4.3.2 GUI sketch

From the data collected in the previous tests described in section 2.2 and from the previous value analysis seen in subsection 4.3.1, it was possible to elaborate a first user interface sketch. This first graphical attempt is composed of four screens. The interactions between those screens are illustrated in figure 4.4. The circles show the action needed to reach the next screen. The change from one screen to another is depicted by the arrows. The area which is responsible for the beginning of the spraying is placed at the top of the screen according to the observations made in section 2.2. To respect the constraints of realism, the concept of the artist's bag is here introduced. This virtual bag shown on figure 4.4 second screen, contains a limited number of spray cans with defined colors, a folder with stencils and the possibility to clear the wall. The stencils' screen is reachable touching the stencils' folder. The stencils' screen is composed of a limited number of stencils. The user can thus select a stencil and use it with a specific orientation.

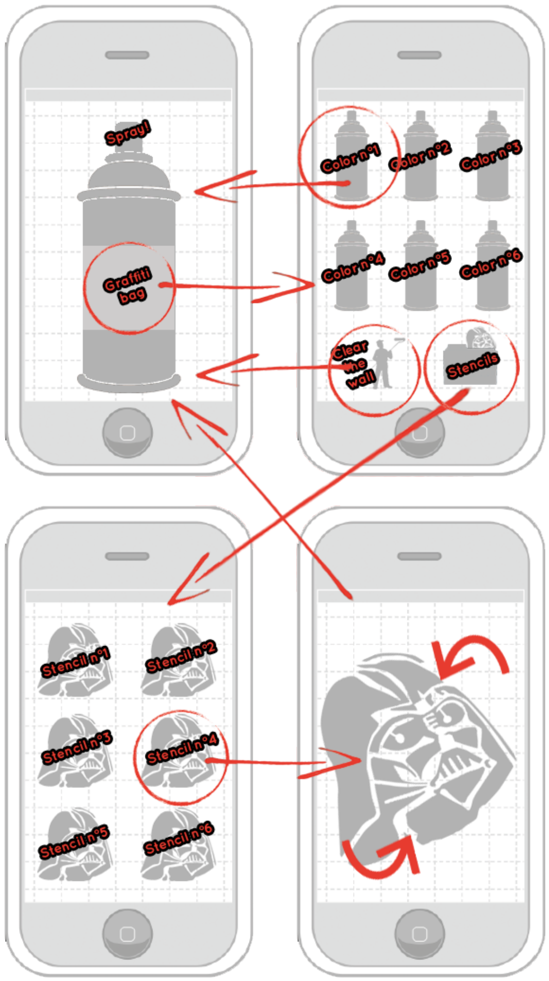


Figure 4.4: GUI sketch

The next step of the elaboration of the graphical interface is to bring this sketch to life via a low-fi prototype.

4.3.3 GUI Low-Fi Prototype

The low-fi prototype, illustrated in figure 4.5, is a version of the GUI sketch made with pencils and paper. It contains all the elements needed to reproduce the sketch but also some new components to correct and even improve the proposed interface. It is composed by a device, a spray can, some spray can labels, some spray can icons, some stencils and also some additional buttons.

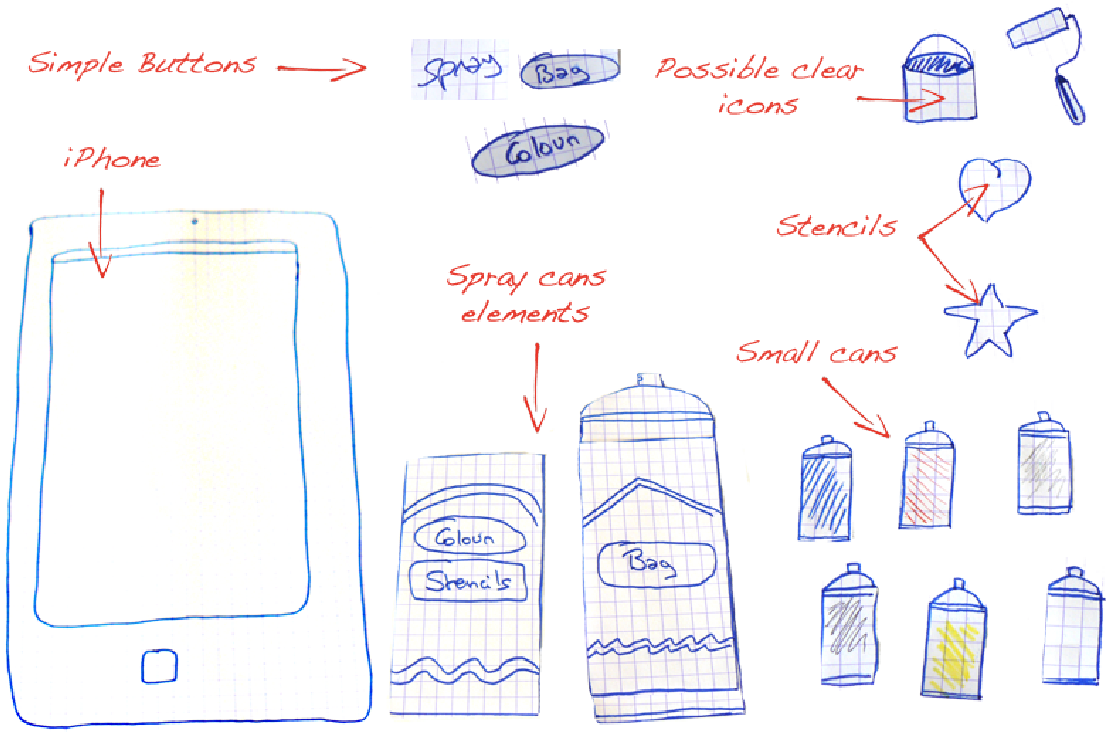


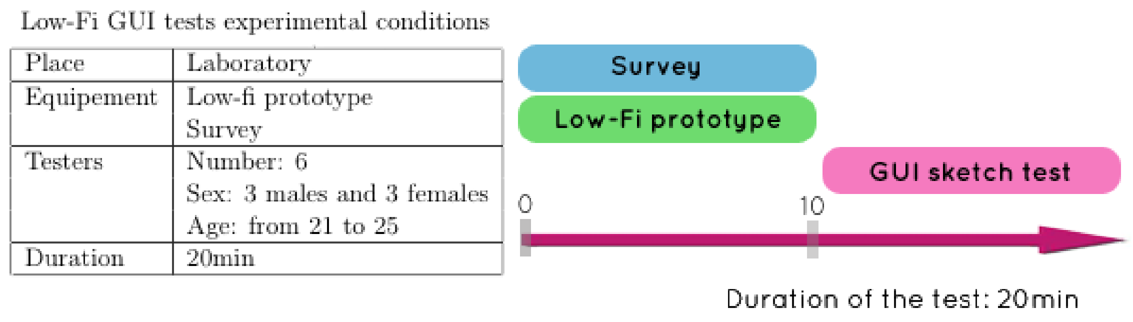
Figure 4.5: Low-Fi Prototypes elements

The aim of this low-fi prototype is to submit it to the participants in order to approve, correct or improve the GUI sketch. As every elements are removable, the participants will be able to transform the proposed interface according to their feelings and needs. This is the purpose of the following subsection.

4.3.4 GUI Low-Fi Test

To keep on working on a User Centered Design, the project group decided to process the test concerning the GUI. This test is performed as a workshop. The aim is here to check the validity of the GUI proposal and to observe the participant's behavior with the low-fi prototype. Is the succession of views logical? It is also the occasion to test the level of realism. Does it interfere with the participant's feelings and with the usability? Figure 4.6 describes the experimental conditions of the test and the course of the test:

The test started with an introduction of the project made by the project group. Then this series of tasks was performed:



Low-Fi GUI tests experimental conditions

Place	Laboratory
Equipement	Low-fi prototype Survey
Testers	Number: 6 Sex: 3 males and 3 females Age: from 21 to 25
Duration	20min

Figure 4.6: GUI Low-Fi Test

Survey The survey, available in appendix B, starts with some personal questions about the user. The following questions are specific to the Graphical User Interface. The aim of this survey is to check if the sketch presented in subsection 4.3.2 is relevant. Another essential point of the survey is to detect the level of realistic factors and to see if this realism is the best option to design the application. The participants can use the low-fi prototype to give more accurate answers.

Low-Fi prototype - GUI Sketch The low-fi prototype test is conducted in parallel with the survey as a workshop. There are two steps in this task. Firstly the participants can move the different low-fi elements to create their ideal interface. They can be inspired by the questions of the survey. They have to simulate all the screens and their logical transitions. They also have to mention what kind of gesture has to be done to start and stop an action. For the second task, the project group reproduces the GUI sketch and presents it to the participants. The participants are asked to use this interface and to express their feelings about it. The participants can move the low-fi elements at anytime in order to bring some ideas or some improvements.

During this experiment, the project group members are the observers. They have to note every propositions made from the participants. They also have to pay attention to the participant's behavior and write down the possible gesture mistakes caused by the User Interface.

Results - Survey The survey stressed out some interesting points. Figure 4.7 illustrates the result of the survey. First, the participants are 66% to enjoy having a graphical design evoking a spray can instead of having just a blank screen with buttons. 84% of the participants would prefer to manage the drawing functions on the phone rather than on the wall, arguing that the tools are always near to the painter and that would be more realistic. The way to access these tools cannot be decided according to the results. The participants had all different opinions about this subject even if the option of embedded buttons on the spray can seems to be the preferred one.

Concerning the colors, all the participants agree with the fact that it would be more realistic to have a limited number of colors although 84% of them think they would enjoy having an unlimited choice of colors. In the case of a limited range of colors, an average of

6,8 colors should be enough to paint an honorable graffiti, according to the participants. This number is approximately the same for the number of stencils.

About the painting wall, 50% of the participants would prefer to have a blank wall. The other half of the participants would like to have a background image reminding a wall or a painting surface. They suggested this idea to increase the realism of the installation. The participants were asked about what kind of icon could be chosen for the clearing action. Several original ideas came out such as an angry man cleaning furiously the wall or a wave erasing the painting. The other propositions were a rubber or a painting pot inspired by some graphical design softwares.

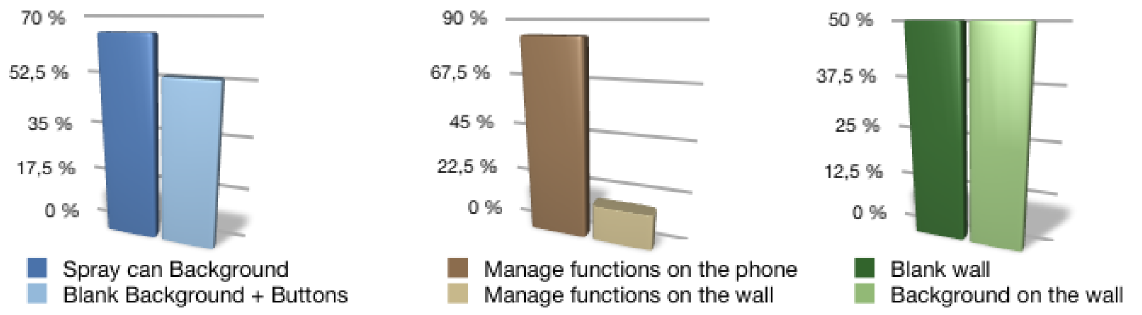


Figure 4.7: Survey results

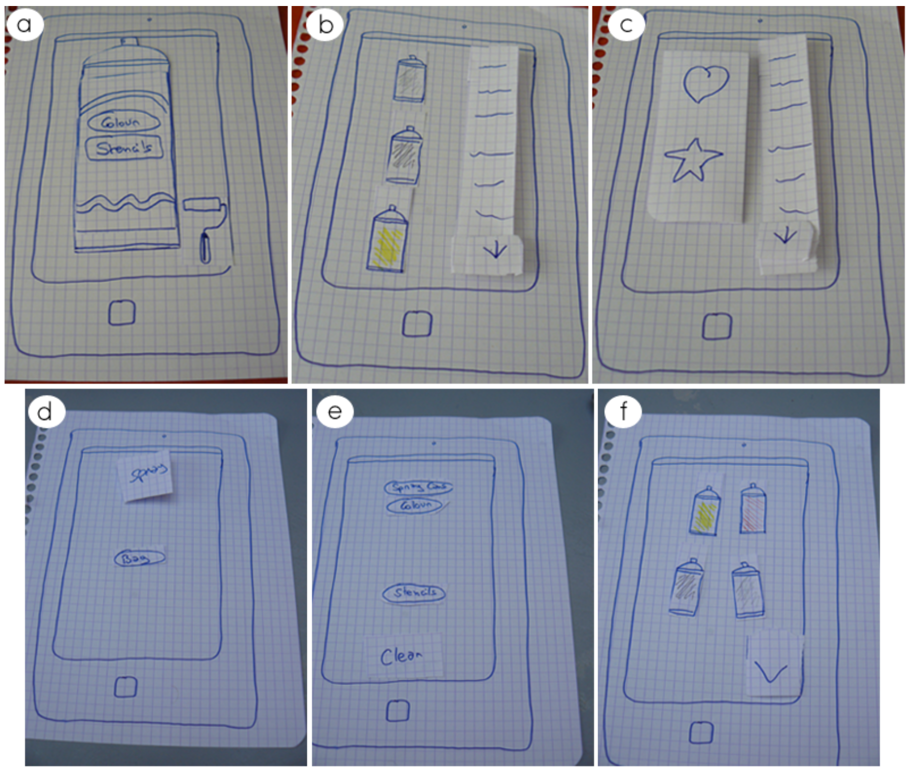


Figure 4.8: Low-Fi prototype - Participant's results

Result - Low-Fi prototype Figure 4.8 describes the two main trends concerning the user's ideal interface. As this task was a workshop, the participants were free to build their interface. As they did not have any background in UI conception, this test was more useful to analyze new ideas and propositions. The first trend, from image a to c, is with a spray can on the painting view. Here the participants would have preferred to have two buttons on the spray can to access to the colors and to use the stencils. The concept of the bag is here let down. The icon to clear the wall is placed on the painting screen (image a) too. The color selection screen (image b) presents the spray can in one column. The same disposition is shown for the stencils' screen (image c). The proposition of another participant (images d to f) is to avoid the spray can on the painting screen but to use the "Bag" button (image d). This will lead to an intermediary screen where it is possible to go in the colors' screen, stencils' screen or to clear the wall (image e). This solution may contain too many transition screens to access the drawing functions.

