TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ÁLVARO GARCÍA - MOYA HERRERA

MULTIPLATFORM MOBILE SOFTWARE DEVELOPMENT

Master of Science Thesis

Examiners:
Professor Mikko Tiusanen and
MSc Juha-Matti Vanhatupa
Examiners and topic approved by
the Faculty of Computing and
Electrical Engineering Council
03.04.2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Faculty of Computing and Electrical Engineering

GARCÍA-MOYA HERRERA, ÁLVARO: Multiplatform mobile software development
Master of Science Thesis, 49 pages, 35 appendix pages
April 2013

Major subject: Multiplatform software development
Examiners: Professor Mikko Tiusanen
MSc Juha-Matti Vanhatupa

Keywords: Multiplatform software development, educational mobile game, game design, game development.

This thesis considers how to achieve efficient multiplatform mobile software development. A mobile application is developed as a vehicle to demonstrate how this can be done in a particular instance. The educational mobile game is focused on road safety for pedestrian, mostly children.

The thesis is divided into three parts. In the first section, issues related to multiplatform applications are explored. Second chapter explains about educational game, alongside the target platforms selected and the integrated development environment chosen. Finally, the evaluation process is described, where the performance of the educational game is tested on the target platforms and the results are shown.

PREFACE

During the seven years that I spent between my Bachelor and my Master degree, I have never been sure if I wanted to be a computer engineer. I guess that is normal for someone who always questions any kind of job, looking for advantages and disadvantages. There were only two things that I was sure about when I started my studies. One was that I wanted to study engineering and the other was that I wanted to go abroad as an exchange student.

Thus, I moved to Finland in 2011 to finish my degree at Tampere University of Technology. For my Master's thesis I wanted to work on something that I would be proud of when it was finished. In that sense, I felt lucky to meet my supervisors, prof. Mikko Tiusanen and assistant professor Juha-Matti Vanhatupa, because together we found a great idea as topic for my thesis. I would like to thank them for their support and patience during feedback sessions.

The greatest gratitude is for my family, my girlfriend Patricia, and my friends. Without them, none of my plans would be possible. Firstly, my family and friends supported my idea of going to Europe to study. My family encouraged me to do it by any means necessary. Then, Patricia helped and supported me over these two years in any way she could.

Tampere, April 23th, 2013

Álvaro García – Moya Herrera

LIST OF ABBREVIATIONS

API	Application Programming Interface. An interface for software components to communicate to each other.
CSS	Cascading Style Sheets. A style sheet language used for describing the presentation semantics of a document written in HTML.
EC-n	Evaluation criterion.
ECA-n	Evaluation criterion of Acceptance level.
ECD-n	Evaluation criterion of Domain technique.
ECF-n	Evaluation criterion of Functionality level.
ECI-n	Evaluation criterion of Integration level.
ECP-n	Evaluation criterion of Performance level.
ECR-n	Evaluation criterion of Regression level.
ECS-n	Evaluation criterion of Load/Stress level.
ECUI-n	Evaluation criterion of User Interface level.
ESA	Entertainment Software Association. Trade association of the video game industry in the United States.
HTML	HyperText Markup Language. Markup language for creating web pages.
IDE	Integrated Development Environment. A software application that provides facilities to computer programmers for software development.
IP	Internet Protocol.
MEI	Mobile Entertainment Industry.

MVC	Model View Controller. A software architecture pattern that separates the representation of the information from interaction with it.
NFR-n	Non-Functional Requirement
NFMR-n	Non-Functional Maintainability Requirement
NFPR-n	Non-Functional Performance Requirement
NFPOR-n	Non-Functional Portability Requirement
NFPOLR-n	Non-Functional Policy Requirement
NFRR-n	Non-Functional Reliability Requirement
NFSR-n	Non-Functional Security Requirement
NFSAR-n	Non-Functional Safety Requirement
NFTR-n	Technical Environment Constraints
NFUR-n	Non-Functional Usability Requirement
OS	Operating System. Software that manages computer hardware resources and provides services for programs.
SDK	Software Development Kit. Software development tool that helps to create applications for a certain software package or framework.
UC-n	Use Case
UCD	Use Case Document
UR-n	User Requirement
UFR-n	User Functional Requirement
XML	Extensible Markup Language.

CONTENTS

1. INTRODUCTION

Technology has become an integral part of our lives. Computers have been accepted as a way to educate the new generations. Mobile phones are more important for some people than caffeine or exercise. According to Van Dillen et al. (2012), the number of mobile devices in use exceeds the number of people with shoes.

Smartphones have become attractive and common as personal devices, surpassing the former mobile phones. As smartphone and tablet use rises, companies are developing mobile applications for customers and end users. According to Eddy (2011), there were more than 550 million mobile Internet users in 2010, and the number is expected to surpass 1 billion by the end of 2013. Users of all these mobile devices, like iPhone (Apple 2013), iPad (Apple 2013), Blackberry (Blackberry 2013), Android (Android 2013), Windows Phone 7 (Microsoft 2013), or even eReaders (Schuessler 2010), create a never-ending demand for more and more unique and useful mobile applications. Users want mobile applications to be simple and fast. Just one bug or usability issue can spoil the entire experience.

Mobile gaming is what people use their phones for most of the time, over checking the weather, maps, or social networking. Between 70% and 80% of all downloads are games, 37% of users of iPhone play daily, and 84% of tablet owners play games (Corasaniti 2010). That is why the Mobile Entertainment Industry (MEI) has grown 63%, in terms of worth, from 2010 to 2014. One reason for the increased interest towards games is the increasing popularity of games. The Entertainment Software Association (2011) stated that gamers are not children or teenagers exclusively. The average of a gamer now is above 30 years old. Games have grown in complexity and keep growing as the technology advances. Games include features like artificial intelligence in the game characters, networking or sophisticated engines (Blow 2004). However, Bethke (2003) noted a trend that the most successful games only have a few gameplay elements that have been extremely well tuned.

According to Janssen (2011), the next step, in terms of revenue, in gaming is about in-game purchases. Those are game items or points that a user can purchase for use within a virtual world to improve the character or enhance the playing experience. In-game purchases are expected to reach $11 billion by 2015.

This thesis will consider multiplatform mobile software development. An educational mobile game will be created oriented to road safety hazards for children, as a vehicle to the thesis. The game will be implemented using Appcelerator Titanium as the integrated development environment (IDE). Then the performance will be tested over two target platforms: iOS 6 (Apple 2013) and Android OS 4 (Android 2013). The testing process and all the experiences about the platforms, the IDE, and the mobile game will be reported.

The thesis is structured in five parts. Chapter 2 considers effective multiplatform software development more closely. In Chapter 3, the educational mobile game developed is explained. Chapter 4 presents the design of the mobile game. In Chapter 5, the technologies used in this thesis are explained. Chapter 6 exposed the implementation of the game, relating the game design with the technologies used. Chapter 7 covers the evaluation process of the application, before conclussions.

2. EFFECTIVE MULTIPLATFORM MOBILE SOFTWARE DEVELOPMENT

Dallera (2011) states that implementing software on mobile devices is hard, everything is more complicated to accomplish than it is on the web or on the desktop. Since the platforms are fairly new, development tools and frameworks still have some room for improvement. Nowadays, if a customer needs an application to run on several platforms, the development team is forced to implement a different application for each platform. This is hard, because every small change or update of the application needs to be made in each of the native environments.

A common debate between developers is on which solution is better, native or multiplatform environment. When implementing on native environment, applications are fast and there is full access to the resources of the device. Those native applications are available on online stores, like Apple Store (Apple 2010) or Google Play, formerly known as Android Market (Google 2012). However, native applications are expensive to develop and it is mandatory to pass an approval process to be published. On the other hand, implementing on multiplatform environment provides a set of advantages. Like native applications, multiplatform applications get access to the resources of the device, are quick, and are available on online stores. Unlike native applications, the cost of the development process on multiplatform applications is reasonable.

Table 2.1. *Summary of Native versus Multiplatform environments*

Feature	Native environment	Multiplatform environment
Accessibility to resources	Full	Full
Average performance speed	Fast	Fast
Online stores	Available	Available
Development cost	High	Reasonable
Approval process	Mandatory	Customer's decision

As Table 2.1 depicts, both options provide a set of similar benefits, and users usually do not distinguish between them. According to Viswanathan (2012), multiplatform applications seem to be popular, because business of any size is able to offer products and services on mobile devices, under a reasonable cost, instead of implementing in the native environments, where the costs rise.

The idea behind most multiplatform frameworks is to limit development time, and the ideal solution is to write once and run everywhere, so in that sense, multiplatform development helps (DuPont 2012; Warren 2012). Developers can save time by coding only in one programming language, which can be compiled for and executed in several platforms. Figure 2.1 illustrates the process of how a multiplatform framework actually works. It manages a unique source code, handling the database of devices, in case the application needs it. Later, through the framework, the source code is built and executed over any platforms that the customer requires.

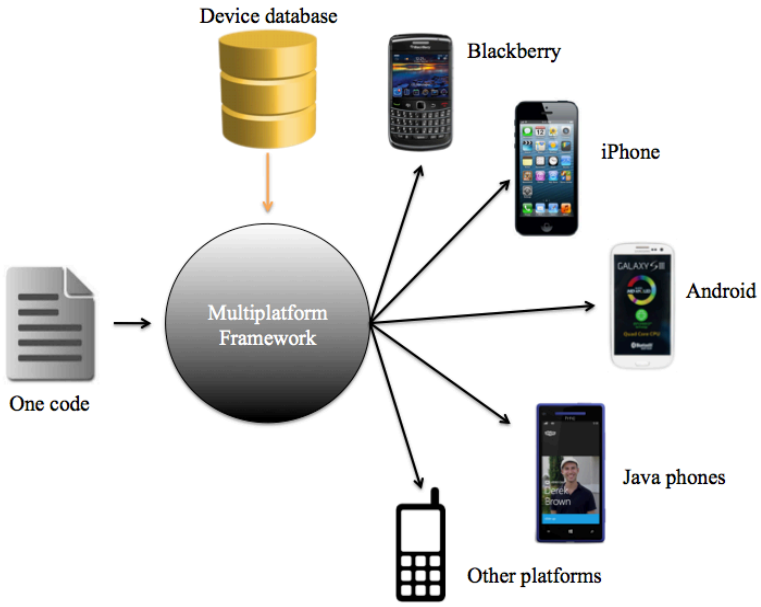

Figure 2.1. *Multiplatform Framework Process*

Multiplatform environments may be divided into two categories. The first category requires building individually for each platform that it supports. This does not mean that there is a native environment. It means that even using a multiplatform environment; developers have to find a different way to implement the code for each. For example, developers create functions, one per each platform that produces the same result but

differently. The other category can be executed on any platform without specific preparation. An example of the second category is a source code implemented in an interpreted language, like JavaScript, or a pre-compiled portable bytecode. For those source codes, the interpreters or run-time packages are standard components of all platforms supported.