Concerning the proposed sketch, the participants have been enthusiastic. They found the concept of the bag attractive even if some of them would prefer to change the color and to use the stencils directly on the spray can, avoiding a transition screen. This is the only change they wished to the proposed sketch.

Conclusion of the low-fi test The test showed that the sketch submitted to the participant was on the good track. The workshop brought to the project group some indication about the number of colors and stencils expected by the users: around seven or eight each. The participants gave some original ideas to find a way to clear the wall. Their propositions will be studied for the following steps of the design. The management of the painting tool directly on the phone is confirmed. The painting screen with the spray can in background is kept. The idea of a bag is let down in order to reduce the number of transition screens. The user should be able to access the colors and the stencils screens directly from the painting screen.

4.3.5 Mock-up

To create the mock-up of the Graphical User Interface, some small corrections of the previous sketch shown in figure 4.4 are necessary. According to the results of the GUI test, the idea of a "bag" which would have contained all the painting tools of the artist is abandoned. The way to change the colors and to chose the stencils are now placed directly on the spray can to avoid a transition screen. The final mock-up, shown in figure 4.9 is the graphical interface designed for the mobile application. It is composed of five screens. Screen number 1 is the main screen called "painting screen". The background represents a spray can. The body of the spray can change according to the user's selected color. It is composed of four buttons. The top of the can is the spray button and is responsible for spraying on the wall. There is also a button called "Change Color". If hit, the button leads the user to the color screen. In the middle of the can is placed the stencil button. It brings the user to the screen in charge of the choice of the stencil. Finally, at the bottom of the spray can, the bar code is the setting button where the user can access the setting screen.
Screen number 2 is the color screen. Here the user can chose among eight colors. A simple touch of a colored spray can is enough to select a color. This action brings the user back to the main painting screen and updates the color of his/her spray can. In the color screen,

the user can also chose to clear the wall. This brings the user back to the painting screen and starts an animation to clear the wall.
Screen number 3 presents the stencils that the user can use. A "Back" button is placed at the left top of the screen and brings the user back to the painting screen. The user can also select a stencil. To do so, a simple touch on the stencil is necessary. This action leads the user to the screen number 4. Here, it is possible to control the stencil in real time on the wall. Two fingers are necessary to create a multitouch gesture which rotates the stencil. A double tap on the stencil brings back to the painting screen.
Screen number 5 shows the configuration of the system. The settings can be changed by the user if necessary.

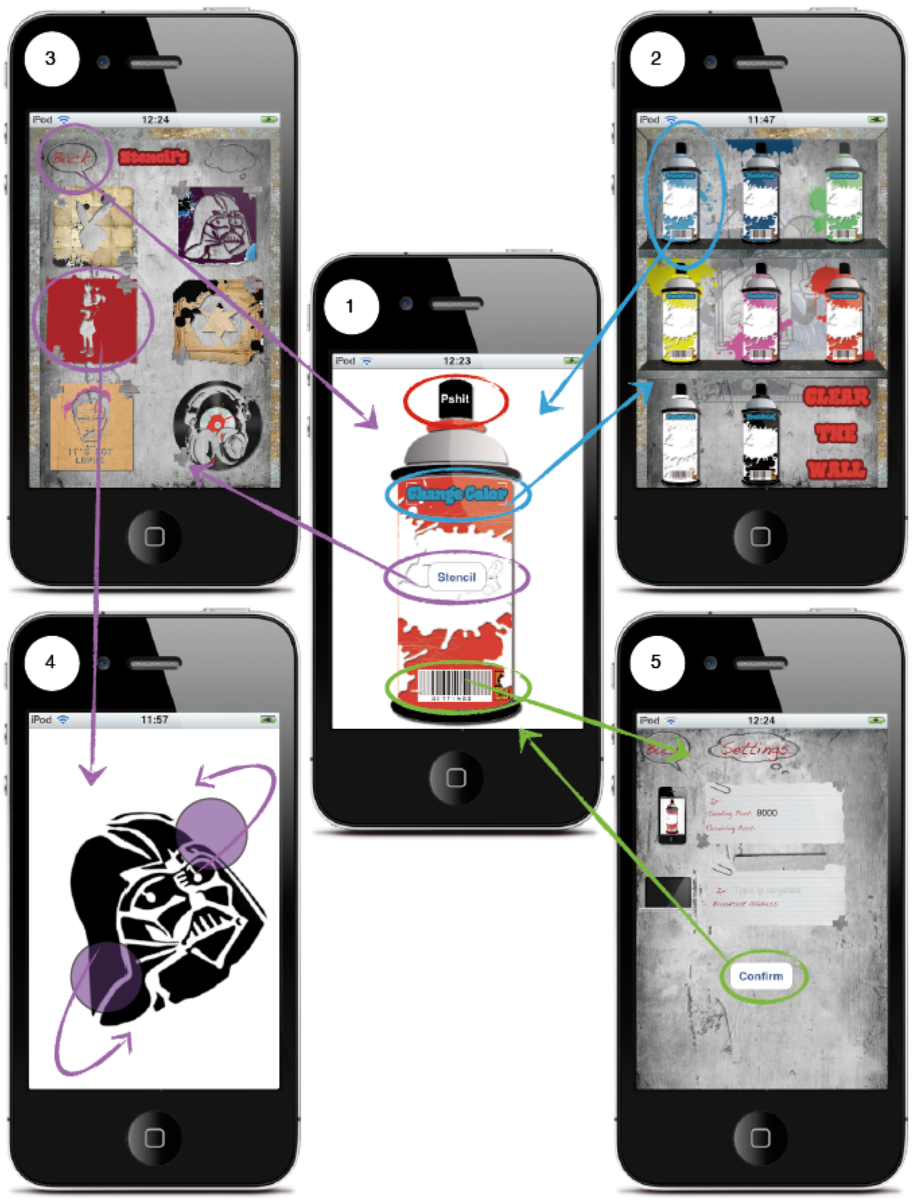


Figure 4.9: Mobile application Mock-Up

4.3.6 Architecture

The mobile application can be divided into several modules responsible for some specific functions. The architecture of the mobile application is shown in figure 4.10. The first module is in charge of the connection. An automatic link is established between the mobile application and the computer side application. The mobile application broadcasts its ip in an OSC message through the network and receives as a response the ip of the computer side application and the id of the player. This id is needed for the multiplayer mode. The painting module is responsible for transmitting the coordinates of the user's moves to the computer side application. It also informs if the user is painting or not. Another module is in charge of the stencils. When the user chooses his/her desired stencil, the mobile application sends in an OSC message the name of the stencil to the computer side application. The user can change the orientation of the stencil in real time so the stencil module updates the orientation sending the new angle to the computer side application. The last module is responsible for the color. It sends the RGB code to the computer side application in an OSC message.

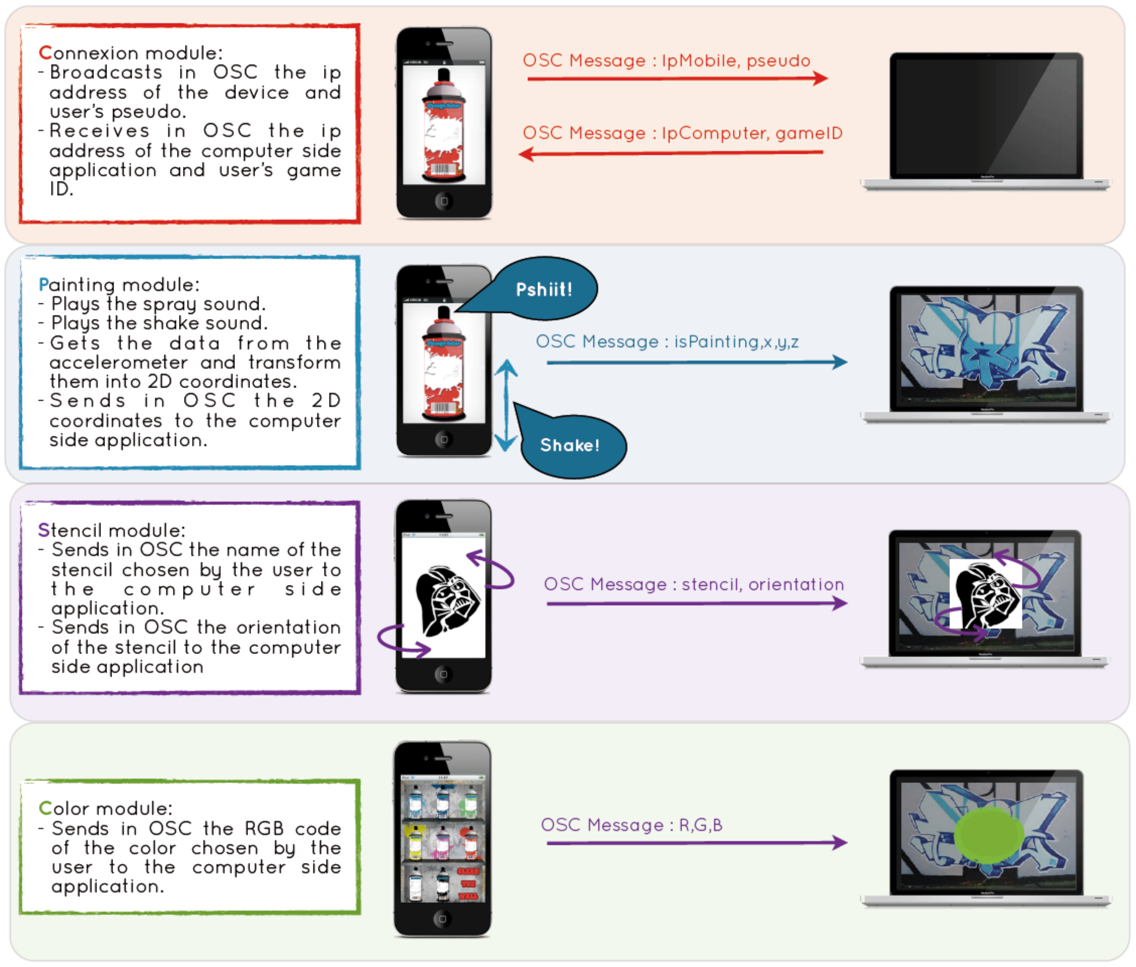

Connexion module:
- Broadcasts in OSC the ip address of the device and user's pseudo.
- Receives in OSC the ip address of the computer side application and user's game ID.
OSC Message : IpMobile, pseudo
OSC Message : IpComputer, gameID
Painting module:
- Plays the spray sound.
- Plays the shake sound.
- Gets the data from the accelerometer and transform them into 2D coordinates.
- Sends in OSC the 2D coordinates to the computer side application.
Pshiit!
OSC Message : isPainting,x,y,z
Shake!
Stencil module:
- Sends in OSC the name of the stencil chosen by the user to the computer side application.
- Sends in OSC the orientation of the stencil to the computer side application
OSC Message : stencil, orientation
Color module:
- Sends in OSC the RGB code of the color chosen by the user to the computer side application.
OSC Message : R,G,B

Figure 4.10: Modules of the mobile device application

4.4 Communication design

Since the mobile applications (one for each player) have to communicate with the computer side application, it is necessary to define the messages they will exchange in a protocol. Table 4.3 summarizes the different valid messages. The direction of the message is stated (up is for mobile-to-computerside and down is for computerside-to-mobile) and it is possible to observe that most of the messages are upcoming because once the server gave an ID to the client, there is no need for a response after each command. The commands will have direct consequences on the drawing, which is enough.

The format of the messages is really simple as it is basic OSC ones : the bundle has an address (name of the message) and a variable list of typed parameters.

Sequence diagram 4.11 illustrates the use of these messages in a common context. The protocol design is important because it will allow the use of any kind of clients (android, iphone, etc.) for the service as long as they communicate using these preset messages.