Developing on multiplatform environments carries some problems, such as needing to adapt to the different sizes of the screen or the lack of enough memory to run applications. The main aspect about multiplatform applications is to create a single source code that is available to execute on several platforms. Another important issue is to investigate what are the functional differences of the code in each platform. This thesis considers how to efficiently develop multiplatform mobile software, that is, investigates how to develop mobile application for different platforms only once, without using native environments.

The approach to the problem is constructive. It is going to present and evaluate an example implementation of an educational game, which executes using a multiplatform framework. The evaluation of the thesis provides results from the tests driven over the application. Those results demonstrate some differences in performance on different platforms.

3. EDUCATIONAL GAMING FOR TRAFFIC SAFETY

Schofer et al. (1995) found that a majority of child pedestrian crashes involved a sudden appearance of the child pedestrian moving quickly across the street: this suggests that children cross the road at unsafe locations. In the same way, Forsythe and Berger (1973) reported that the reason for unsafe pedestrian crossings was mainly time-related. A need to hurry or a desire to keep moving was the main reason behind the lack of caution. Hamed (2000) concludes that pedestrians' expected waiting time is influenced by the number of attempts needed to successfully cross the street.

According to Brewer et al. (2005), pedestrians did not always wait to cross the street when all lanes were completely clear, rather, they anticipated that the lanes would clear as they crossed and used a "rolling gap" to cross the street, a separate gap for each lane of traffic coinciding with the pedestrians' path across the street. Table 3.1 summarizes the conclusions of the previous studies.

Table 3.1. *Summary of safety pedestrian studies*

Author	Ideas
Schofer et al. (1995)	Majority of pedestrian crashes involve a child moving quickly across the street.
Forsythe and Berger (1973)	Reason for unsafe crossings was mainly time-related.
Hamed (2000)	Pedestrians' waiting time has profound influence on the number of successful attempts to cross.
Brewer et al. (2005)	Pedestrians anticipate that the lanes would clear, crossing through rolling gaps.

Thus, it is necessary to educate children to be aware of the threats of the roads. Safety education can be one of the best ways to arm children against traffic hazards.

Safety tips often given to children for crossing the street involve very easy-to-understand directions.

Hochbaum (2000) gives a set of specific strategies to parents about teaching their children, ages 8 to 9: to cross at a green light, look in both directions of the road before crossing, not to cross between parked cars, and to avoid streets with heavily traffic or difficult intersections. Parents and educators should be the best to educate children about safety pedestrian education, but they may assume that finding a safe place to cross is a relatively simple task when in fact it is not intuitive for young children. Parents often overestimate children's' abilities. Dune et al. (1992) examined parental expectations of their children's knowledge and road safety behavior. They found that parents expected their 5- and 6-year-old children to be as proficient in knowledge and behavior as 9-to-10-year-old children. Rothengatter (1981) found that, in general, video training improved children's knowledge of safety but did not change their behavior. In video training children watch educational videos about road safety, about dangerous situations and its consequences.

However, Thomson et al. (2005) developed a computer training program where children aged 7, 9, and 11 participated in four training sessions with an adult trainer and two other children. In the computer program, the children would guide a character through a neighborhood, and when it was necessary to cross the street, the children would press a "go" button when they thought it was safe to do so. When it was safe, the character crossed and when it was not safe, the image would freeze. The goal of the trainer was to listen children's reasoning about why they chose to make the incorrect decision. The trainer guided their thinking in the appropriate direction, and avoided imposing solutions.

Table 3.2. *Summary of educational programs and studies*

Author	Ideas
Hochbaum (2000)	Provide a set of strategies to parents about teaching children to cross safely.
Dune et al. (1992)	Examined parental expectations of children's knowledge and their behavior on the street.
Rothengatter (1981)	Found that video training did not change children's behavior.
Thomson et al. (2005)	Developed a computer-training program for children.

In order to change children's behavior regarding road safety, one approach, suggested by the studies summarized in Table 3.2, could be training sessions relying on games. Gaming helps young people to learn through practice, they learn how to fail, overcome that failure, and succeed. In fact, according to Pivec (2009), a game environment provides the motivation necessary for persistent re-engagement by a player and hence achieves the practice-makes-perfect scenario.

Educational gaming is a topic where many developers want to make a difference, and nowadays, there are many options available for children. Educational gaming has found a great opportunity to spread knowledge and tips for children through Internet. There are a few online games where the main objective is to lead the main character from a point A to a point B safely.

The online game "3M Streetwise" (Figure 3.1 left) provides the user with a set of predefined actions to choose. It does not let the user choose what to do using the same options as in real life, such as crossing the street over with red light on for pedestrians or to wait before the green light appears (Streetwise 2006). The game is implemented in Adobe Flash (Gay 2001), formerly known as Macromedia Flash, a software platform used for authoring vector graphics, animation, and games.

The video game "Frogger" (Konami 1981; Figure 3.1 center) known worldwide has as goal to bring frogs, main characters of the game, safely to their homes, avoiding cars, trucks, or other animals. This is not a reasonable option to train children about safety

when crossing roads, because the objective in this game is to avoid the hazards by advancing in rolling gaps, which is dangerous as noted above.

The game “Stop, Look, and Listen” (Tales of the Road 2009; Figure 3.1 right) is an online game based on stopping the main character, observing, and crossing at the moment the user believes it is right, otherwise, the character is knocked down by a car and thrown far away, and the user has to try again.

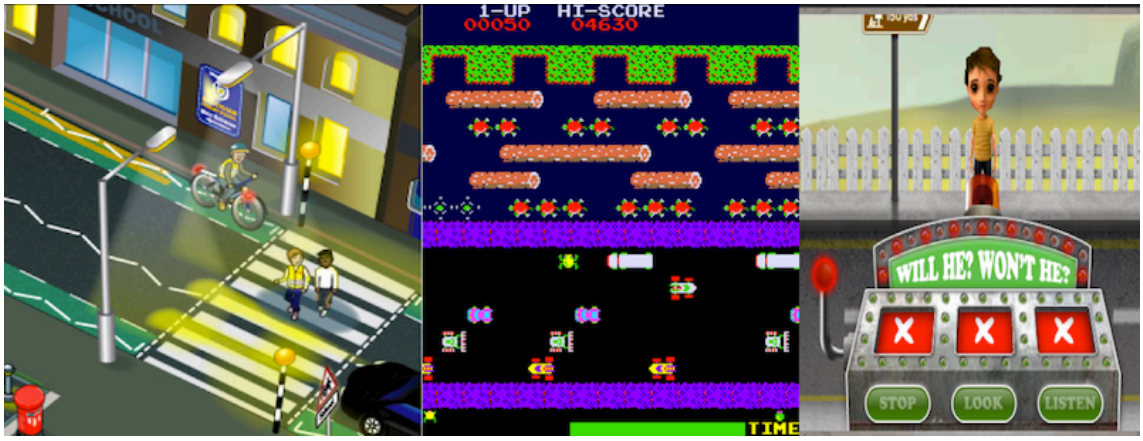

Figure 3.1. *Safety pedestrian education online games*

None of these educational games illustrated in Figure 3.1 are ideal, because none of them provide the freedom to the user to fully control the character, interacting with the game and the environment created. They focus on providing a set of predefined actions where the user can only choose one option out of three. That means that the learning process in the same for all the users, conducted by the developer, who is in charge of deciding what is important for the user and what is not.

A game, which is educational about road safety, has to be able to merge gaming with education in an approach attractive to the user. The game has to provide more freedom to the user, giving a wider selection of possibilities, making it evident that there are places where vehicles have serious dificulties to stop. Through practice, users should become more cautious when they cross roads in their daily life. In this way, a user would get an understanding of the relation between cause and effect, good or bad. In that sense, the user should experience real situations and acquire implicit knowledge about them.

4. TIWALKINGSAFE

The mobile application developed in the thesis is a mobile game oriented to pedestrian safety education. It is called TiWalkingSafe and it is placed in a 2D worldview, to make it as simple as possible for the user to get in touch with the environment. The game is not time-related: there is no time limit to perform actions or going to some point of the world. The reason because the game is not time-related, is to teach the user that there should not be hurry when he or she is going to cross the road, in real life.

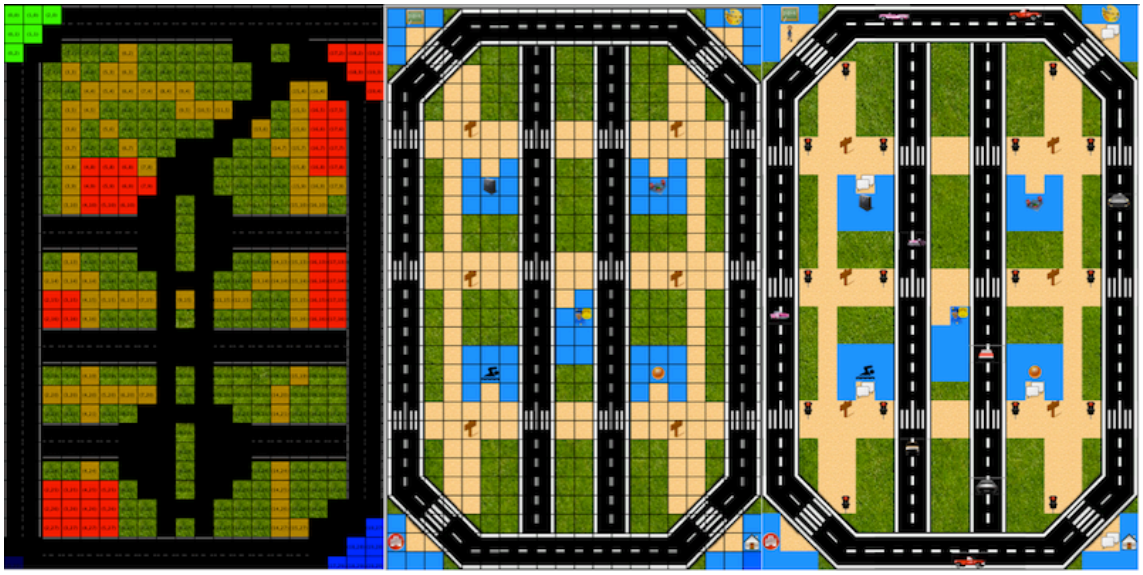

Figure 4.1. *Evolution of game world*

Figure 4.1 illustrates the evolution of the game world. The first design, after the paper prototypes depicted in Appendix I, is the image of the prototype in the Tiled Map Editor (Lindeijer 2008). It has only a rough design of the buildings, roads and crosswalks. In the second design, the roads have lines and the corners are more detailed. There are crosswalks for pedestrians at some points and there are also designed places, like shopping center, school, or home, where the user shall guide the character. In the last iteration of the design, the final version, there are cars and traffic lights placed in the

world and also the initial spot where the player starts the level. The initial spot for the player is in the upper left corner of the image, in front of the school, where the game starts.

Every level has “synchronization points” and the goal is to collect them. Each synchronization point contains new instructions for the user to go to a new address. In order to collect these points, the user has to keep the character safe from the hazards of the game. When the user brings the main character in front of the synchronization point, the user has to click on the screen where the point is placed, and if the main character is placed correctly, the user receives new instructions, if not, the user has to bring the character to the correct location.

In that way, the game is intended to allow users to interact with the game and learn about road safety, how traffic lights work, and avoid putting themselves in dangerous situations. The mission of each level is quite standard, the character is requested to go somewhere to complete a task and return home safe. As in any game, there is a story to motivate the user to play. In this case, the story is about a 7- to 9-year-old character, illustrated in Figure 4.2, who has to walk to different places after school, before returning home for dinner. The character has to accomplish a set of activities, like going the supermarket, then going to basketball practice, or picking up his younger relative from somewhere and returning home safely. In order to do that, the user leads the character through the world designed, which contains standard road hazards, such as cars, depicted in Figure 4.3, pedestrian crosswalks, and traffic lights.

Figure 4.2. *Game main character*

Figure 4.3. *Game cars*

The game promotes knowledge about road safety to users who spend enough time practicing with it. Users learn when they fail, in particular about why they fail and how they can avoid that failure in the future.

5. TECHNOLOGIES

TiWalkingSafe is the educational mobile game implemented in this thesis. It is developed using JavaScript as the programming paradigm. There are two target platforms where the performance of the game is analyzed, Apple OS (Apple 2013) and Android OS (Android 2013). The integrated development environment of the multiplatform development framework employed called Appcelerator Titanium (Appcelerator 2006) is also described.

5.1. Apple OS

Apple OS (Apple 2013), previously named iPhone OS and currently known as iOS, is developed and distributed by Apple, Inc. It is a proprietary operating system that supports Apple devices, such as iPhone, iPad, iPod Touch, or Apple TV, on a per-device basis, meaning that it is dependent of the Apple device. Apple does not license iOS for installation on non-Apple devices. Apple iOS is a mobile version of Apple's OS X operating system (Apple 2002). As such it has a UNIX (Ritchie & Thompson 1978) basis, like OS X has. Currently the last major update is iOS 6, released in September 2012. The user interface of iOS is based on the concept of direct manipulation, which uses touch inputs and transforms them into actions and requests, using some internal controls. Figure 5.1 illustrates a few examples of direct manipulation gestures. The first image is the gesture of clicking at some point on the screen. The second gesture is to zoom in and out at a picture or a website, done dragging two fingers. Finally, the last gesture is to move an object using three fingers (Caughey 2012). Apart from the gestures illustrated in Figure 5.1, there are other gestures such as double-tap, (to double click) or tap-and-move, which is to make a drag-and-drop action (Caughey 2012).

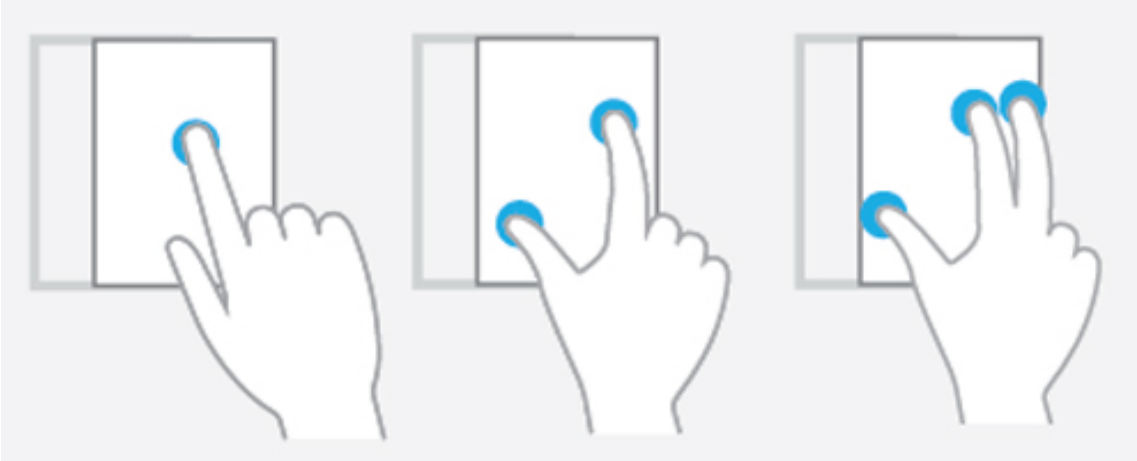

Figure 5.1. *Direct Manipulation Gestures (Caughey 2012)*

The applications developed for iOS are written in Objective-C, a high-level, object-oriented programming language that adds messaging to the standard C programming language. Xcode became the development environment for iOS, with the release of Xcode 3.1. (Brocklehurst 2010)

After the release of iOS 4, multitasking is no more limited to a selection of applications, as it was before. Multitasking can now work on several applications such as background audio, voice over IP, notifications, task completition, or fast application switching.

A grasp of the iOS architecture helps to understand how it works. Figure 5.2 provides a graphical overview of the architecture of iOS. Each layer of the iOS architecture has a different function. The application layer is where applications installed into iOS are running. Under it, there is the layer of frameworks and the Application Programming Interface (API), used for communication between software components. Below it, there is the layer comprised of Objective-C and C libraries. Then, there are the kernel, drivers, and services of iOS. The last layers are the ARM processor (Masterton 2011), the firmware, and the hardware of the device. When the hardware makes a change in a register, each layer receives the update from the previous one and communicates it to the next layer till the information reaches the application layer.

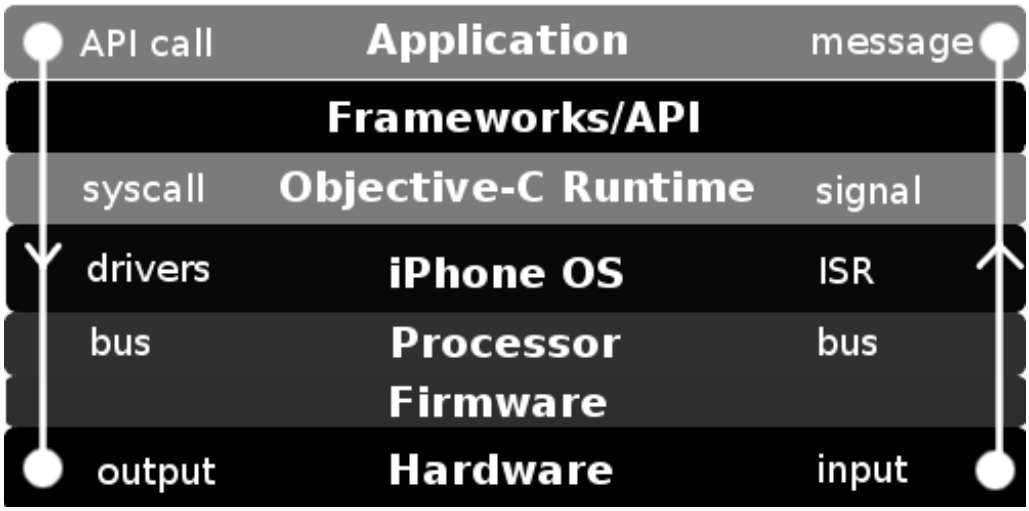

Figure 5.2. *Overview of iOS Architecture (Silicon News 2012)*

Lynn (2012) conducted an analysis of applications that run over iOS, presented in Table 5.1. These applications are the best-known features of the system, like the possibility of launching applications by voice commands, making a video conference over the cellular network, reading a website in offline mode, 3D mapping, or sharing photo streams.

Table 5.1. *Analysis of iOS 6 (Adapted from Lynn 2012)*

Feature	iOS
Launch application by voice	Native application
Launch third-party applications by voice	Native application
Automotive integration	Yes
Message replies for incoming calls	Yes
Set call reminders	Yes
Video calling over cellular network	Yes
Offline reading of websites	Yes
Share photo streams	Native application
Store and access tickets	Native application
Navigation	Turn-by-turn
3D Mapping	Apple Maps
Information about nearby businesses	Yes

Apple iOS has native applications for launching application by voice, it has automotive integration, and it is possible to make a video call over a cellular network. In the same way, iOS allows the user to reply with messages to incoming calls, and set call

reminders, so the user can call someone in the future. The navigation is turn-by-turn, that is directions for a selected route are continually presented to the user in the form of spoken and visual instructions (Button & Hensher 2001). In the same way, it has 3D Mapping using Apple Maps (Apple 2013), and it is possible to get information about nearby businesses. Apple iOS includes native applications for sharing photo streams and for storing and accessing tickets, like flight or movie tickets.

5.2. Android OS

Android OS (Android 2013) is a Linux-based (Linux 2009) operating system designed primary for touchscreen mobile devices, such as smartphones and tablet computers. According to Elgin (2005), it was initially developed by Android, Inc, which Google bought in 2005.