Table 4.3: Communication protocol between the applications

Message	**Parameters**	**Dir.**	**Comments**
/info	userName : String ipMobile : String	up	Broadcast sent by the mobile application to find the server and connect to it
/infoResponse	userId : Integer ipComputerSide : String	down	Response from the server with it's IP address (no more broadcast) and assignation of a user ID
/color	userId : Integer colorR : Integer colorG : Integer colorB : Integer	up	RGB color selection
/paint	userId : Integer isPainting : Boolean	up	Start or stop painting
/pos	userId : Integer x : Float y : Float	up	Position of the pointer
/clear	userId : Integer	up	Clear the canvas (fill in blank)
/useStencil	userId : Integer file : String	up	Start using stencil *file*
/rotateStencil	userId : Integer angle : Float	up	Rotate the stencil by *angle* radians
/stopRotateStencil	userId : Integer	up	Set the stencil, it will no longer be able to rotate nor translate. Now ready for painting over !
/endUseStencil	userId : Integer	up	Stop using a stencil
/quit	userId : Integer	up	Disconnect from the game (important for cleaning operations)

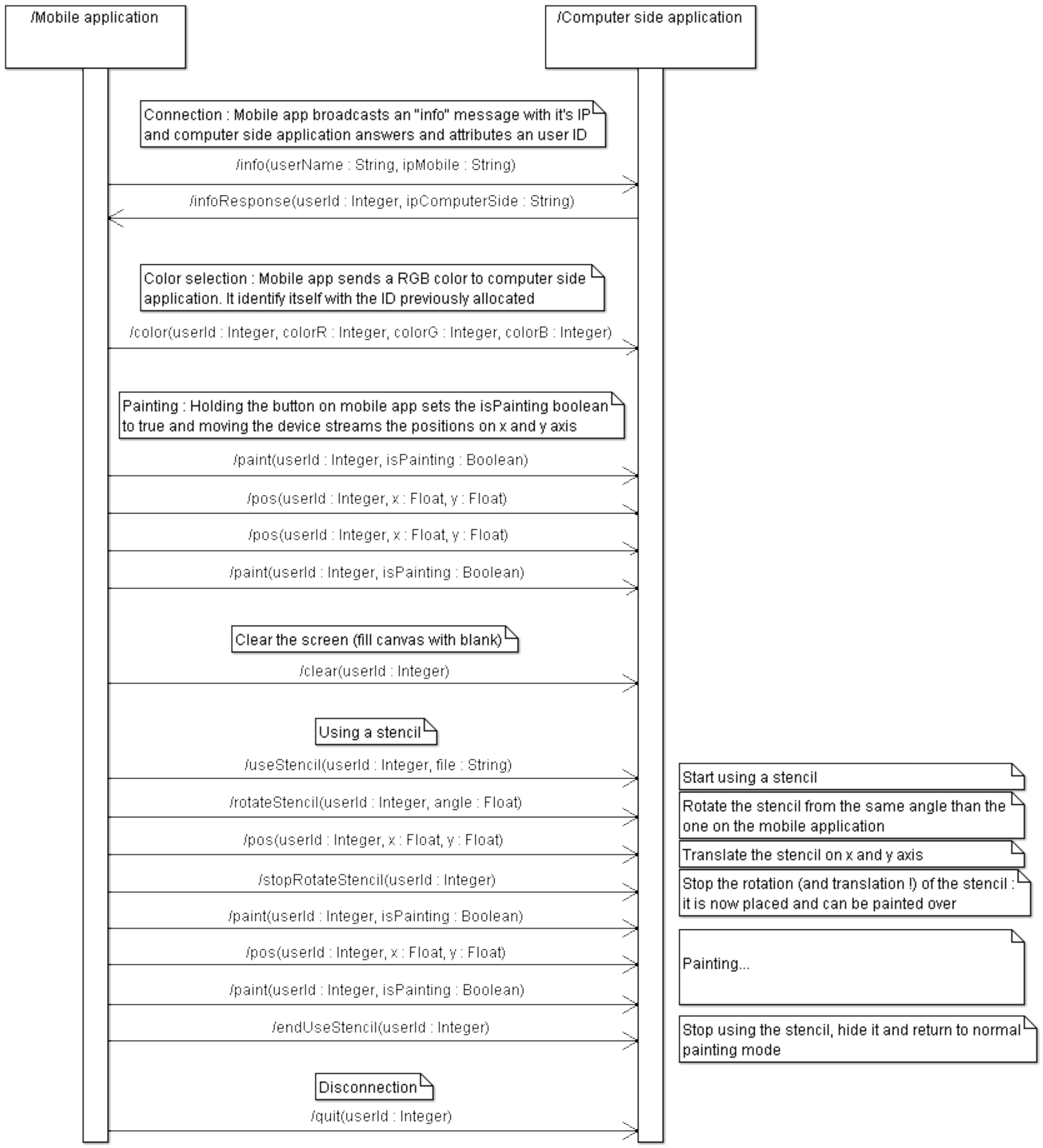

/Mobile application
/Computer side application
Connection : Mobile app broadcasts an "info" message with it's IP and computer side application answers and attributes an user ID
/info(userName : String, ipMobile : String)
/infoResponse(userId : Integer, ipComputerSide : String)
Color selection : Mobile app sends a RGB color to computer side application. It identify itself with the ID previously allocated
/color(userId : Integer, colorR : Integer, colorG : Integer, colorB : Integer)
Painting : Holding the button on mobile app sets the isPainting boolean to true and moving the device streams the positions on x and y axis
/paint(userId : Integer, isPainting : Boolean)
/pos(userId : Integer, x : Float, y : Float)
/pos(userId : Integer, x : Float, y : Float)
/paint(userId : Integer, isPainting : Boolean)
Clear the screen (fill canvas with blank)
/clear(userId : Integer)
Using a stencil
/useStencil(userId : Integer, file : String)
Start using a stencil
/rotateStencil(userId : Integer, angle : Float)
Rotate the stencil from the same angle than the one on the mobile application
/pos(userId : Integer, x : Float, y : Float)
Translate the stencil on x and y axis
/stopRotateStencil(userId : Integer)
Stop the rotation (and translation !) of the stencil : it is now placed and can be painted over
/paint(userId : Integer, isPainting : Boolean)
/pos(userId : Integer, x : Float, y : Float)
/paint(userId : Integer, isPainting : Boolean)
Painting...
/endUseStencil(userId : Integer)
Stop using the stencil, hide it and return to normal painting mode
Disconnection
/quit(userId : Integer)

Figure 4.11: Communication sequence diagram
]

4.5 Computer side application

The computer side application has to render the painting from the messages it receives from the user. It is divided into four modules, as shown in figure 4.12 :

- An OSC Module which receives, adapts and forwards the OSC messages (defined in the protocol from previous section) to the controllers.
- A Controller for each player, which manages all the resources according to the requests.
- Drawing Primitives (model), which are basic java beans (serializable class with only a constructor and accessors) used to store in memory the shapes drawn by the users.
- A Layer Manager (view), which renders the painting and the different tools (stencils, pointer, etc.) on the screen.

The Player Controllers implement the different functions provided by the system to each player. They are in charge of all the resources (data stored in the model, and rendering layers) and are an abstraction to make the code more object oriented and therefore modular. It is possible to move the pointer, draw, clear the screen, use stencils, save the painting, etc.

The layers manager is used to draw the different parts of the rendering in real time. It allows stacking different layers and updating them independently so it is not necessary to redraw everything, just the changes on each layer. There are three different types of layers. The paint layer is the basic canvas the players are painting on in "normal" mode, i.e. when no stencils are used. The stencil layers are then stacked on top of the paint layer when required, allowing the users to paint on them and copying the pixels on the Paint Layer when done. Finally the pointer layers are stacked over the pile to show each player's position.

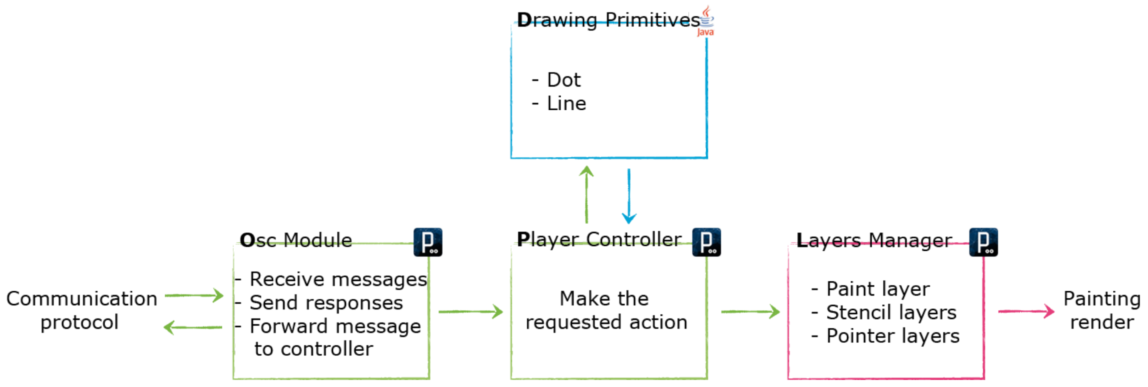


Figure 4.12: Computer side design

Chapter 5

Implementation

The design part described the architecture chosen to create the system. According to this general architecture, it is now possible to decide of one implementation. The following section explains the different processes implemented to create a functional prototype.

5.1 Mobile application

The device chosen is an iPhone so the mobile application is based on the iOS platform. For this implementation, the devices used to build and test the application are an iPhone4 and an iPodTouch 2G. The programming language taken in account by the iOS SDK is Objective-C.

5.1.1 Architecture and design pattern

The iOS SDK implements the MVC (Model View Controller) design pattern [39][p6]. The model is the representation of the real actors/objects/tools which will have a role in the mobile application such as a position, an accelerometer, an OSC message etc. The views display the graphical interface. Each view has its own controller. The controllers are the link between the views and the model. When a user's action is detected, the controller refers to the model to update the view. Figure 5.1 illustrates the interactions implied by the MVC design pattern. The Objective-C application contains a delegate which is responsible for its initialization.

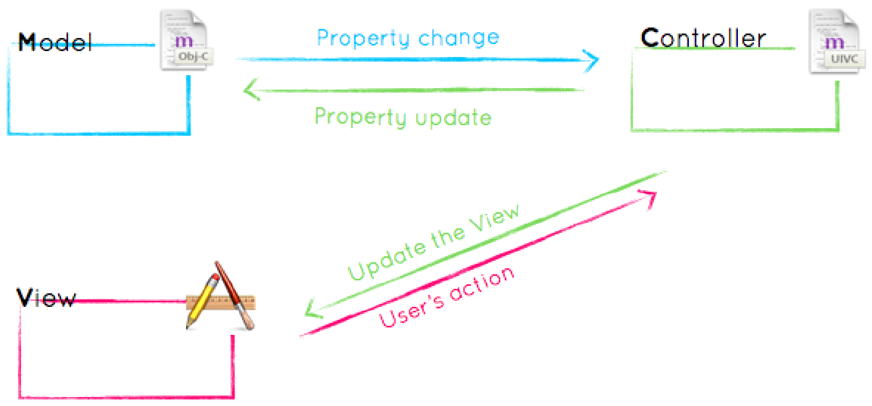
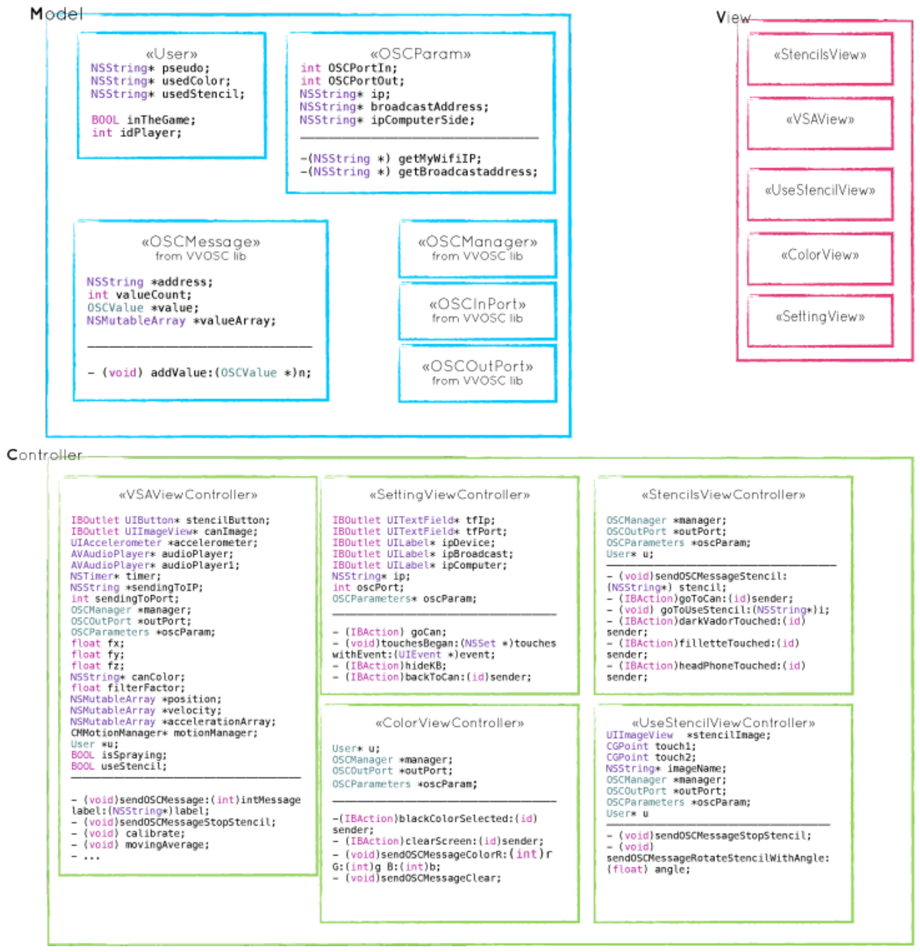