Android is open source and its permissive licensing allows the software to be modified and freely distributed by device manufacturers, wireless carriers, and developers. Latest stable version is Android OS 4.2.1, Jelly Bean, released in November 2012. The user interface of Android OS is based on the same concept of direct manipulation as iOS. However, the architecture of Android is different from iOS (Figure 5.3).

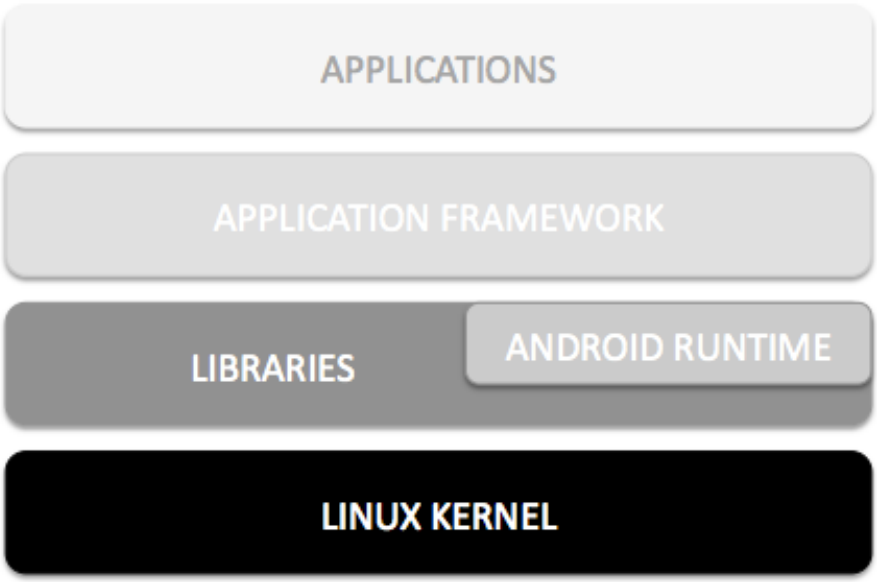

Figure 5.3. *Overview of Android OS Architecture (Adapted from Kann 2010)*

Android OS is a software stack, where each layer is a group of several program components. Each layer provides services to the layer just above it and receives requests from the layer below it. The top layer in the architecture is the applications layer. Applications, like a web browser, SMS client application, or the contact manager, are in this layer. Just below it, there is the application framework layer, which includes a set of managers. Those managers control the basic functions of the device, like resource management, location, and voice call management. The next layer, below the application framework, is the Android runtime layer. This layer consists of a virtual machine and Core Java libraries. Although these libraries provide most of the functionalities defined in the Java core libraries, they are different from these and Java ME libraries (Oracle 2013). Below, there are the native libraries of Android, which enable the device to handle different types of data, such as SQLite databases (SQLite 2010) or OpenGL (OpenGL 2013). The base layer is the Linux kernel. It interacts with the hardware and contains all the drivers, programs that control the hardware. It also manages the memory, processes, networking, and security.

It is clear that Android OS architecture is similar to iOS architecture, but the difference remains in the hardware. Unlike iOS, completely dependent on the hardware, Android OS cannot take advantage of all the different resources and capabilities of each device, because it can run on different devices. Lynn (2012) concluded that both are very similar in terms of usability, but he pointed out that the hardware is an important issue. He analyzed on Android OS the same set of features that he analyzed on iOS, in order to find some differences between them. Table 5.2 depicts the results of the analysis.

Table 5.2. *Analysis of Android OS 4 (Adapted from Lynn 2012)*

Feature	Android OS
Launch application by voice	Third-party application
Launch third-party applications by voice	Third-party application
Automotive integration	Yes
Message replies for incoming calls	Yes
Set call reminders	No
Video calling over cellular network	Yes
Offline reading of websites	Yes
Share photo streams	Only Samsung Galaxy S III
Store and access tickets	No
Navigation	Turn-by-turn
3D Mapping	Google Maps
Information about nearby locations	Yes

Android has less native applications than iOS, because third-party companies or developers make them. Android OS incorporates automotive integration, video calling over cellular network, and offline website reading, as iOS does. However, Android OS is not able to set reminders for future calling or store and access tickets. One feature that is more complete and useful for iOS is Google Maps, which works better than Apple Maps (Chan 2012).

5.3. IDE: Appcelerator Titanium

An IDE is a software application that provides facilities to software developers for their work (Nourie 2005). It usually consists of a source code editor, build automation tools, which run scripts such as compiling source code or packing binary code, and a debugger, a tool that helps test and examine source code.

A suitable IDE for this thesis had to be multiplatform, able to emulate the target platforms, iOS and Android OS, and use same common source code. According to Hay (2012), there are ten solutions for creating cross-platform mobile applications, which are Sencha Touch 2 (Sencha 2008), jQuery Mobile (jQuery 2009), Tiggzi (Exadel 1999), AppMakr (AppMakr 2012), iBuildApp (iBuildApp 2010), Widgetbox (Flite 2009), foneFrame (Azalea 1992), PhoneGap (PhoneGap 2004), Appcelerator Titanium

(Appcelerator 2006), and appMobi XDK (AppMobi 2012). These require initial deep knowledge about HTML (HyperText Markup Language; Berners-Lee 1990), CSS (Cascading Style Sheets; W3C 2013), Javascript (Flanagan 2006), and, sometimes, about jQuery (jQuery 2009) or XML (eXtensible Markup Language; W3C 2013), to develop an application using the previous development environments. The exceptions are Appcelerator Titanium and PhoneGap, which only require knowledge about Javascript. A deciding factor between the two last environments is that PhoneGap is oriented towards business applications, since Adobe had become its sponsoring organization, and Appcelerator Titanium had been created as a game designer tool. As the application was an educational mobile game, Titanium was likely to be a more suitable IDE to work with. It is a platform for developing mobile, tablet, and desktop applications using web technologies, introduced in December 2008. All application source code gets deployed to the mobile device where it is interpreted using a Javascript engine (Flanagan 2006).

The architecture of the Appcelerator Titanium IDE is depicted in Figure 5.4. It is divided into four layers, the top one being the application coded in JavaScript or HTML. Below the application layer, there is the Titanium software development kit (SDK), a set of software development tools that allow the creation of applications, and the API libraries. The next layer is the mobile operating system, like iOS or Android OS, or the browser, for mobile web applications.

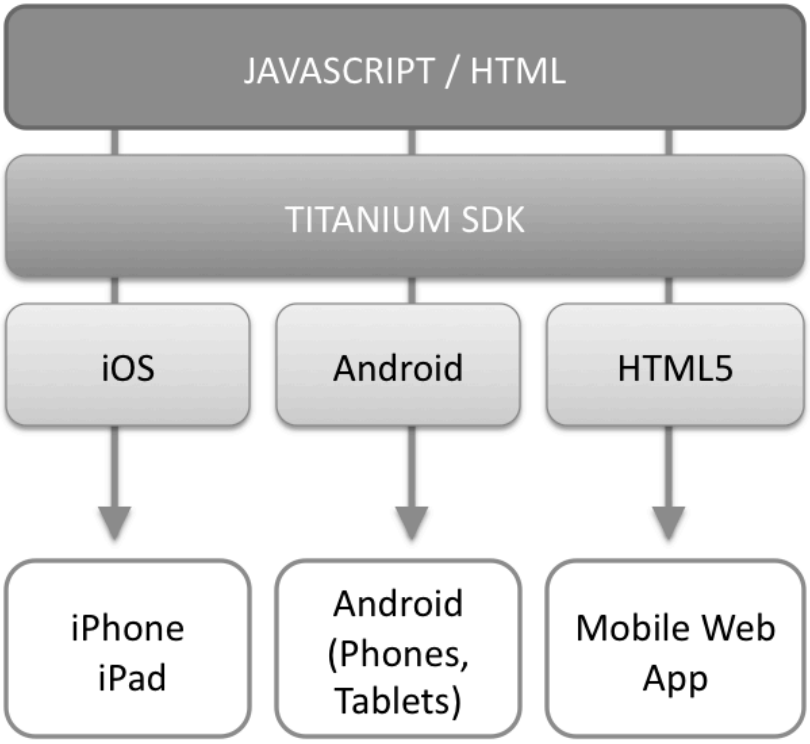

Figure 5.4. *Overview of Appcelerator Titanium Architecture (Titanium 2006*)

Appcelerator Titanium allows rapid prototyping, is web-oriented, and uses a common programming paradigm, namely Javascript. It is a cross-platform environment for development, and it has a growing community, around 200,000+ developers sharing their knowledge and with more than 35,000 applications. On the other hand, there are also a few disadvantages, like the growing complexity directly linked to the complexity of the application, the bigger the application gets the hardest is to keep simple the code, or the increasing difficulty to get free modules for developers. Despite the disadvantages, Appcelerator Titanium was chosen as the IDE to create the mobile game for this thesis.

6. GAME IMPLEMENTATION

TiWalkingSafe is implemented in the integrated development environment called Appcelerator Titanium, in JavaScript, and it is executed on emulators for iOS and Android OS. The implementation process of the game starts with the design of the interfaces of the game, Figure 6.1, till the creation of the engine that moves the cars through the world and changes the color of the traffic lights.

Figure 6.1. *Examples of interfaces of the game*

The architecture of the game is the Model-View-Controller (MVC; Reenskaug et al. 2008) architecture. The view layer is the user interface of the game, Figure 6.2, the controller layer is in charge of transforming the user's inputs into actions or requests for the game, and the model layer is going to control all the changes produces by the user's actions. Through Titanium, the three layers are implemented using JavaScript.

The game has been developed using layers of objects, as depicted in Figure 6.2. These layers are an easy way to implement the central engine of the game. In fact, there is no need to use pre-built modules provided by Appcelerator, which in some cases, are old modules that do not work with the current versions of the emulators. The idea is to create a stack of layers, where each layer is above the previous. Then, all of them are combined to create the final view of the game, as Figure 6.3 depicts. For example, as depicted in Figure 6.2, the lowest layer is the map of the world. The map is inactive, so it does not perform any action or movement. Then there are cars placed over the map, moving on the roads. Over the cars, there are traffic lights placed, on the map, trying to avoid setting a car and a traffic light in the same spot. Over them, lay the synchronization points and, finally, the main character layer on top of them, but it interacts with the rest of the layers below, especially with cars and synchronization points. The user controls the main character, but he does not control the cars or traffic lights, thus those features have to be in different layers. It is the user who interacts with a synchronization point, clicking it, not the main character.

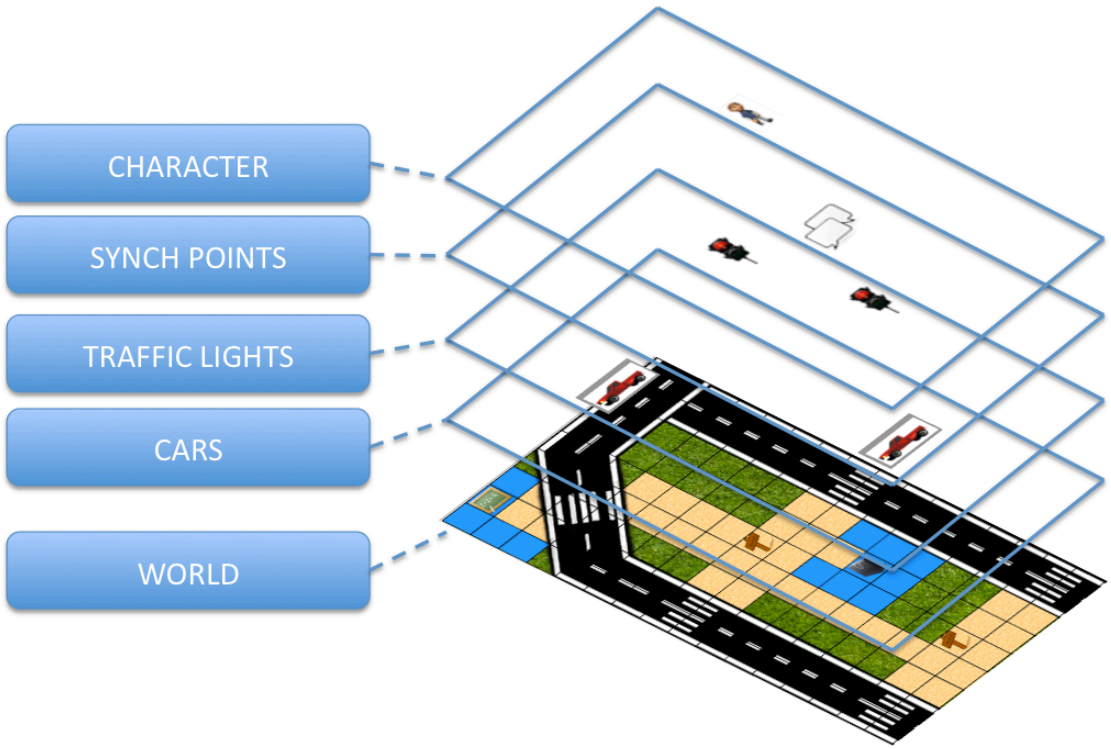

Figure 6.2. *Layer architecture of the game.*

Cars and traffic lights are independent entities of the game. They proceed driven by the game engine, not by the user. Cars are implemented to stop when the traffic light is green for pedestrians, when it is red for cars. Cars drive on the right side of the road, like for 66% of world population (Lucas 2005). The traffic lights change their color every predefined amount of seconds from red to green.

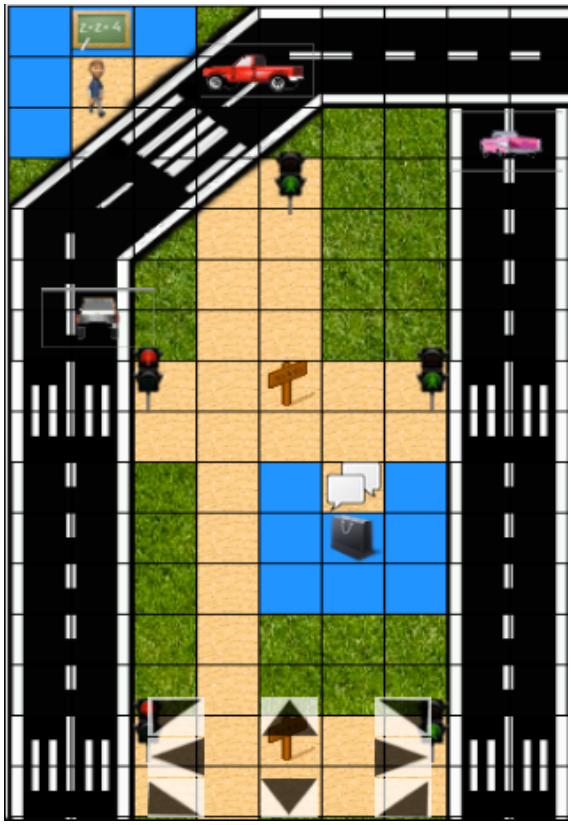

Figure 6.3. *Example of the initial level of the game*

As it is explained before, in each level of play the user has to handle the main character to collect synchronization points, distributed in the world, while avoiding dangerous situations. In order to reach safely the synchronization points, the user shall use the pedestrian road-crossings when the traffic light is green for pedestrians. When the user considers that the main character is placed correctly near the point, the user interacts with the game, clicking on the point, and if the character's location is correct, the user receives new instructions and the map is cleared.

The user controls the main character through the set of buttons on the lower part of the game interface (Figure 6.3). The set of buttons, which direct the character up, down,

left, right, or in diagonal, are at the top of the layer architecture. The set of buttons is divided in three columns, and between those columns, there are small spaces: this is the only way to get the eight-button set in one layer without overlapping. When the user has driven the main character to the synchronization point and touches the screen where the symbol of the point is placed, the game checks if the location of the main character is correct and if so, the map is cleared and the user receives a message with the next address where to go. If not, the game shows a message to the user that the main character is not in the right place to receive the information.

Appcelerator Titanium does not require HTML skills, because the user interface and the engine of the game are implemented using JavaScript. Following, there are examples of the code, represented using JavaScript or pseudocode. All the interfaces of the games are windows, which is how Titanium represents interfaces. A set of variables, such as buttons, image views, text fields and labels, are added to those windows. Each variable has its own characteristics, like size, position or, title (Program 6.1). The attributes written in percentages, like size or position of the object, are percentages of the size of the screen.

```
var start = Ti.UI.createButton
({
        title : “START”,
        top : ‘70%’,
        width : ‘70%’,
        height : ‘25%’
});
```

Program 6.1. *Creation of the button Start.*

In order to show the first window, or interface of the game the function open of the window needs to be called. On the one hand, iOS needs to create a navigation group to handle the navigation between interfaces. Then, it loads the initial interface as the window of the navigation group (Program 6.2). The rest of the navigation is based on opening the new interface through the navigation group (Program 6.3). On the other hand, Android OS only opens the new interface and closes the previous one (Program 6.4), but the developer has to remember which close it when necessary. The navigation group

of iOS manages the interfaces automatically. Program 6.2 and 6.3 are only executed when the application is running on iOS emulator, and Program 6.4 is executed only on Android, but the codes are meant to do the same, being the navigation controller for the application.

```
var navGroup = Ti.UI.iPhone.createNavigationGroup
({
        window : index
});
```

Program 6.2. *Creation of Navigation Group.*

```
button.addEventListener("click", function(e)
{
        navGroup.open(new_window);
});
```

Program 6.3. *Navigation through iOS.*

```
button.addEventListener("click", function(e)
{
        new_window.open();
        current_window.close();
});
```

Program 6.4. *Navigation through Android OS.*

In each interface of the game there is at least one button. To capture the action of pushing the button, Titanium provides event listeners, as in Program 6.3 and 6.4. The system activates these listeners when a button is pushed. Sometimes, those actions are only to change the interface for a new one. Inside the game interface, the directional buttons are ready to receive a click to move the main character in a specific direction. Titanium states that the position x, y of the character has two different values. The x-location is called left, and the y-location is called top. When the user pushes the right button, the x-location of the main character increases by 10 pixels, so the character moves forward to the right side and, position on the x-axis is increased by 10. If the user

pushes the down-and-right button, the x-location and y-location of the main character increase by 10 pixels each one, so the character moves on both axes, x and y, and their values are increased by 10 each (Program 6.5).

```
var moveDownRight = (function() {
        player.top += 10;
        player.left += 10;
});
```

Program 6.5. *Movement of the main character.*

Once the button to start the game has been pushed, the rest of the objects are placed over the layer of the world: the main character, the synchronization point, the cars, and traffic lights. The user does not control the cars and the traffic lights. For traffic lights, the behavior dictates that every 3 seconds its color has to change from green to red or vice versa (Program 6.6). For cars, it is more complicated. Each car has to move forward each second. However, the car needs to check the current direction in advance, because it may have to turn to another direction along the road (Program 6.7).

```
var changeLightColor = (function() {
        // Loop over all the traffic lights
        for(i to all_lights){
                // If the light is red
                if(i is red){
                        // Change to green
                        change i to green;
                } else {
                        // If not, then is red
                        change i to red;
                }
        }
});
```

Programme 6.6. *Pseudocode of behavior of the traffic light.*

```
var aiCars = (function(car, dir, i) {
        check_direction(car, x, y, dir, i);
        // If dir is 14, 1 means Up
        // and 4 means Right
        if(dir == 14) {
                // Move Up-Right
                car.left += 10;
                car.top -= 10;
        }
        ...
});
```

Program 6.7. *Behavior of the cars.*

Once the user has taken the main character to the synchronization point and clicks it, the synchronization point compares the x- and y-location of the main character with the x- and y-location of the point. If the main character is far away from the synchronization point, the function shows an error message to the user, asking to get closer to the synchronization point. If the main character is next to the synchronization point, the functions shows an acceptance message to the user, explaining where the main character is and what is the new address to go to. After the message, the world is cleared (Program 6.8). Cars are stopped and traffic lights do not change their color anymore, because the user passed the level and there is a new level available.

```
var synchronize = (function() {
        if (player.top == point.top &&
        player.left == point.left){
                alert("Congratulations!...");
                clearInterval(aiLights);
                clearInterval(aiCars);
        } else {
                alert("Content blocked!...");
        }
});
```

Program 6.8. *Behavior of the synchronization point.*

7. EVALUATION

Evaluation of the thesis consists on a set of criteria, transformed in test cases, evaluated over the performance of the mobile game executed on the emulators of iOS and Android OS. The results are presented in tables, followed by the validation of the testing process and the summary of results.