Figure 5.1: Interaction in a pattern Model View Controller

5.1.2 Class Diagram

Figure 5.2 shows the class diagram used for the implementation of the mobile application. The classes are included in the three packages of the project: Model, View and Controller.



```
NSString* pseudo;
NSString* usedColor;
NSString* usedStencil;

BOOL inTheGame;
int idPlayer;
```

```
int OSCPortIn;
int OSCPortOut;
NSString* ip;
NSString* broadcastAddress;
NSString* ipComputerSide;
-------------------------------------
-(NSString *) getMyWifiIP;
-(NSString *) getBroadcastaddress;
```

```
NSString *address;
int valueCount;
OSCValue *value;
NSMutableArray *valueArray;
-------------------------------------
- (void) addValue:(OSCValue *)n;
```

```
IBOutlet UIButton* stencilButton;
IBOutlet UIImageView* canImage;
UIAccelerometer *accelerometer;
AVAudioPlayer* audioPlayer;
AVAudioPlayer* audioPlayer1;
NSTimer* timer;
NSString *sendingToIP;
int sendingToPort;
OSCManager *manager;
OSCOutPort *outPort;
OSCParameters *oscParam;
float fx;
float fy;
float fz;
NSString* canColor;
float filterFactor;
NSMutableArray *position;
NSMutableArray *velocity;
NSMutableArray *accelerationArray;
CMMotionManager* motionManager;
User *u;
BOOL isSpraying;
BOOL useStencil;
-------------------------------------
- (void)sendOSCMessage:(int)intMessage
label:(NSString*)label;
- (void)sendOSCMessageStopStencil;
- (void) calibrate;
- (void) movingAverage;
- ...
```

```
IBOutlet UITextField* tfIp;
IBOutlet UITextField* tfPort;
IBOutlet UILabel* ipDevice;
IBOutlet UILabel* ipBroadcast;
IBOutlet UILabel* ipComputer;
NSString* ip;
int oscPort;
OSCParameters* oscParam;
-------------------------------------
- (IBAction) goCan;
- (void)touchesBegan:(NSSet *)touches
withEvent:(UIEvent *)event;
- (IBAction)hideKB;
- (IBAction)backToCan:(id)sender;
```

```
OSCManager *manager;
OSCOutPort *outPort;
OSCParameters *oscParam;
User* u;
-------------------------------------
- (void)sendOSCMessageStencil:
(NSString*) stencil;
- (IBAction)goToCan:(id)sender;
- (void) goToUseStencil:(NSString*)i;
- (IBAction)darkVadorTouched:(id)
sender;
- (IBAction)filletteTouched:(id)
sender;
- (IBAction)headPhoneTouched:(id)
sender;
```

```
User* u;
OSCManager *manager;
OSCOutPort *outPort;
OSCParameters *oscParam;
-------------------------------------
-(IBAction)blackColorSelected:(id)
sender;
- (IBAction)clearScreen:(id)sender;
- (void)sendOSCMessageColorR:(int)r
G:(int)g B:(int)b;
- (void)sendOSCMessageClear;
```

```
UIImageView  *stencilImage;
CGPoint touch1;
CGPoint touch2;
NSString* imageName;
OSCManager *manager;
OSCOutPort *outPort;
OSCParameters *oscParam;
User* u
-------------------------------------
- (void)sendOSCMessageStopStencil;
- (void)
sendOSCMessageRotateStencilWithAngle:
(float) angle;
```

Figure 5.2: Class Diagram

The object User regroups personal information about the painter. It contains a pseudo, an id and the name of the color and the stencil currently used. OSCParam is a class which contains the configuration of the connection such as the ip of the device, the ip of the computer side application and the ip to broadcast on the network. The input and output port belongs also to this class. Other classes from the VVOSC library could be included in the model package such as the OSCMessage class or the OSCManager class. This library will be described later in the report.

The package View regroups the different screens needed for the application. It contains the different elements of the graphical user interface. The major part of the views is created with Interface Builder incorporated in the iOS SDK. Interface Builder allows the developers to place the graphical elements (UIButton, UILabel...) and to link them to the functions of the Controllers. However, some updates of the graphical interface must be done programmatically and are coded into the controller functions.

The last package is called Controller. Each controller is in charge of one view. The controllers are called when a user action occurs. The graphical elements of the view are linked to the "IBAction" functions of the controllers. Those functions update the views according to the model. The controllers function are also responsible for sending the OSC messages to the computer side application according to the user's interactions.

5.1.3 VVOSC Library & OSC Protocol

The VVOSC Library is used to handle the OSC protocol in the project. This library is based on the UDP protocol. It contains 42 classes which implement the characteristics of the OSC protocol. This study will focus on four main classes which are the result of this library: OSCManager, OSCMessage, OSCOutPort, OSCInPort. Figure 5.3 to 5.5 show samples of code, implemented in the project, to set-up and use the VVOSC library.
The first step is to include the controller class VVOSC.h. The second step to set up the the OSC protocol is to allocate and initialize the OSCManager. The manager prepares the application to receive and to send OSC data. In order to send OSC messages, an output port has to be initialized with the manager. To do so, the destination ip and the destination port are needed. After this the set up is over. Those steps are illustrated by Figure 5.3.

```
//*********** OSC Set-Up ***********//
manager = [[OSCManager alloc] init];
[manager setDelegate:self];
sendingToIP = [oscParam ipComputerSide];
sendingToPort = [oscParam OSCPortOut];
outPort = [manager createNewOutputToAddress:sendingToIP
atPort:sendingToPort];
NSLog(@"Configuration IP out: %@ On Port: %d", sendingToIP, sendingToPort);
```

Figure 5.3: OSC set-up

The function to send the OSC message must now be implemented. This function is shown on figure 5.4. To send a message, three elements are needed: an instance of the class OSCMessage, the label of the message and the values to send. The OSC message is an envelope. It will contain the label and the values. The label is the name of the message. It is how the messages can be differentiated at the reception. The syntax has to respect this rule: /label. The values are the data of the message. It's possible to add multiple values of different types in one message. The last step is to send the message calling the send function of the OSCOutport already created.

```
//*********** To Send OSC Message ***********//
- (void)sendCoordinatesOSCMessageX:(float)floatX Y:(float)floatY Z:(float)
floatZ label:(NSString*)label {
    OSCMessage *msg = [OSCMessage createWithAddress:label];
    [msg addInt:[u idPlayer]];
    [msg addFloat:floatX];
    [outPort sendThisPacket:[OSCPacket createWithContent:msg]];
}
```

Figure 5.4: Sending OSC messages with VVOSC

The reception of the OSC message is simpler. The manager sets up the application to receive the OSC messages. So, when an OSC message is sent to the device ip and on the input port, the application handles the event thanks to the function receivedOSCMessage. It is then possible to implement the treatment of the incoming OSC message according to its label. The way to implement the reception of OSC messages is shown on figure 5.5.

```
//*********** To Receive OSC SMessage ***********//
// called by delegate of the application on message
- (void) receivedOSCMessage:(OSCMessage *)m {

    NSString *address = [m address]; //label
    NSMutableArray *valueArray = [[NSMutableArray alloc] initWithArray:[m
           valueArray]];
    oscParam.ipComputerSide = [[valueArray objectAtIndex:1] stringValue];
    u.idPlayer = [[valueArray objectAtIndex:0] intValue];
```

Figure 5.5: Receiving OSC messages with VVOSC

5.1.4 Navigation through the application

The iOS SDK allows the developers to chose several templates of projects which have different methods of navigation. For this implementation, a window based project has been chosen. The window based project contains only the window and the delegate of the application. This way, the developer is free to chose the logical succession of views and the way to navigate.

The navigation object chosen is a UINavigationController but its implementation is quite different from the basic way to do it. An instance of this class is created in the delegate of the application. It will be in charge of the navigation through all the application. To set-up the UINavigationController the first step is the same for every Objecive-C objects: allocation and initialization. Then, the first view controller needs to be allocated and initialized too in order to be added to the navigation controller. The navigation controller is then asked to push the added view as the top view. The implementation of the previous steps is illustrated by figure 5.6.

```
navigationController = [[UINavigationController alloc] init];
VSAViewController *viewController = [[VSAViewController alloc]
initWithNibName:@"VSAViewController" bundle:nil];
[nanavigationController pushViewController:viewController animated:NO];
[viewController release];
[window addSubview:nanavigationController.view];
[self.window makeKeyAndVisible];
```

Figure 5.6: Initialization of the application navigation

With this basic implementation, the UINavigationController places automatically on the top of the view a navigation bar. The logical order of the views is saved into the UINavigationController object. The navigation controller manages the title of the current view and the back button of the next view according to the current view properties. This logic is illustrated by figure 5.7. For ergonomic reasons, the graphical user interface of the mobile application doesn't need the navigation bar. As the navigation bar is erased of the screen, the logical way to navigate is broken because the back button is not present anymore. To fix this issue, a custom back button is implemented and linked to a custom back function which pushes the previous view as the top view thanks to this code: [self.navigationController popViewControllerAnimated:YES].

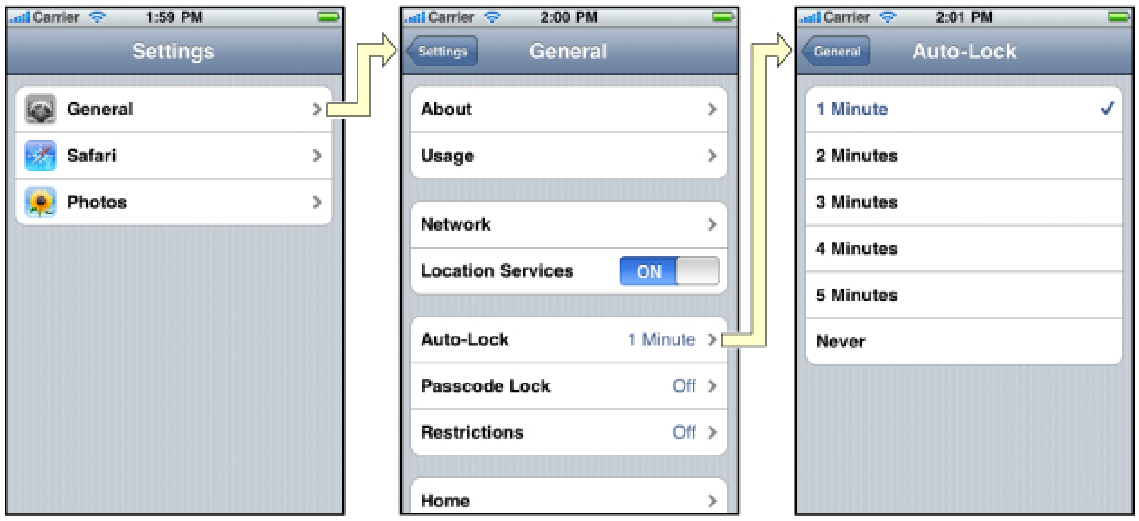


Figure 5.7: Logical navigation management [11]

5.1.5 Core Motion API

The Core Motion API is listening to the accelerometer and the gyroscope [40]. It is capable to get data from both sensors. Data from both sensors allow to have some very useful information: device's attitude, rotation rate, direction of gravity and acceleration given by the user. Coupling the accelerometer with the gyroscope offers six degrees of freedom (DoF). The three first ones are the translations along each axis: x, y and z. The three other DoFs are simply the rotation on each axis and are called roll, pitch and yaw.
Data can be accessed through a CMDeviceMotion object. The function responsible to listen to the accelerometer and gyroscope is startDeviceMotionUpdates. For the purpose of the project, the data needed is the acceleration given by the user to the iphone. This data is stored in the userAcceleration property of the CMDeviceMotion object. This acceleration is also called linear acceleration. It is obtained by subtracting the gravity to accelerometer data.

Now, that the way to get acceleration values thanks to the Core Motion API has been introduced, it is important to present how to go from those values to coordinates. As this process is done by the computer side application it is developed in section 5.2.4 (page 58).

5.1.6 Stencil manipulation

The project implements the possibility to use different stencils. To make this function realistic and attractive, the user should be able to manipulate the stencil in order to place it on the wall. For this, the user needs to translate the stencil, to make it move onto the desired part of the wall. This first functionality is solved using the movements of the device already sent to the computer side application. A translation is thus possible following the x and y axis. A second functionality is to rotate the stencil. This functionality is implemented directly on the mobile device in the class UseStencilViewController. A view controller can handle the user event such as the touches or the taps. iOs manages multitouch, so it is possible to find the coordinate of several touch events on the same view. Furthermore, the view controller can identify when the touches are moving on the screen. Thanks to this information, it is simple to retrieve the angles made by the two touches using the two arguments arctangent (arctan(x,y)) function. The aim of this function is to take the cartesian coordinates of one point of a two dimensional plan (x,y) and to compute the angle done by this point with the x axis [41][p260]. formula 5.1:

$$angle = \frac{arctan(Ytouch2 - Ytouch1, Xtouch2 - Xtouch1)180}{\pi} \tag{5.1}$$

Thus, it is possible to compute the angle for the previous and current positions of the two touches. From the difference between those two angles results the angle to rotate in order to update the stencil on the view. Figure 5.8 illustrates the implementation of the stencil rortation.

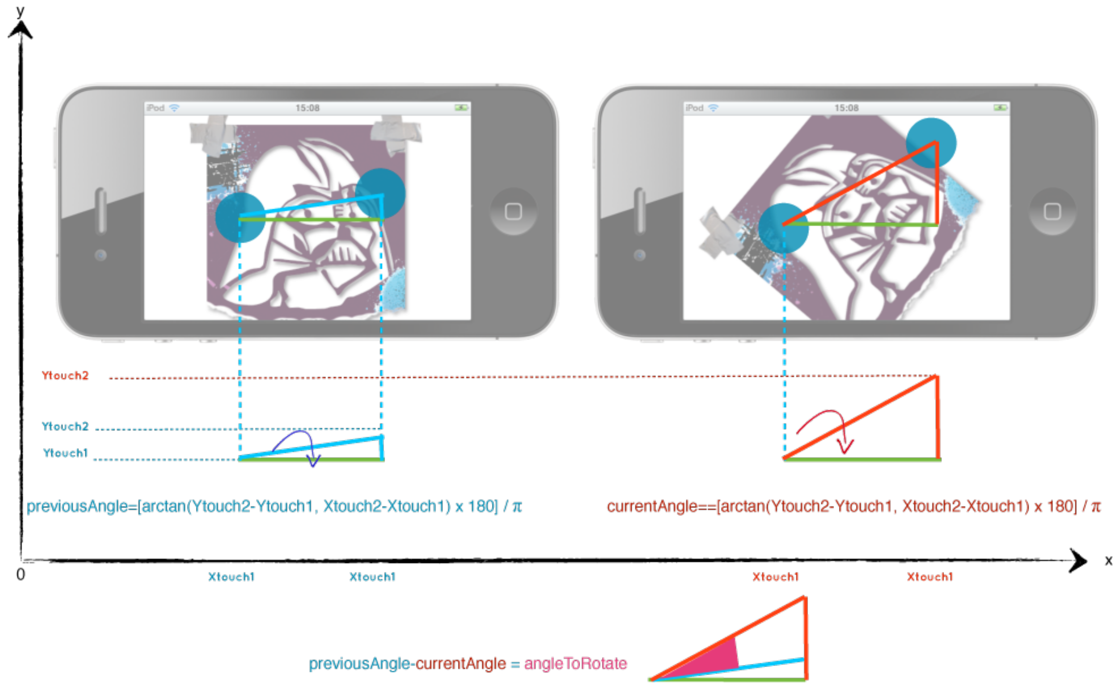

y
iPod
15:08
iPod
15:08
Ytouch2
Ytouch2
Ytouch1
previousAngle=[arctan(Ytouch2-Ytouch1, Xtouch2-Xtouch1) x 180] / π
currentAngle==[arctan(Ytouch2-Ytouch1, Xtouch2-Xtouch1) x 180] / π
0
Xtouch1
Xtouch1
Xtouch1
Xtouch1
x
previousAngle-currentAngle = angleToRotate

Figure 5.8: Stencil rotation principle

5.2 Computer side application

The computer side application has been implemented in Java and Processing languages. Libraries "oscP5" and "Nootropic's layers" were used to respectively provide the communication interface and the paint layers manager. This section will detail the computer side application's implementation based on these tools.

5.2.1 Architecture and design pattern

Since this part of the software is in Processing language, it does not have a real underlying design pattern or base framework to comply to. However, it is possible to say that it is a loose Model View Controller because all the different parts are separated in classes which have the same roles as models, views or controllers (see figure 4.12 page 47 for more information on architecture).

Processing files are not really java files even though the language is really close, they use a library (processing.core) but also allow some structural simplifications like :

- no packages
- no need of a main class
- setup() method called at startup
- draw() method called each frame
- new data type "color" (which is an Integer)
- easy random numbers generator with method random(min,max)

5.2.2 Class Diagram

Figure 5.9 shows computer side application's class diagram. It is divided in four parts which are the main classes, the player controller, the layers and the primitives.

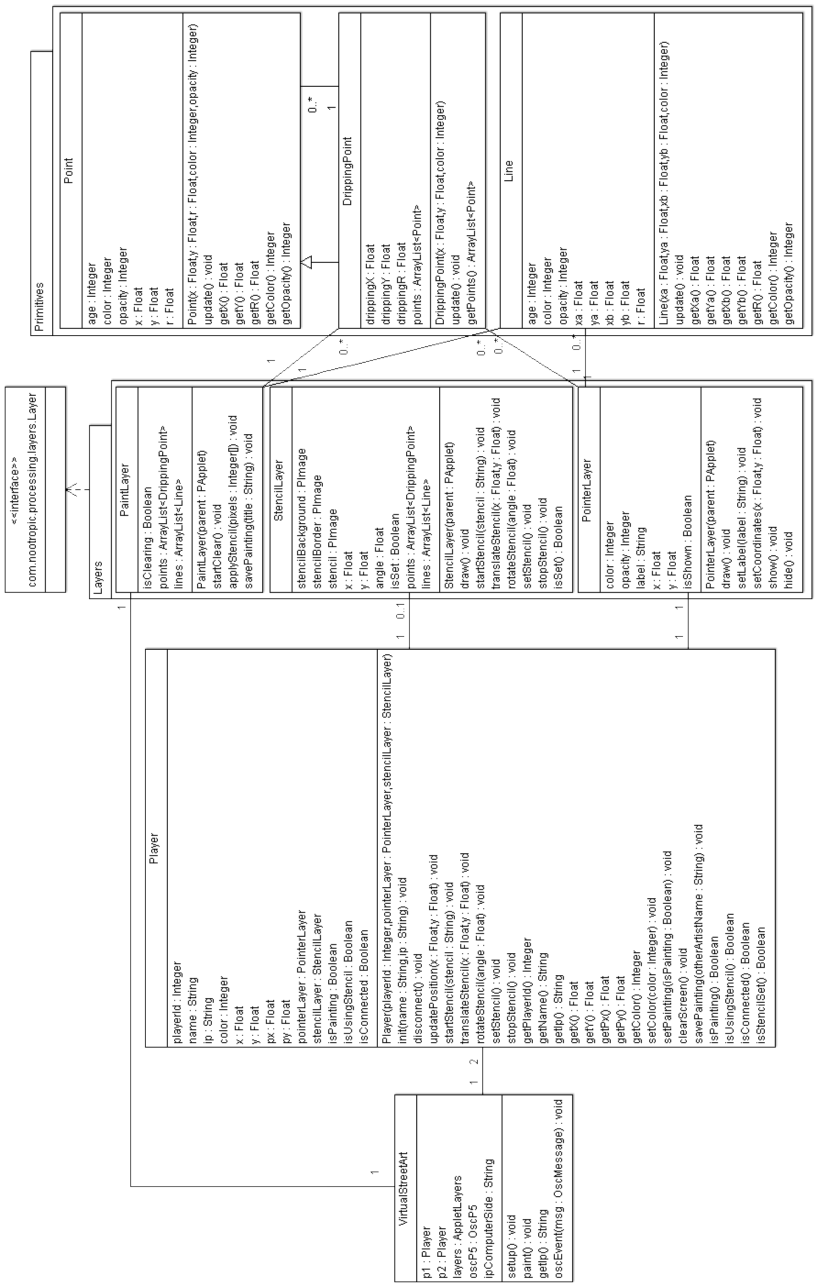


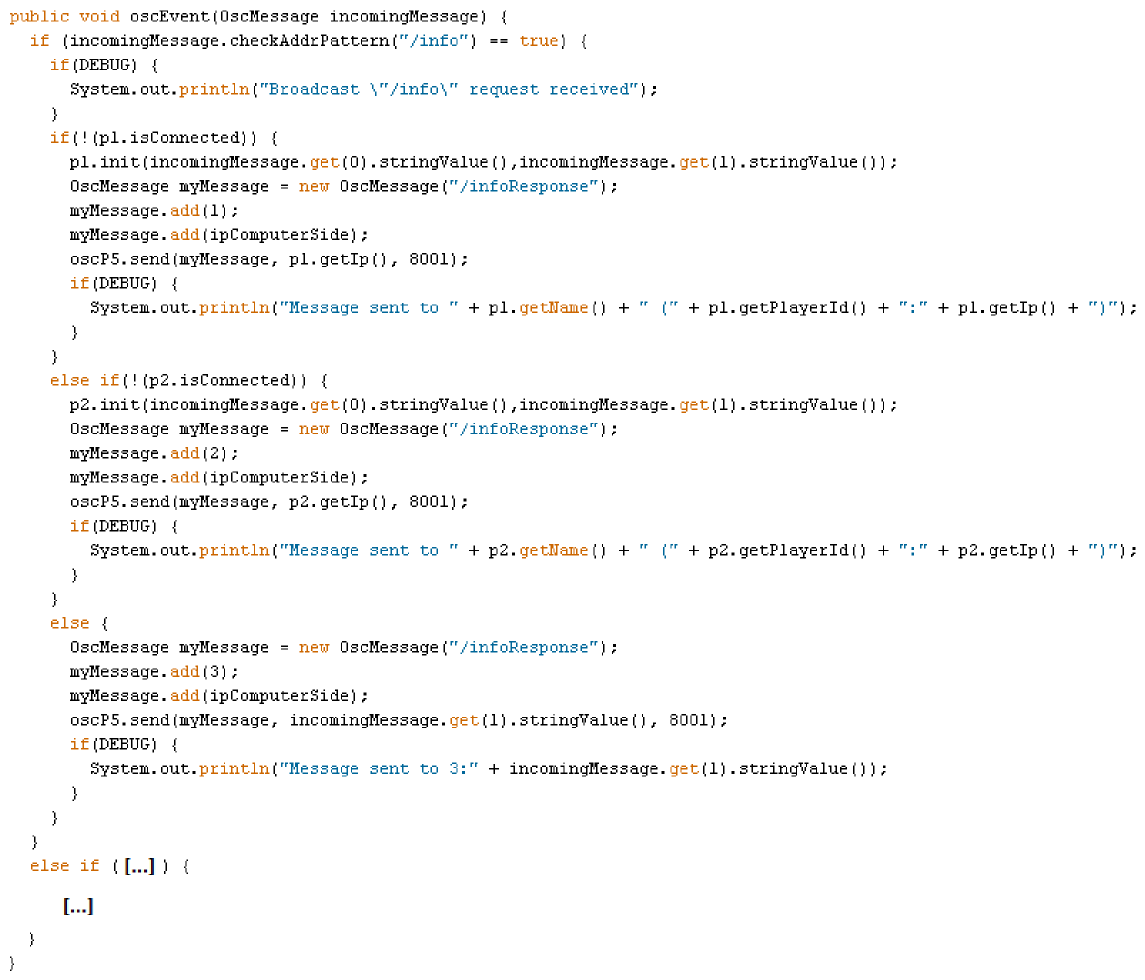
Figure 5.9: Computer side application class diagram

5.2.3 Communication module using oscP5 library

In order to receive the OSC messages sent by the players, this project uses the processing oscP5 library [42].

An object of type OscP5 is declared and initialized on the main class then oscEvent(OscMessage incomingMessage) method is overriden to catch each and every messages defined in the communication protocol (table 4.3 page 45). From these messages, the program calls the appropriate methods on the players controllers with the arguments casted in the right types (Float, Integer, Boolean, etc.) as shown on figure 5.10.

The OscP5 object is also used to send messages down to the client, figure 5.10 shows the implementation of the /infoResponse message and how easy it is to add parameters to the message.



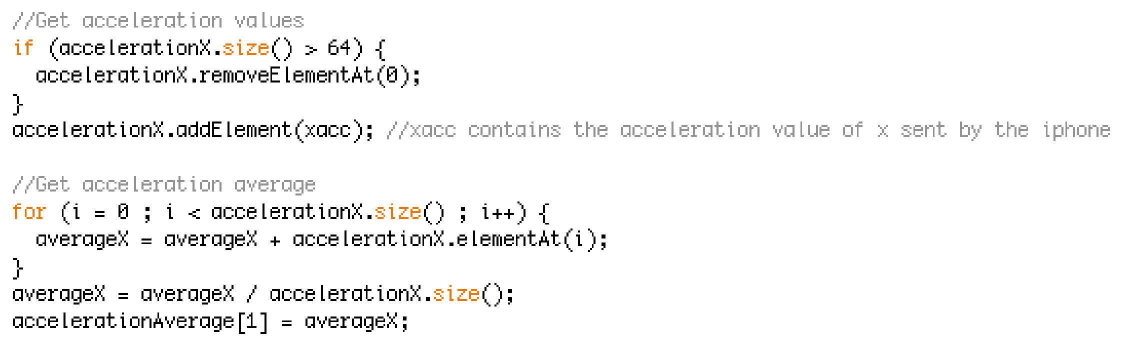
```
public void oscEvent(OscMessage incomingMessage) {
  if (incomingMessage.checkAddrPattern("/info") == true) {
    if(DEBUG) {
      System.out.println("Broadcast \"/info\" request received");
    }
    if(!(p1.isConnected)) {
      p1.init(incomingMessage.get(0).stringValue(),incomingMessage.get(1).stringValue());
      OscMessage myMessage = new OscMessage("/infoResponse");
      myMessage.add(1);
      myMessage.add(ipComputerSide);
      oscP5.send(myMessage, p1.getIp(), 8001);
      if(DEBUG) {
        System.out.println("Message sent to " + p1.getName() + " (" + p1.getPlayerId() + ":" + p1.getIp() + ")");
      }
    }
    else if(!(p2.isConnected)) {
      p2.init(incomingMessage.get(0).stringValue(),incomingMessage.get(1).stringValue());
      OscMessage myMessage = new OscMessage("/infoResponse");
      myMessage.add(2);
      myMessage.add(ipComputerSide);
      oscP5.send(myMessage, p2.getIp(), 8001);
      if(DEBUG) {
        System.out.println("Message sent to " + p2.getName() + " (" + p2.getPlayerId() + ":" + p2.getIp() + ")");
      }
    }
    else {
      OscMessage myMessage = new OscMessage("/infoResponse");
      myMessage.add(3);
      myMessage.add(ipComputerSide);
      oscP5.send(myMessage, incomingMessage.get(1).stringValue(), 8001);
      if(DEBUG) {
        System.out.println("Message sent to 3:" + incomingMessage.get(1).stringValue());
      }
    }
  }
  else if ( [...] ) {

     [...]

  }
}
```

Figure 5.10: oscP5 implementation

5.2.4 From Acceleration to Coordinates

Now that the way to communicate with the iphone has been explained we can focus on the process applied to the data received. This part will focus on acceleration data. As it has been presented in section 3.2 (page 26), data from the accelerometer have to be processed to be transformed into coordinates. This subsection will go through all the filters and algorithms applied to those data in order to do it.