Mobile application testing is more complex than desktop or web application testing, due to the mobile nature of the device when compared to computers. Mobile software testing involves four main factors, shown in Figure 7.1. These factors are: the scope of the test process, the different levels that a test case can have, the environment around the test process, and the different techniques necessary to do the test process (Selvam & Karthikeyani 2011). The approach in the present evaluation will be based on these factors, so they will be presented as guidelines.

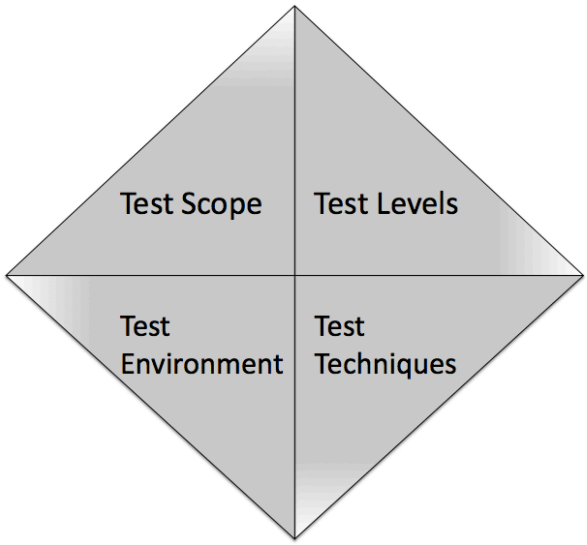

Figure 7.1. *Mobile Application Testing (Selvam & Karthikeyani 2011)*

Emulator testing is considered to be cost effective and useful to cover a wide range of devices and different screen resolutions, but it is not a realistic testing process. Testers have to assume that emulator is using the computer resources, not the resources that

an actual device has. Testing on real devices is considered to be the most realistic solution but is a costly solution. The ideal solution is to start initial phase of the testing on emulator and end on the real devices. (Selvam & Karthikeyani 2011)

The levels of testing consist of integration, user interface, regression, and acceptance testing. The first level is integration testing, which tries to find defects in the interfaces between components and in the interaction between modules. The next level is user interface testing, which is done through touch screens, screen orientation, and shortcuts to the application. The next level is regression testing, which is based on executing corrected parts of the code that contained errors before. Its goal is risk management, because sometimes a fixed code produces more problems than it solves. The last level of testing is acceptance testing, which verifies that the application fulfills the requirement specifications. Users run or test the application to ensure this level. (Selvam & Karthikeyani 2011)

The test scope is based on the test levels, consisting on performance, functional, and load/stress testing. In terms of performance testing, mobile carriers can affect usability and speed of the software application. The ideal approach would be to test the software application in different devices, using carriers from different countries. One of the most essential testing procedures is to verify the basic functionality of the software application. In some cases, the functions may be device-oriented, so in those cases testing should occur as soon as possible, in the early phase of development. The last issue related to the test scope is load/stress. Over the years, mobile software has started using more and more memory and other resources. The best way to stress testing on the mobile application is by performing repeated operations at different speeds, very quickly or very slowly. (Selvam & Karthikeyani 2011)

The last factor in mobile application testing is the technique used to carry out the tests. Techniques can be manual, scenario, and domain testing. Manual testing is based on interaction between tester and device, where tester follows step-by-step instructions and verifies the result. Although sometimes, manual testing process is tedious for the tester. Scenario testing is based on designing as many situations as testers could imagine happening with the application. For example, one case scenario could be having multiple applications running, press the home key, and return to the target application, ob-

serving the results of the pause in the execution. The last technique is domain testing, which is based on variables, like inputs or outputs. Testers try to find errors with different input parameters. (Selvam & Karthikeyani 2011)

7.1. Quantitative results

The evaluation process took place in an emulator environment, because testing on the device is excluded from this thesis, due to high cost. The author performed the tests. The quantitative results of the evaluation process are displayed in tables, using the template in Table 7.1. The results of the evaluation are provided in several tables, from Table 7.2 to 7.9. There is a table for each testing level, scope or technique analyzed.

Table 7.1. *Evaluation results template*

Reference	Description	Rank	Score	Final score
EC-n	Description of test case	M, D, or O	0.1 to 1	0.1 to 5

Table 7.1 displays the set of features stored for each test case executed over the mobile application. Column “Reference” assigns a unique identifier to the test case, followed by column “Description”. Next, there is a rank assigned for each test case, mandatory “M”, with a value of 5, desirable “D” of 3, or optional “O” of 1. Then there are the score in the testing process and the final score, Rank times Score. If the test case is performed perfectly, the Score value is the maximum value, 1, but if the test case has a low performance, the Score is 0.1. Finally, the final score could be in bold type if the score is better in one platform than in the other.

Table 7.2. *Evaluation results on integration testing*

Reference	Description	Rank	Platform	Score	Final score
ECI–01	The application processes any user input.	M	iOS	1	5
			Android	1	5
ECI-02	The application protects user's information.	D	iOS	0.1	0.3
			Android	0.1	0.3
ECI-03	The application protects user's points.	D	iOS	0.5	1.5
			Android	0.5	1.5
ECI-04	User's profile is only accessed through device.	O	iOS	0.1	0.1
			Android	0.1	0.1

Table 7.3. *Evaluation results on performance testing*

Reference	Description	Rank	Platform	Score	Final score
ECP–01	Time to load the application under 5 seconds.	O	iOS	1	**1**
			Android	0.1	0.1
ECP-02	Time to load the application under 10 seconds.	D	iOS	1	**3**
			Android	0.1	0.3
ECP-03	Time to load the application under 15 seconds.	M	iOS	1	**5**
			Android	0.1	0.5
ECP-04	Time between click and response under 15 ms.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECP-05	Time between click and response under 30 ms.	D	iOS	0.9	**2.7**
			Android	0.1	0.3
ECP-06	Time between click and response under 50 ms.	M	iOS	1	**5**
			Android	0.1	0.5

Table 7.4. *Evaluation results on acceptance testing*

Reference	Description	Rank	Platform	Score	Final score
ECA–01	The application manages new users.	O	iOS	1	1
			Android	1	1
ECA-02	The application keeps the synchronization points taken.	D	iOS	0.5	1.5
			Android	0.5	1.5
ECA-03	The application manages a game world.	M	iOS	1	5
			Android	1	5
ECA-04	The application provides actions to the user.	M	iOS	1	5
			Android	1	5
ECA-05	The application manages cars.	M	iOS	1	5
			Android	1	5
ECA-06	The application manages traffic lights.	M	iOS	1	5
			Android	1	5
ECA-07	The application fits the description provided.	M	iOS	0.7	3.5
			Android	0.7	3.5

Table 7.5. *Evaluation results on functionality testing*

Reference	Description	Rank	Platform	Score	Final score
ECF–01	The navigation control of the application works.	M	iOS	1	5
			Android	1	5
ECF-02	The controller button-set of the game works correctly.	M	iOS	1	5
			Android	1	5
ECF-03	The application remains in the same interface if the user does not change it.	M	iOS	1	5
			Android	1	5
ECF-04	The application does not navigate to an interface when it is not supposed to.	M	iOS	1	5
			Android	1	5
ECF-05	The application has the required setup of the game.	D	iOS	0.5	1.5
			Android	0.5	1.5
ECF-06	The application does not perform tasks it was not designed to do.	D	iOS	1	3
			Android	1	3

ECF-07	The application needs GPS.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF–08	The application needs Wi-Fi connection.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF-09	The application synchronizes with Facebook and Twitter.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF-10	The application works with peripherals.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF-11	The touchscreen supports double-tap gesture.	D	iOS	0.1	0.3
			Android	0.1	0.3
ECF-12	The touchscreen supports touch-and-hold gesture.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF-13	The touchscreen supports drag-and-drop gesture.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF-14	The application is accessible from the device desktop.	M	iOS	0.7	3.5
			Android	0.7	3.5
ECF-15	The application changes with the orientation of the device.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF–16	The application includes a user manual.	D	iOS	1	3
			Android	1	3
ECF-17	The application does not lose information if it is interrupted.	D	iOS	0.1	0.3
			Android	0.1	0.3
ECF-18	The notifications of the application are clear and visible.	M	iOS	1	5
			Android	1	5
ECF-19	The notifications of the application can be responded.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECF-20	The application provides appropriate error messages.	D	iOS	0.5	1.5
			Android	0.5	1.5
ECF-21	The application includes a time limit for log-in.	O	iOS	0.1	0.1
			Android	0.1	0.1

Table 7.6. *Evaluation results on user interface testing*

Reference	Description	Rank	Platform	Score	Final score
ECUI–01	The controller button set is user-friendly.	M	iOS	1	5
			Android	1	5
ECUI-02	The resolution of the objects is acceptable for the user.	M	iOS	1	**5**
			Android	0.1	0.5
ECUI-03	The shortcut of the application is clear on the Home screen.	D	iOS	0.7	2.1
			Android	0.7	2.1
ECUI-04	The learning process for playing the game is less than 15 minutes.	M	iOS	1	5
			Android	1	5
ECUI-05	The user figures out how to play the game without manual.	D	iOS	0.5	1.5
			Android	0.5	1.5
ECUI-06	The user assumes knowing how to interact with the system, without instructions.	D	iOS	0.1	0.3
			Android	0.1	0.3
ECUI-07	It is possible to navigate creatively around the application, going from one interface to any other.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECUI–08	The error messages are helpful for the user.	M	iOS	0.5	2.5
			Android	0.5	2.5
ECUI-09	The error messages adhere to good practices.	M	iOS	1	5
			Android	1	5
ECUI-10	The error messages of the application are also oriented to security.	D	iOS	0.1	0.3
			Android	0.1	0.3
ECUI-11	The application follows design guidelines for a particular platform.	O	iOS	0.1	0.1
			Android	0.1	0.1
ECUI-12	The application has big icons to provide user-friendly navigation.	M	iOS	1	5
			Android	1	5