Moving Average: It is the first algorithm applied. As it is a time consuming process, it is needed to find the correct number of samples to record to calculate the average. According to previous studies [23], it appears that the best compromise between accuracy and time consumption is to take 64 samples. Then, the algorithm is divided into two parts. Samples are saved in an array. So, the first step is to check if there are already 64 samples. In this case, the oldest one is deleted and the new one is added. In the other case, the new sample is just added. The second step is just to calculate the average. This algorithm applied to x acceleration is shown on figure 5.11.



```
//Get acceleration values
if (accelerationX.size() > 64) {
  accelerationX.removeElementAt(0);
}
accelerationX.addElement(xacc); //xacc contains the acceleration value of x sent by the iphone

//Get acceleration average
for (i = 0 ; i < accelerationX.size() ; i++) {
  averageX = averageX + accelerationX.elementAt(i);
}
averageX = averageX / accelerationX.size();
accelerationAverage[1] = averageX;
```

Figure 5.11: Moving Average applied to x acceleration

Discrimination window: The accelerometer is sensitive to any movements of the iphone. Even a small movement due to hand shaking has an impact on the acceleration values sent by the accelerometer. For this reason it is important to define what is considered as a movement. This is the role of the discrimination window. It will consider as non-movement a value under a certain threshold. But it is important to define properly this threshold. To determine this value, data have been recorded during 10 seconds while someone was handling the iphone in a non-movement state. The result of this experiment has given an absolute value of 0.025. So if an absolute value of acceleration is below 0.025 it is considered as 0. The discrimination window step is shown on figure 5.12.

Double Integration Now that the data sent by the accelerometer have been filtered, they can be used to get the position of the iphone. Data are integrated a first time to get the velocity from the acceleration. Then, they are integrated a second time to obtain the position from the velocity. This process is shown on figure 5.13.

Out of bound detection The last step is just to be sure that the position is not out of the screen. A simple filter is applied to put the position at the border in case it goes out

```
// Discrimination window
if ( abs(accelerationAverage[1]) < 0.025f ) {
  accelerationAverage[1] = 0.0F; //accelerationAverage[1] contains the current acceleration value of x
}

if ( abs(accelerationAverage[3]) < 0.025f ) {
  accelerationAverage[3] = 0.0F; //accelerationAverage[3] contains the current acceleration value of y
}
```

Figure 5.12: Discrimination window

```
// From acceleration to velocity
for (i = 0 ; i < accelerationAverage.length ; i++) {
  result = velocity[i] + accelerationAverage[i] + ((accelerationAverage[i+1] - accelerationAverage[i]) / 2);
  velocity[i+1] = result;
  i++;
}

// From velocity to position
for (i = 0 ; i < velocity.length ; i++) {
  result = position[i] + velocity[i] + ((velocity[i+1] - velocity[i]) / 2);
  position[i+1] = result;
  i++;
}
```

Figure 5.13: Double Integration of acceleration data

of the limit. This process is shown on figure 5.14.

```
// Out of bound detection
if (position[1] < 0.0F) {
  position[1] = 0.0F;
  velocity[1] = 0.0F;
}
else if (position[1] > 1280.0F) {
  position[1] = 1280.0F;
  velocity[1] = 0.0F;
}

if (position[3] < 0.0F) {
  position[3] = 0.0F;
  velocity[3] = 0.0F;
}
else if (position[3] > 800.0F) {
  position[3] = 800.0F;
  velocity[3] = 0.0F;
}
```

Figure 5.14: Out of bound detection

Limitations The first thing that is important to notice is that only the iphone 4 (or the ipod new generation) are equipped with accelerometer and gyroscope. So such a device is needed to make possible to distinguish the rotation and the acceleration.
The group members had only one iphone 4 to test the implementation. The acceleration value sent by the iphone was not always the one expected. That's why some data have been recorded in a file to be analyzed. The acceleration values of x sent by the iphone during a simple movement to the right (without coming back to the starting point) have

been recorded. These values are presented in a graph on figure 5.15. It is surprising to see two peaks on the graph. The first one clearly represents the acceleration increasing and decreasing while the iphone is moving to the right. But the negative peak is not expected. It should appear only when the iphone is moving to the left. No explanation has been found to this 'rebound' effect.

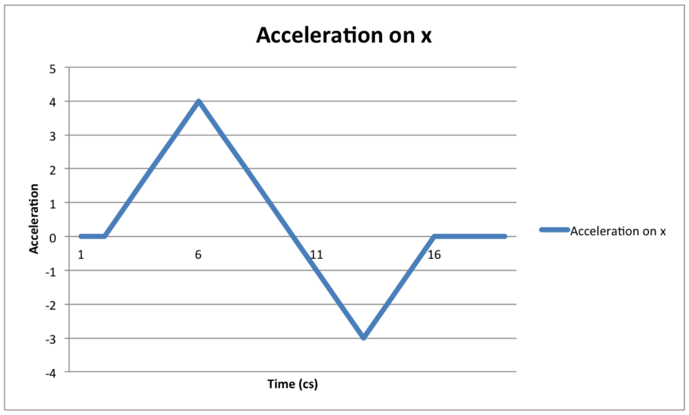


Figure 5.15: Acceleration values of x during a movement to the right

In order to solve this problem a simple algorithm has been implemented. The idea is to check when the values cross zero. When it is detected, the values are put to zero until a new cross zero is detected. This process applied to values shown in figure 5.15 is presented on figure 5.16

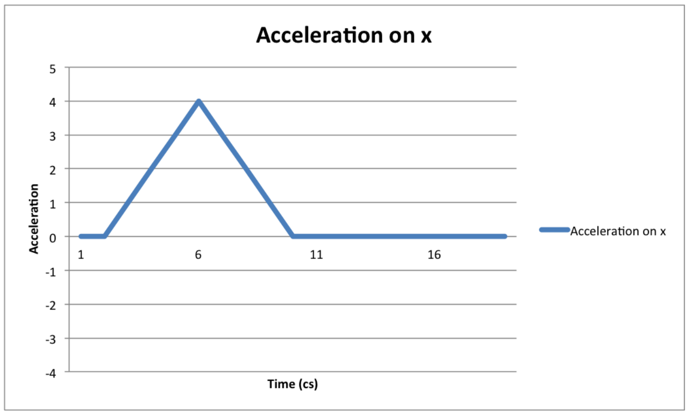


Figure 5.16: Remove 'rebound' effect

The process that is responsible to transform acceleration values into coordinates has just been presented. The next subsection explains how the player controllers work.

5.2.5 Player Controllers

The player controllers are triggered by the OSCEvent forwarded by the communication module. Each player has his own stencils and pointer layers plus a set of tools to manage it. Besides the accessors (getters and setters) and the booleans (isPainting, etc.), the players have several controller methods :

Connection / Disconnection Since the system is designed for maximum two artists, the players are singletons created at the beginning of the program. When a client connects, it does not "create" a new player but connects to an existing one which is not currently in use. If there are already two artists playing, the mobile application gets a SERVERFULL answer. This implies that when a client application disconnects, it has to send a message to free its slot.

Position update There is one particular method which is really important in the controller : void updatePosition(Float x, Float y) (snippet 5.17). It is in charge of updating the current coordinates of the player by using the process shown in subsection 5.2.4, moving the graphic pointer on the pointer layer and moving the stencil if in stencil mode. It is called each cycle and makes the system interactive since it shows a direct response to any of the player's move.

Stencils use and clear screen In order to use stencils, the messages coming from the communication module are directly forwarded to the player's stencil layer without any transformation. To clear the screen, the same principle is applied and the message is forwarded to the paint layer.

Saving the painting When it comes to save the painting, a message is forwarded to the paint layer with a generated name composed by :

- a timestamp (year, month, day, hour, minute, second)
- the artist's name (the one who pressed the "Save" button)
- the other artist's name

This name format allows the creation of an automatic upload script to share these paintings on the internet with all the relative informations.

```
public void updatePointer(Float myx, Float myy) {
  if(isConnected) {
    this.px = this.x;
    this.py = this.y;
    this.x = myx;
    this.y = myy;
    this.x = this.x + WIDTH / 2;
    this.y = this.y + HEIGHT / 2;
    this.pointerLayer.setCoordinates(x, y);
    this.stencilLayer.translateStencil(x, y);
    if(DEBUG) {
      System.out.println("Player " + name + " moved his pointer to x:" + myx + " - y:" + myy);
    }
  }
}
```

Figure 5.17: Position update method

5.2.6 Drawing primitives

The drawing primitives used for this project are really simple java beans. There are three types of primitives : a line, a point and a dripping point which is a particular kind of point. They all have an age (number of rendering cycles since creation maxed at 100), a color, an opacity, a size and coordinates. They also have an update() method, called at each rendering cycle, which computes their age and new coordinates if necessary.

The specific case of the dripping point is a bit tricky. As shown on class diagram 5.9 page 56, dripping points are points (objects inheritance) which contain a list of other points and override the update() method. This list of new points is used to draw an animation : by adding a smaller new point under the main one each cycle (with some random noise in the movement as shown in code snippet 5.18), a dripping effect is created. The animation stops when the radius of the point to be added is becoming too small (0.05F). The list of points also contains a set of "noisy" points around the main one to make a spray paint effect.

```
public void update() {
  super.update();
  if ((drippingr * drippingradiuscoef) > 0.05F) {
    drippingr *= drippingradiuscoef;
    drippingx += (random(-1, 1) * 0.5);
    drippingy += drippingcoef * sqrt(sqrt(sqrt(drippingr)));
    points.add(new Point(drippingx, drippingy, drippingr, getColor(), 120));
  }
}
```

Figure 5.18: Dripping paint effect

With this model of dots and lines, the rendering of the paint is not perfect but can be compared with the real behavior (see figures 5.19 and 3.8 page 33) and could be improved with speed delta to change the opacity of the paint depending on painter's move speed.

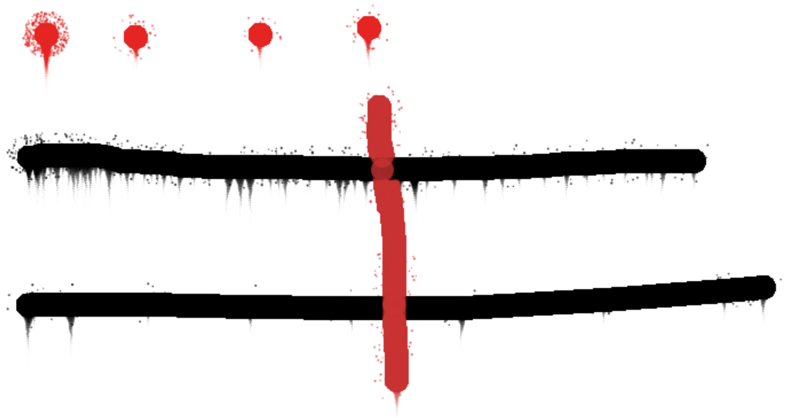


Figure 5.19: Paint effects test

5.2.7 Layers management using Nootropic's layers library

Nootropic's layers library [43] allows to create layers by extending the Layer class and to manage them in an AppletLayers object. Code snippet 5.20 is taken from the main setup() method. It shows how the layers are instanciated and then added to the layer manager. When the players are instanciated, the corresponding pointer layer and stencil layer are passed through the constructor to link them with the controller.

```
layers = new AppletLayers(this);
// Layer 0 : paint layer
PaintLayer paintLayer = new PaintLayer(this);
layers.addLayer(paintLayer);
// Layer 1 : player 1 stencil layer
StencilLayer stencilLayer1 = new StencilLayer(this);
layers.addLayer(stencilLayer1);
// Layer 2 : player 2 stencil layer
StencilLayer stencilLayer2 = new StencilLayer(this);
layers.addLayer(stencilLayer2);
// Layer 3 : player 1 pointer
PointerLayer pointerLayer1 = new PointerLayer(this,"Player 1");
layers.addLayer(pointerLayer1);
// Layer 4 : player 2 pointer
PointerLayer pointerLayer2 = new PointerLayer(this,"Player 2");
layers.addLayer(pointerLayer2);
oscP5 = new OscP5(this, 8000);
ipComputerSide = getIp();
p1 = new Player(1,(PointerLayer)layers.getLayer(3),(StencilLayer)layers.getLayer(1));
p2 = new Player(2,(PointerLayer)layers.getLayer(4),(StencilLayer)layers.getLayer(2));
```

Figure 5.20: Layers management with AppletLayers object

Paint layer The paint layer is unique. Both artists paint on it and can clear it with an animation simulating white paint dripping down from the top of the screen, filling it totally. This is made with a static image by updating it's y coordinates to make it go down each frame. If not in clearing mode, the layer is composed of a list of DrippingPoints and Lines which are rendered each frame. These primitives are placed by the artist when he paints : a DrippingPoint is placed for each (x,y) couple received from OSC module and a Line is drawn between last point and current one to make it smooth. The save method relies on the really convenient processing method "save(String filename)". Finally there is a tricky method which "applies" a stencil to the paint layer. Snippet 5.21 shows it. For each pixel of the mask (which contains the binarized stencil image plus the paint the user placed over it), a mask is applied to remove the stencil image selected by it's color. The alpha values are also filtered because of the antialiasing methods used by processing which could result in noise on the painting (mainly if the stencil has been rotated) and of course not to copy transparent pixels.

```
public void applyStencil(int[] maskPixels) {
        this.loadPixels();
        for(int i=0; i < maskPixels.length; i++){
                if(alpha(maskPixels[i])>175 && maskPixels[i]!=color(50,50,50,255)){
                        pixels[i]=maskPixels[i];
                }
        }
        this.updatePixels();
}
```

Figure 5.21: Stencils application method

Stencil layer The stencil layers are composed by three images which are superposed at each frame by the draw function : a background (the stylish design of the stencil), a canvas (where user paints) and a border (computed by an edge detector). Snippet 5.24 shows how these images are initialized :

- Background image is the stylish stencil image loaded from a file.
- Border image is a copy of the background image. It is first binarized based on the alpha values to a pink (RGB values 255,0,255) and transparent (pixel = 0) image.

Then an edge detector based on a 3x3 sliding window (figure 5.22) is applied to paint in gray the borders of the stencil. This is made to let the artist know where are the limits of the stencil even if he already painted over it (figure 5.23 sums it up).

- Canvas image contains the binarized image of the stencil in gray (RGB color 50,50,50) and transparent but will just print the user paint on screen.

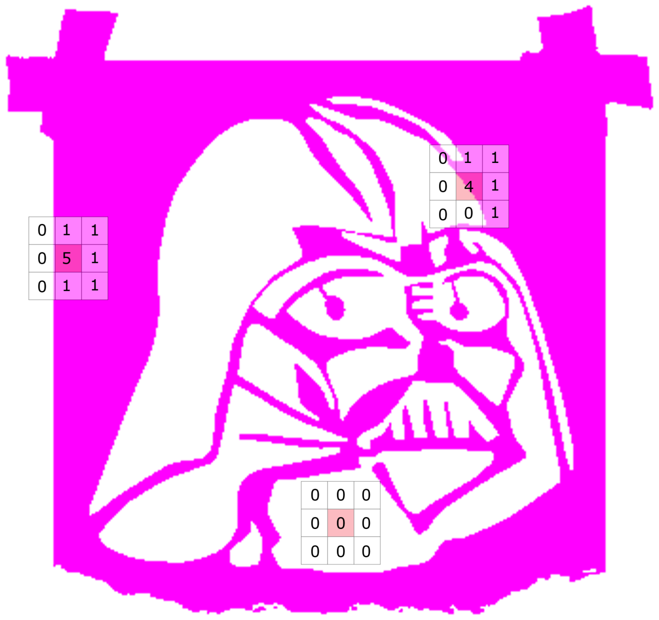


Figure 5.22: Sliding window based edge detector

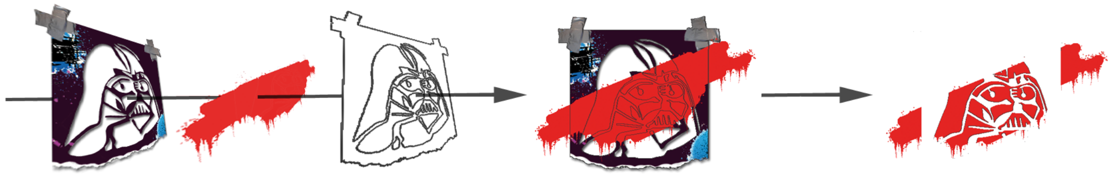


Figure 5.23: Stencil images superposition, rendering and result

Pointer Layer The last type of layers (which is on top of the layers stack) is the PointerLayer. They are used to show each player's position in real time and are just circles (same size as the brush to help the artist being more precise) identified by a label with the name of the player.

```
public void startStencil(String stencilName) {
  this.isSet = false;
  this.stencilBackground = loadImage(stencilName);
  this.stencilBorder = loadImage(stencilName);
  this.stencilBorder.loadPixels();
  for(int i=0; i<(this.stencilBorder.pixels).length; i++) {
    if(alpha((this.stencilBorder.pixels)[i])>200) {
      (this.stencilBorder.pixels)[i]=color(255,0,255,255);
    }
    else {
      (this.stencilBorder.pixels)[i]=0;
    }
  }
  for (int y = 1; y < this.stencilBorder.height-1; y++) {
    for (int x = 1; x < this.stencilBorder.width-1; x++) {
      float sum = 0;
      for (int ky = -1; ky <= 1; ky++) {
        for (int kx = -1; kx <= 1; kx++) {
          int pos = (y + ky)*this.stencilBorder.width + (x + kx);
          if(this.stencilBorder.pixels[pos]==color(255,0,255,255)) {
            sum++;
          }
        }
      }
      this.stencilBorder.pixels[y*this.stencilBorder.width + x] = (sum>0 && sum<4)?color(50,50,50,200):0;
    }
  }
  this.stencilBorder.updatePixels();
  this.stencilCanvas = loadImage(stencilName);
  this.stencilCanvas.loadPixels();
  for(int i=0; i<(this.stencilCanvas.pixels).length; i++) {
    if(alpha((this.stencilCanvas.pixels)[i])>200) {
      (this.stencilCanvas.pixels)[i]=color(50,50,50,255);
    }
    else {
      (this.stencilCanvas.pixels)[i]=0;
    }
  }
  this.stencilCanvas.updatePixels();
  this.isShown = true;
  this.x = (float) WIDTH / 2;
  this.y = (float) HEIGHT / 2;
  this.angle = 0.0F;
}
```

Figure 5.24: Stencils start method

Chapter 6

Testing

The implementation described the solution chosen to build this system of virtual graffiti. The next step is to submit the last prototype to a series of tests. The aim is to verify that the main functions of the system work properly and to ensure the usability fits the user's gesture and behavior.

6.1 Aim of the final test

This final test is to verify that the system answers the problematic of the project: to virtualize a graffiti. To do so, the test must check the four main axis defined in the introduction part:

- To replace the spray can by a device
- To render the graffiti on a surface
- To establish a link between the device and the rendering
- To make the system realistic

6.2 Experimental conditions

The test takes place in a movie room equipped with a white flat screen and a projector. To optimize the graphical rendering, the room is put in the complete dark. The additional hardware required for this test is an iPhone or iPod Touch equipped with an accelerometer and a gyroscope, a computer/laptop and a wifi router with an accessible wifi network. From the network point of view, two UDP ports are needed to established the communication between the mobile device and the computer side application. Five participants were volunteers to perform the final test. None of them assisted to a previous test of this project. Table 6.1 describes the experimental conditions of this final test.

Table 6.1: Final test experimental conditions

Place	Dark Room
Equipement	1 Computer 1 iPhone 4 1 Projector 1 Wifi router
Network configuration	2 Open UDP ports
Participants	Number: 5 Sex: 4 males and 1 female Age: from 21 to 24 3 of them possess an iPhone
Duration	45min

6.3 Process of the test

The test is divided into a series of tasks followed by a survey. The project group has a role of observer. The process of the test is described in the next paragraphs.

Basic shapes drawing The aim of this task is to check if the participant can draw the elementary shapes which compose a graffiti with the system. The participant has to draw the following shapes: a dot, a line, a circle, a square and a triangle. The participant has 5 seconds to draw the shape. The task is completed if the participant succeeds to draw the shape during this amount of time.

Stencils utilization The participant is now asked to use the stencils. He/She is not guided by the project group. The participant has first to select a stencil, to place it, to paint it and finally to find a way to come back to the normal painting mode. The observer validates the task if the participant succeeds to draw the stencil in a laps of time of 40 seconds.

Multi-user mode For this task, two participants are painting at the same time. They are free to draw anything they want, but they are asked to try to paint over the other participant's graffiti. They also have to use at least once the stencils at the same time so that two stencils are present on the wall. The duration of this task is 15 minutes long. This last task can be performed as a workshop. There is no completion criteria and the participants are asked to signal any anomalies during the utilization of the system.

Survey The last task is the survey. It sums up the previous tasks and asks the participants to give their opinion about the system. It is divided into three sections. The first one called "Painting function". Here the participants can evaluate the difficulty they had drawing the basic shapes. They can put a grade from 1 (difficult) to 5 (easy). Another section focuses on the stencils. The participants can say if they managed to use the stencils easily or not, following the same graduation. The last section is about the realism. The participants can evaluate the realism of the spray can, the rendering and the multi-user mode, giving a mark from 1 (not realistic) to 5 (really realistic).

6.4 Results

Concerning the painting test, the results illustrated by figure 6.1 are logical with the implementation issue. The participants did easily the simplest shapes such as the dot, the straight line and the square. The most difficult drawings are the shapes containing curved lines. The participants were 80% to have difficulties to draw the circle. This result is due to the technical issue encountered with the accelerometer. The algorithm in charge of canceling the rebound effect is checking when the accelerations are changing of direction. This works perfectly for a simple linear move cause it has a unique direction. The move needed to draw a circle includes two directions. So major parts of the circles drawn are actually semicircles. The rebound effect makes also the diagonal lines difficult to draw and 40% of the participants had trouble drawing it.

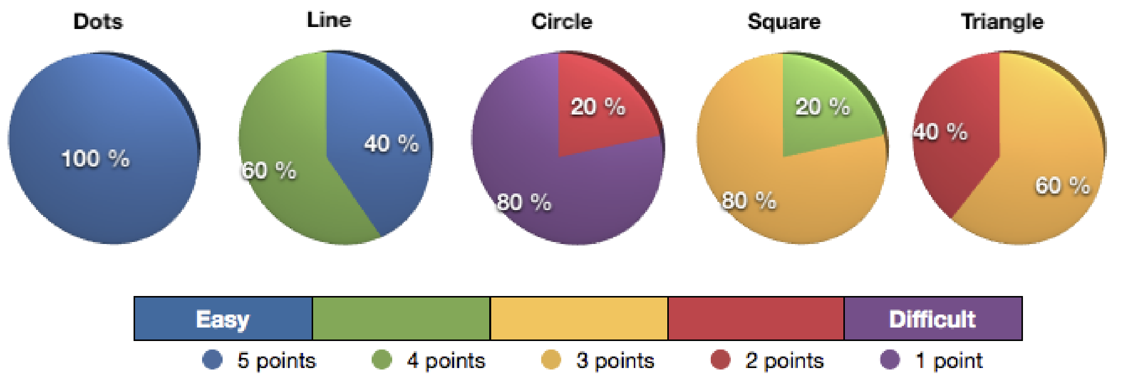


Figure 6.1: Painting test results

The test of the stencils utilization, illustrated by figure 6.2, shows some good results. Most of the participants (60%) found this function really easy to use. From the choice of the stencil until the placement and the rotation, the participants had no troubles to manage the stencils. The only hesitation was to come back to the painting mode. 80% of the participants hesitated to find the way to paint on the stencil the first time. Finally, after a few seconds of reflexion, every participants found the double tap action to come back to the paint mode. They declared that this action is not obvious but they point out that the double tap makes sense to validate when you are used to smart-phones or touch screens.

Participants expressed a positive opinion concerning the degree of realism of the system. Figure 6.3 illustrates the results of the realism test. Participants really enjoyed the realism of the spray can. Both spray and shake sounds contribute to forget the device according to all the participants. Also 100% of the participants enjoyed the graphical design with a spray can as background. Every participant gave the maximal note for every criteria concerning the realism of the spray can. The same result can be observed for the paint rendering. 100% of the participants liked the texture and the graphical effects applied to the paint. 60% of the participants would have preferred some effects when a layer of paint is covered by another one such as an alpha on the top layer which let the former stripe of paint visible. But this is contradictory with the real observations previously made in the

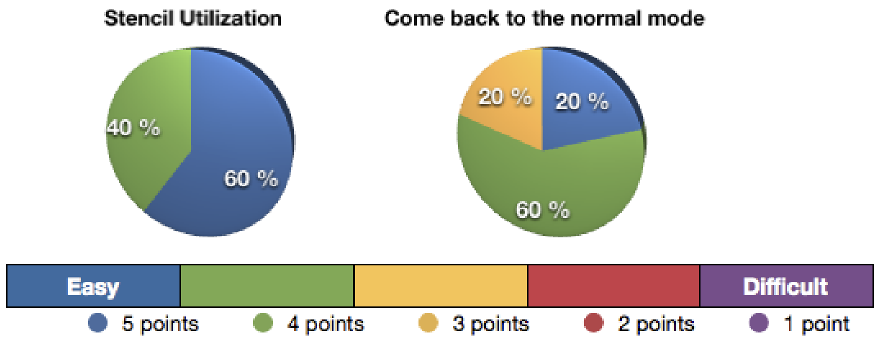


Figure 6.2: Stencil test results

analysis part. Participants liked and found realistic the way to clear the wall. However, 40% would have preferred some more spectacular graphical effects. Finally, the participants really enjoyed and found realistic the graphical appearance and the way to manage the stencils. The orientation of the stencil responded in real time to their moves and gesture.

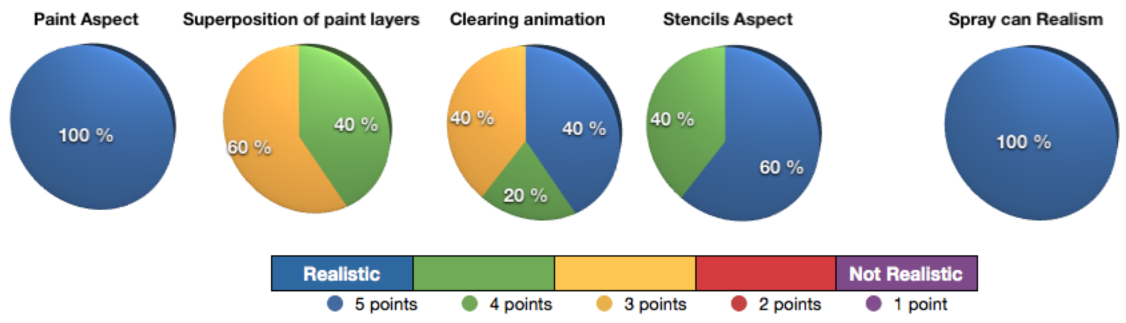


Figure 6.3: Degree of realism test results

Concerning the multi-user mode, most of the participants noted the same behavior as in the single user mode. The only thing the participants pointed out is some graphical mistakes during the utilization of the stencils. Sometimes the black outline stayed after the use of the stencil. That is the reason why 60% of the participants judged the stencil function in multi-user mode not realistic enough. Figure 6.4 shown the results of the multi-user mode.