ECUI-13	The application keeps the same interface-design all the time.	M	iOS	1	5
			Android	1	5

Table 7.7. *Evaluation results on domain testing*

Reference	Description	Rank	Platform	Score	Final score
ECD–01	The application does not allow incomplete information as input.	D	iOS	0.1	0.3
			Android	0.1	0.3
ECD-02	The application hides the password information.	M	iOS	0.1	0.5
			Android	0.1	0.5
ECD-03	The application avoids sharing user's information.	O	iOS	1	1
			Android	1	1

Table 7.8. *Evaluation results on load/stress testing*

Reference	Description	Rank	Platform	Score	Final score
ECS–01	The application is loaded correctly each time.	M	iOS	1	5
			Android	1	5
ECS-02	The application works with multiple hits on the buttons.	D	iOS	0.1	0.3
			Android	1	**3**
ECS-03	The application performs correctly when there are quick hits and slow hits.	D	iOS	0.7	2.1
			Android	1	**3**
ECS-04	The application performs correctly when being used for long time, more than an hour.	M	iOS	1	5
			Android	1	5
ECS-05	The application is pushed to the limits and still works correctly.	O	iOS	0.1	0.1
			Android	1	**1**
ECS-06	The application notifies errors when it is needed.	D	iOS	1	3
			Android	1	3
ECS-07	The application behaves properly when the user is switching between applications.	D	iOS	1	3
			Android	1	3

Table 7.9. *Evaluation results on regression testing*

Reference	Description	Rank	Platform	Score	Final score
ECR–01	The application behaves better after a change in the code is made.	M	iOS	1	5
			Android	1	5
ECR-02	All the errors have been corrected in the evaluation process.	D	iOS	0.3	0.9
			Android	0.3	0.9

7.2. Validation

All too often, testing is thought of as being entirely planned and predictable, full of scripts and plans. However, in this evaluation, the testing process tries to evaluate problems that can be explained and measure without scripts, to provide valuable information that enables reaching a valid conclusion. The result of this testing process cannot be predictable, because each platform has its own behavior.

In that way, there are some aspects of the evaluation process that can be misunderstand or perceived as threats. In the first place, the testing technique used is manual testing in 80% of the cases, for both platforms. This technique is tedious for testers, but it is also a great opportunity to clarify the result of each test case in the application, instead of visualizing a result provided by the computer. If the technique is automatic, there is a possibility to introduce errors in the code of the test case that provides a wrong score on the evaluation. In the rest of the testing process, the techniques of scenario and domain testing has been applied also, but only in 20% of the evaluation, because those techniques focus on only a few features of the application.

Another aspect is that the testing environment, as explained before, is not the best option, because it needs testing in real devices. At the same time as the evaluation, the emulator has been analyzed, trying to find out whether there are exactly as a device or different. However, the emulators have a good performance on the evaluation process. In fact, the emulators shall access to a limited portion of the resources of the computer,

to simulate the resources of the device, and apparently the emulator do it that way, but this is not possible to verify it.

User interface testing and functional testing covered from 60 to 70% of the time in the evaluation of a mobile application. Thus, the design of a test case has to take into consideration this aspect, to avoid focusing only on evaluating the user interface and functionality of the application. In this thesis work, acceptance testing and integration testing are important features of the evaluation. These are designed to evaluate whether the application fulfills the system in Appendix 1 and 2.

7.3. Summary

Table 7.10 illustrates the final comparison between the results obtained in the evaluation of the application over iOS and the results of the evaluation with Android OS. The total score is obtained as the sum of the final score of each test case evaluated.

Table 7.10. *Comparison between results obtained with platforms*

Platform	Total score
Apple iOS	150.9
Android OS	136.8

It is clear that the evaluation score obtained with iOS is slightly better than the evaluation obtained with Android OS. The maximum amount of points that can be reached in the evaluation is 205. In percentages, iOS had 74% of success in the evaluation and Android OS had 67% of success. In that sense, the success of iOS executing the application is not huge over the performance of Android OS with the application, because there is only a 7% difference between them. The following explores the differences in the results.

The first difference noticed on the evaluation is the loading time of the game. Android OS needs more time than iOS to load the game. That is related to the emulator, because the emulator of Android is heavier than the emulator of iOS. For Android, it

takes much longer because it creates features like activity threads, analytics, or power management. And then, when the game starts, the emulator of Android checks in each movement of the player, all the objects of the world, the cars, the traffic lights, the main character. On the other hand, iOS does not check all the objects in each movement, only at the beginning of the game and in the ending.

There is a difference between the resolution of the screen of iOS and the screen of Android. The screen of Android is bigger than the screen of iOS, so the resolution of the objects, like the main character, the cars and traffic lights, are not accurate. In fact, it seems to be out of proportion, but it could be correct if the dimensions of the objects were in percentage, instead of using static proportions. There is a problem with using percentages, because the movement of the objects would be affected, making them inaccurate.

Despite the fact that the emulator for Android OS is slower than the emulator for iOS, once loaded, Android OS provides service even if the user hits the buttons of the controller multiple times. This reaction of the user is odd, but often users are so focused on the game that they try to finish the level earlier, so they push the buttons of the controller multiple times in fast sequences. Usually this behavior is not covered in the evaluation, and the result is that the application stops or shuts down because it is unable to handle all the requests caused by the user.

Unlike Android OS, iOS does not provide mobility to the character if it received multiple hits on the buttons. The emulator of iOS takes a couple of seconds to react and keeps running. When this test case was being evaluated, the main character of the game did not move from the previous place to the new one. That could mean that iOS would not store the requests to perform them later. It may get the newest one and handles it.

Acceptance testing and integration testing get the same amount of points for both platforms, because these test cases are related to the application and the requirements. In that sense, it is understandable that the score is the same in both platforms, but acceptance and integration testing are needed in any mobile software evaluation process. Finally, regression testing reached successful results for both platforms. This testing feature is important because a poor regression testing results means that when an error in

the application is fixed, a set of new errors arise and the actual code carries more failures than the former code.

8. CONCLUSIONS

Multiplatform applications are the next step in the mobile software development environment. It is the cheapest option, in terms of resources and time, for individuals or small businesses that want to offer their products and services. However, there is a disadvantage in multiplatform development. The bigger the application is, the harder it is to keep the same coding for each platform, because each one needs different implementations, meaning that it costs more time and resources than it saves. If the application is complex, developers have to find new ways of coding, which using native environment is not necessary. As mentioned before, this thesis considers how to develop effective multiplatform mobile software. In that sense, the multiplatform tool provides well-enough features to implement the software, because the application was designed as simple as possible.

Mobile games have grown in complexity and keep growing as the technology advances. However, there are successful games that include only a few gameplay elements, extremely well implemented. The educational mobile game developed in this thesis is a reasonable solution for pedestrian safety education. The game is simple and small, but it provides all the basic functionality to recreate the reality of road safety, and enough to test how it works and provide results for this work. In case of a larger game, maybe native environment application are better options, because at the end developers save more time than using multiplatform environments, where, the programming language used not always fulfill the expectations or support the functionality required from the developer.

To evaluate the application is a tedious task. Therefore, the evaluation is complete if the test cases take place also in real devices, because using emulator. There is always a small chance that the result is not entirely truthful. In terms of emulator, iOS works significantly better than Android OS, it loads faster, and it does not check a lot of unneces-

sary features of the emulator. However, it is a surprise that Android OS is able to manage quickly repeated pressing of the direction buttons and move the character correctly. On the other hand, iOS is able to provide a good view of the game to the user, because Android OS generates a disproportionate view of the game, in terms of the size of the objects, like cars, traffic lights, and player, around the game world.

In conclusion, the thesis reached the goals stated at the beginning successfully. It evaluates a multiplatform mobile application on two target platforms, Apple iOS and Android OS, and concludes that provided a better general performance, which was Apple iOS. The multiplatform mobile application is an educational game, which apart from being a vehicle to the thesis has its own value as an educational tool for children.

Further analysis about multiplatform mobile applications would be useful to complete the conclusions provided in this work. In terms of the educational mobile game, there are two possibilities for the near future. One is the chance to create another game, based on the present one, where the game environment is 3D, three-dimensional. The user could perform more actions than simply move and interact with the synchronization points. The other option is to add new modules or features to the present game, keeping it as simple as possible for the user.

REFERENCES

Android. 2013. Android 4.2. Jelly Bean. Google, Inc. [Online]. Available: http://www.android.com/. [Accessed: 03.04.2013].

Appcelerator. 2006. Appcelerator Titanium. [Online]. Available: http://www.appcelerator.com/. [Accessed: 25.09.2012].

Apple. 2002. Mac OS X Mountain Lion. Apple, Inc. [Online]. Available: http://www.apple.com/osx/. [Accessed: 03.04.2013].

Apple. 2010. iTunes Preview: Apple Store by Apple, Inc. Apple. [Online]. Available: https://itunes.apple.com/us/app/apple-store/. [Accessed: 13.03.2013].

Apple. 2013. iOS. Apple, Inc. [Online]. Available: http://www.apple.com/ios/. [Accessed: 03.04.2013].

AppMakr. 2012. AppMakr. [Online]. Available: http://www.appmakr.com/. [Accessed: 12.03.2013].

AppMobi. 2012. appMobi XDK. [Online]. Available: http://www.appmobi.com/. [Accessed: 12.03.2013].

Azalea. 1992. foneFrame. Azalea Software. [Online]. Available: http://www.qrdvark.com/foneFrame/. [Accessed: 12.03.2013].

Berners-Lee, T. 1990. Information Management: A Proposal. CERN. [Online]. Available: http://www.w3.org/History/1989/proposal.html. [Accessed: 12.03.2013].

Bethke, E. 2003. Game Development and Production. Wordware Publishing, Inc.

Blackberry. 2013. Blackberry. [Online]. Available: http://us.blackberry.com/. [Accessed: 10.04.2013].

Blow, J. 2004. Game Development: Harder Than You Think. Queue 10, 1, pp. 28–37.

Brocklehurst, S. 2010. Did Apple Make A Mistake Choosing Objective-C For iPhone SDK? Psynixis. [Online]. Available: http://psynixis.com/blog/2008/04/25/did-apple-make-a-mistake-choosing-objective-c-for-iphone-sdk/. [Accessed: 11.03.2013].