Conclusion of the final test From the final test can be drawn several observations and conclusions. First of all, the previous results showed that the transformation of a device into a spray is done. The combination of the sounds, the visual aspect and the gestures transformed an iPhone into a virtual spray can. The second point is that the rendering into a virtual surface is also accomplished. The participants are unanimous about the paint effects such as drifts or spots. It is also possible to say that the link between the mobile device and the computer side application is fully functional. The communication protocol allows the participants to move and rotate the stencil in real time from the device to the painting wall. However, some technical problems interfere with the painting module. It is difficult to draw complicated shapes which include curve lines and circles. Here is the main

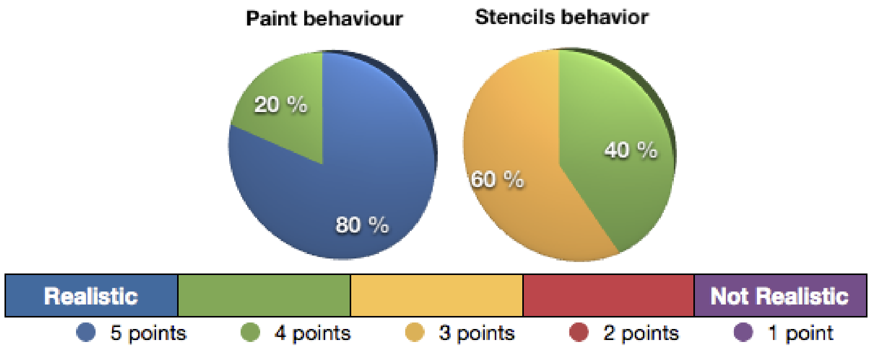


Figure 6.4: Multiuser test results

drawback of the system for now. Concerning the degree of realism, the participants are more than 80% to be satisfied by the system. The implementation of essential functions with realistic constraints (a limited number of colors) reminds the participants the realistic aspect of street art. The behaviors, gestures, sounds and graphical rendering are elements which bring the user to think he/she is using a realistic virtual system.

Chapter 7

Conclusion

7.1 Project achievements

After this semester, the project group managed to address most of the stakes stated in the introduction of this report :

- Replace the spray can by a device people are more likely to have with them when they go out
- Link the device to a computer side application
- Render a graffiti on a surface
- Make the system realistic

The spray can has been replaced by a smart phone, which most people always have with them nowadays. Moreover the smart phone can connect easily with the computer side application using a wireless network and OSC protocol. This communication is fast enough so we can consider the system to be in real time. Finally, the rendering solution projects a realistic virtual representation of the paint, with noisy paint dots, leaks, etc. However, there are still some problems with the accelerometers' data preprocessing. As discussed previously in the report, the group faced unexpected behaviors from the iPhone accelerometers resulting in rebounds on the acceleration graphs. This is a very important issue and will require further investigation.

7.2 Further developments

As stated in previous section, the project group came across an unexpected behavior regarding iPhone accelerometers. Despite the group's attempts to solve this issue it still has to be investigated and addressed (Kalman's filter didn't manage to remove this "noise"). However, a lot of other features could be improved in this project. First, the rendering of the paint could be even more realistic and interactive (optimization of the algorithms, introduction of new paint behaviors, etc.). Second, it would be smart to save the graffitis in GML [44] files so the other graffiti softwares can read it. And third, the community aspect of the project could be improved by providing a web plateform allowing users to share their paintings and even their home made stencils. It would also be possible to introduce a business model on this plateform by selling spray can refills, new colors and new stencils. The system is supposed to be installed in bars and other entertainement places (like amusement parks or modern art museums for example) so the business model could also involve these actors. From a different point of view, the application could be democratized so anybody could use it at home with their phone and personnal computer. In this last case, we could just sell the applications to the customer. Since the communication between the devices is defined in the protocol the team created (see 4.3 page 45), it is also possible to develop client applications on other devices than iPhones (Android would be a good start since it is one of the most widely spread smart phone OS together with iOS). The server side application is in Java and already works on Windows, MacOS and Linux.

7.3 Personnal achievements

From a personnal point of view, this project has been a perfect conclusion for the 10th semester. Indeed, the project group had to face problems broadly spread around the Vision, Graphics and Interactive Systems field. The knowledge acquired along the semester has been used to solve issues from design to implementation. The UCD (User Centered Design) technics were widely used to design the mobile application since it is the direct interface between the user and the system. The user should not need any technical knowledge to use an interactive system like this one, therefore it makes this approach really adapted in the context of this project. On a more technical aspect, the group had the occasion to discover and apply several data and image processing algorithms (acceleration preprocessing and convertion into position, edge detection, etc.). To conclude, this project has been raising keen interest from both developers and other participants so it is far from being over ! The group will keep on working and hopes this report will provide other developers good help.

Thank you for reading, project group 11-1024.

Bibliography

[1] Lascaux, 2011. www.lascaux.culture.fr.

[2] J. Scheible, "Mobispray.com, use your mobile phone as a spray can," 2008. http://www.mobispray.com/.

[3] T. O. Jorgen Scheible, "Mobispray: Mobile phone as virtual spray can for painting big anytime anywhere on anything," 2009. www.mediateam.oulu.fi/publications/pdf/1216.pdf.

[4] M. Venn, "Virtual graffiti," 2010. http://www.instructables.com/id/virtual-graffiti/.

[5] M. Lihs and F. Matuse, "Wiispray," 2007. http://www.wiispray.com/about/.

[6] Graffiti Research Lab, "L.a.s.e.r," 2007. http://graffitiresearchlab.com/projects/laser-tag/.

[7] Switch On The Code, "iphone tutorial - reading the accelerometer." http://www.switchonthecode.com/tutorials/iphone-tutorial-reading-the-accelerometer.

[8] W. Baxter, J. Wendt, and M. C. Lin, "Impasto: A realistic, interactive model for paint," 2004. http://gamma.cs.unc.edu/IMPASTO/publications/Baxter-IMPaSTo_Web-NPAR04.pdf.

[9] J. Stam, "Real-time fluid dynamics for games," 2003. http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/GDC03.pdf.

[10] V. Martins, "Gpu viscosity," 2010. http://www.pixelnerve.com/portfolio/2010/10/03/gpu-viscosity/.

[11] Nimblekit.com, "Tutorial 2: Navigation within application," 2010. http://www.nimblekit.com/tutorials.php?action=display&id=32.

[12] Oxford Dictionaries, "graffiti," 2011. http://www.oxforddictionaries.com/definition/graffiti?view=uk.

[13] Wallbomber, "History of street art," 2011. http://www.wallbomber.com/street-art.php.

[14] Rutgers, "Wax cylinder phonograph," 2010. http://edison.rutgers.edu/cylinder.htm.

[15] P. M. Ann Morrison and S. Viller, "Evoking gesture in interactive art," 2008. `www.hiit.fi/~morrison/gesture.pdf`.

[16] J. Scheible and V. Tuulos, *Mobile Python: Rapid prototyping of applications on the mobile platform (Symbian Press)*. John Wiley and Sons, 2007.

[17] A. Bensky, *Short-range wireless communication: fundamentals of RF system design and application*, vol. 1. Newnes, 2004.

[18] C. R. Ben Fry and volunteers, "Processing," 2001. `http://www.processing.org/`.

[19] M. Lihs, "Wiispray if," 2009. `http://www.wiispray.com/2009/07/wii-spray-honered-with-if-design-award/`.

[20] D. C. Jeffrey Rubin, *Handbook of usability testing*. John Wiley and Sons, 2008.

[21] Dimension Engineering, "A beginner's guide to accelerometers." `http://www.dimensionengineering.com/accelerometers.htm`.

[22] University of British Columbia, "Acceleration, velocity, and position." `http://www.ugrad.math.ubc.ca/coursedoc/math101/notes/applications/velocity.html`.

[23] K. Seifert and O. Camacho, "Implementing positioning algorithms using accelerometers," 2007. `http://www.freescale.com/files/sensors/doc/app_note/AN3397.pdf`.

[24] S. ChartSchool, "Moving averages," 2011. `http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_averages`.

[25] R. Don Kellogg, Senior Manager and T. N. C. Insights/Telecom Practice, "Iphone vs. android," 2010. `http://blog.nielsen.com/nielsenwire/online_mobile/iphone-vs-android/`.

[26] M. Murphy, *Beginning Android 2*. Apress, 2010.

[27] S. K. Prasant Mohapatra, *Ad-hoc networks: technologies and protocols*. Springer, 2005.

[28] Apple inc, "Cocoa," 2011. `http://developer.apple.com/technologies/mac/cocoa.html`.

[29] K. Scarfone, *Guide to Bluetooth Security: Recommendations of the National Institute of Standards and Technology*. DIANE Publishing, 2009.

[30] G. E. Clarke, *CompTIA Network+ Certification Study Guide*. McGraw Hill Professional, fourth edition ed., 2009.

[31] J. Huntington, *Control Systems for Live Entertainment*. Focal Press, third edition ed., 2007.

[32] M. Wright, "The open sound control 1.0 specification," 2002. `http://opensoundcontrol.org/spec-1_0`.

[33] www.tremble.org, "Vvosc objective-c osc wrapper," 2010. `http://www.trembl.org/codec/438/`.

[34] T. Cockshott, J. Patterson, and D. England, "Modelling the texture of paint," *Computer Graphics Forum*, vol. 11, no. 3, pp. 217–226, 1992.

[35] Blender.org, "Fluid simulation," 2011. `http://wiki.blender.org/index.php/Doc:Manual/Physics/Fluids`.

[36] 3DAliens, "glu3d for 3dsmax, maya and xsi," 2009. `http://3daliens.com/joomla/index.php?option=com_content&view=article&id=50:glu3dmaxmaya&catid=36:products&Itemid=66`.

[37] ACUSIM, "Acusolve," 2011. `http://www.acusim.com/html/acusolve.html`.

[38] V. V. Stephane Grumbach, Liying Sui, *Advances in computer science–Asian 2005: data management on the Web : 10th Asian Computing Science Conference.* BirkhLuser, 2005, illustrated ed., 2009.

[39] C. Chung, *Pro Objective-C Design Patterns for IOS.* Apress, first ed., 2011.

[40] iOS Developer Library, "Motion events," 2010. `http://developer.apple.com/library/ios/#documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/MotionEvents/MotionEvents.html`.

[41] R. M. Corless, *Essential Maple 7: an introduction for scientific programmers.* Springer, 2, illustrated ed., 2002.

[42] A. Schlegel, "oscp5," 2010. `http://www.sojamo.de/libraries/oscP5/index.html`.

[43] Nootropic design, "Processing layers," 2011. `http://nootropicdesign.com/processing-layers/`.

[44] U. Brandes, M. Eiglsperger, and J. Lerner, "Graphml primer," 2002. `http://graphml.graphdrawing.org/primer/graphml-primer.html`.

Appendix A

Low-Fi Testing: Survey

What is your name?

How old are you?

Gender?

What is your email address?

Do you have any artistic background?

Have you ever used a spray paint can?

What kind of phone do you own?

Do you generally feel at ease with smart phones?

How would you naturally hold the smartphone and where would you press? (If it can help you can draw it)

How would you like to manage options (changing color, shape, size...)?

Do you think that speech recognition could be relevant? To change options?

Would a bar with such a system interest you?

Would you be more interested in this system for it's realism or it's ease of access?

Would you like to share you graphs on a social network?

Do you have any other comments?

Appendix B

UI Low-Fi Prototype: Survey 2

What is your name?

How old are you?

Gender?

What is your email address?

Would it be important to see the shape of a spray can on the main painting view?

Would you prefer manage the drawing function via the phone or via the painting wall?

Would it be more realistic to manage the drawing functions via the phone or via the painting wall?

The access to specific functions such as to change color, to use stencils or to clear the wall should be:
A-Embedded on the design of the spray can
B-In a navigation bar
C-With independent buttons on the view

What will be the more realistic way for a graffiti maker to carry his/her tools?

Do you like the idea of the artist's bag where it would be possible to access to all the drawing functions?

Do you think a limited choice of color would be more realistic?

Would you enjoy to have an unlimited choice of color?

How many colors would be sufficient?

Do you like scrolling?

How many stencils would be useful?

What kind of icon would be more obvious for you to clear the wall?

Would you like to have a background on the wall or do you prefer a blank screen ?

Appendix C

Final Test: Survey 3

What is your name? ____________________

How old are you? ____________________

Gender? ____________________

I. Painting function Please, try to draw with the system the following simple shapes. Then, write down how difficult was it (1 = difficult 5 = easy).

- A dot: 1 - 2 - 3 - 4 - 5
- A line: 1 - 2 - 3 - 4 - 5
- A circle: 1 - 2 - 3 - 4 - 5
- A square: 1 - 2 - 3 - 4 - 5
- A triangle: 1 - 2 - 3 - 4 - 5

II. Stencils function Now, try to use the stencils. Pick up one of the stencils from the stencil library and paint over it. Once this is done, come back to the normal painting mode.

Is it easy to use the stencils? (1 = difficult 5 = easy) 1 - 2 - 3 - 4 - 5

Did you manage to come back to the simple painting mode easily?

____

____

III. Degree of realism This system is a simulation of real street art: Graffiti/Stencils paintings. The following questions will ask you to express your feelings about the realism of this virtual version of graffiti pantings. Mark the degree of realism of these followings elements (1 = Not realistic 5 = Really realistic).

Spray Can/iPhone

- Spray sound: 1 - 2 - 3 - 4 - 5
- Shake sound: 1 - 2 - 3 - 4 - 5
- General aspect: 1 - 2 - 3 - 4 - 5

Wall/Rendering

- Paint aspect: 1 - 2 - 3 - 4 - 5
- Superposition of paint layers: 1 - 2 - 3 - 4 - 5
- Clearing animation: 1 - 2 - 3 - 4 - 5
- Stencils aspect: 1 - 2 - 3 - 4 - 5

MultiUser Mode

- Paint behavior: 1 - 2 - 3 - 4 - 5
- Stencils behavior: 1 - 2 - 3 - 4 - 5