Button, K.J. and Hensher, D.A. 2001. Handbook of Transport Systems and Traffic Control. Emerald Group Publishing, pp. 495–497.

Caughey, W. 2012. Hidden Tech Features in Windows 8. Emerging Experiences. [Online]. Available: http://emergingexperiences.com/2012/11/hiddentechwin8. [Accessed: 14.01.2013].

Chan, C. 2012. Google Maps vs. Apple Maps: A Side-by-Side Comparison. Gizmodo. [Online]. Available: http://gizmodo.com/5918176/google-maps-vs-apple-maps-a-side-by-side-comparison. [Accessed: 12.03.2013].

Corasaniti, N. 2010. How Do People Use Their Smartphones? Blog Bits. [Online]. New York Times. Available: http://bits.blogs.nytimes.com/2010/09/14/report-looks-at-trends-with-mobile-apps/. [Accessed: 28.01.2013].

Creech, J. 2011. Android vs. iOS; A Usability Battle. Spyrestudios. [Online]. Available: http://spyrestudios.com/android-vs-ios-a-usability-battle/. [Accessed: 11.01.2013].

Dallera, A. 2011. Why you should stay away from Appcelerator's Titanium. [Online]. Available: http://usingimho.wordpress.com/2011/06/14/why-you-should-stay-away-from-appcelerator-titanium/. [Accessed: 17.01.2013].

Dune, R.G, Asher, K.N. and Rivara, F.P. 1992. Behavior and Parental Expectations of Child Pedestrians. Pediatrics 89, pp. 486–490.

DuPont, B. 2012. Tips for Writing Multiplatform Mobile Applications. Information Week. Education. [Online]. Available: http://www.informationweek.com. [Accessed: 01.02.2013].

Eddy, N. 2011. Mobile Internet Usage to Top Wireline Surfing by 2015: IDC Report. eWeek. [Online]. Available: http://www.eweek.com/c/a/Mobile-and-Wireless/Mobile-Internet-Usage-to-Top-Wireline-Surfing-by-2015-IDC-Report-617848/. [Accessed: 20.03.2013].

Elgin, B. 2005. Google buys Android for its mobile arsenal. Bloomberg Businessweek. [Online]. Available: http://www.webcitation.org/5wk7sIvVb. [Accessed: 11.03.2013].

Entertainment Software Association. 2011. Essential facts about the Computer and Video game Industry. E.S.A. 10, pp. 15–20.

Exadel. 1999. Tiggzi. [Online]. Available: http://tiggzi.com/home. [Accessed: 12.03.2013].

Flanagan, D. 2006. JavaScript: The Definitive Guide. O'Reilly & Associates. 5th edition, pp. 85–90.

Flite. 2009. Widgetbox. [Online]. Available: http://www.widgetbox.com/mobile/. [Accessed: 12.03.2013].

Forsythe, M.J. and Berger, W.G. 1973. Urban Pedestrian Accident Countermeasures Experimental Evaluation 1, Appendix C, Biotechnology, Inc. Falls Church, VA; Washington DC, US Department of Transportation.

Gay, J. 2001. The History of Flash. Adobe Systems, Inc. [Online]. Available: http://www.adobe.com/macromedia/. [Accessed: 03.04.2013].

Google. 2012. About Google Play – Google Play Help. Google Support. [Online]. Available: http://support.google.com/googleplay/. [Accessed: 13.03.2013].

Hamed, M.M. 2000. Analysis of Pedestrians' Behavior at Pedestrian Crossings. Safety Science 38, pp. 63–82.

Hay, D. 2012. 10 Solutions for Creating Cross-Platform Mobile Apps. Six Revisions. [Online]. Available: http://sixrevisions.com/mobile/cross-platform-mobile-apps/. [Accessed: 28.02.2013].

Hochbaum, Z. 2000. Safety strategies. Parents Magazine, pp. 14–33.

IBuildApp. 2010. iBuildApp. [Online]. Available: http://ibuildapp.com/. [Accessed: 12.03.2013].

Janssen, C. 2011. In-Game Purchases. Definition – What does in-game purchases mean? Technopedia. [Online]. Available: http://www.techopedia.com/definition/27615/in-game-purchases/. [Accessed: 28.01.2013].

JQuery. 2009. jQuery Mobile. [Online]. Available: http://jquerymobile.com/. [Accessed: 12.03.2013].

Kann, P. (2010). Architecture and characteristics of Android OS. Androideur. [Online]. Available: http://www.androideur.com/architecture-et-caracteristiques-du-os-android/. [Accessed: 11.03.2013].

Konami Corporation. 1981. Frogger. [Online]. Available: http://www.happyhopper.org/. [Accessed: 25.09.2012].

Lindeijer, T. 2008. Tiled. [Online]. Available: http://www.mapeditor.org/. [Accessed: 25.09.2012].

Linux. 2009. What is Linux: An Overview of the Linux Operating System. Linux Foundation. Linux.com. [Online]. Available: https://www.linux.com/learn/new-user-guides/. [Accessed: 03.04.2013].

Lucas, B. 2005. Which side of the road do they drive on? [Online]. Available: http://brianlucas.ca/roadside#roadnetwork/. [Accessed: 12.03.2013].

Lynn, L. 2012. Comparing Apple iOS 6 with Android OS 4.0, Windows Phone 7.5. CNET. [Online]. Available: http://reviews.cnet.com/8301-19512_7-57450741-233/comparing-apple-ios-6-with-android-4.0-windows-phone-7.5/. [Accessed: 14.01.2013].

Masterton, K. 2011. What makes ARM-based chips relatively power efficient? Quora. [Online]. Available: http://www.quora.com/What-makes-ARM-based-chips-relatively-power-efficient/. [Accessed: 03.04.2013].

Microsoft. 2013. Windows Phone. Microsoft. [Online]. Available: http://www.windowsphone.com/en-us. [Accessed: 10.04.2013].

Nourie, D. 2005. Getting Started with an Integrated Development Environment. Sun Microsystems. [Online]. Available: http://www.oracle.com/technetwork/java/. [Accessed: 17.01.2013].

OpenGL. 2013. OpenGL Overview. OpenGL. [Online]. Available: http://www.opengl.org/about/. [Accessed: 03.04.2013].

Oracle. 2013. Java. Oracle Technology Network. [Online]. Available: http://www.oracle.com/technetwork/java/index.html. [Accessed: 03.04.2013].

PhoneGap. 2004. PhoneGap. [Online]. Available: http://phonegap.com/. [Accessed: 12.03.2013].

Pivec, P. 2009. Game-based Learning or Game-based Teaching? Becta, pp. 19–21.

Reenskaug, T. and Coplien, J. 2009. The DCI Architecture: A New Vision of Object-Oriented Programming. Artima Developer. [Online]. Available: http://www.artima.com/articles/dci_vision.html. [Accessed: 10.04.2013].

Ritchie, D.M. and Thompson, K. 1978. The UNIX Time-Sharing System. American Telephone and Telegraph Company. The Bell Systems Technical Journal 57, 6, pp. 1927–1929.

Rothengatter, J.A. 1981. The Influence of Instructional Variables on the Effectiveness of Traffic Education. Accident Analysis and Prevention 13, pp. 241–253.

Schofer, J.L, Christoffel, K.K., Donovan, M., Lavigne, J.V., Tanz, R.R., and Wills, K. 1995. Child Pedestrian Injury Taxonomy based on Visibility and Action. Accident Analysis and Prevention 27, pp. 317–333.

Schuessler, J. 2010. The Godfather of the E-Reader. Sunday Book Review. The New York Times. [Online]. Available: http://www.nytimes.com/2010/04/11/books/review/Schuesslert.html?pagewanted=all&_r=0/. [Accessed: 10.04.2013].

Sencha. 2008. Sencha Touch 2. [Online]. Available: http://www.sencha.com/products/. [Accessed: 12.03.2013].

Selvam, R. and Karthikeyani, V. 2011. Mobile Software Testing – Automated Test Case Design Strategies. International Journal on Computer Science and Engineering 3, 4, pp. 1450–1461.

Silicon News. 2012. An Overview of the iOS Architecture. Silicon News. [Online]. Available: http://silicon-news.com/news/2012/06/15/ios-hardware-architecture. [Accessed: 11.03.2013].

SQLite. 2010. About SQLite. SQLite. [Online]. Available: http://www.sqlite.org/. [Accessed: 03.04.2013].

Streetwise. 2006. [Online]. Available: http://www.3m.co.uk/intl/uk/3mstreetwise/. [Accessed: 25.09.2012].

Tales of the Road. 2009. [Online]. Available: http://talesoftheroad.direct.gov.uk/. [Accessed: 25.09.2012].

Thomson, J.A., Tolmie, A.K., Foot, H.C., Whelan, K.M.; Sarvary, P., and Morrison, S. 2005. Influence of Virtual Reality Training on the Roadside Crossing Judgments of Child Pedestrians. Journal of Experimental Psychology 11, pp. 175–186.

Titanium. 2006. Titanium Mobile Overview. Appcelerator Titanium. [Online]. Available: http://docs.appcelerator.com/titanium/3.0/. [Accessed: 12.03.2013].

Van Dillen, K., Lieberman, P., and Sonsev, V. 2012. Mobile Industry Overview and Trends. Women in Wireless. Golden Seeds Forum. Part 1, pp. 53–56.

Viswanathan, P. 2012. Android OS vs. Apple iOS – Which is better for Developers? Pros and Cons of the Android OS and the Apple iOS. About.com. [Online]. Available: http://mobiledevices.about.com/od/kindattentiondevelopers/tp/Android-Os-Vs-Apple-Ios-Which-Is-Better-For-Developers.htm. [Accessed: 14.01.2013].

Warren, C. 2012. The Pros and Cons of Cross-Platform App Design. Mashable. [Online]. Available: http://mashable.com/2012/02/16/cross-platform-app-design-pros-cons/. [Accessed: 01.02.2013].

W3C. 2013. W3C. [Online]. Available: http://www.w3.org/. [Accessed: 10.04.2013].

APPENDIX 1: SYSTEM REQUIREMENT DOCUMENT

This section provides description of the System Requirements Document. It begins with the User Requirements specification document, a specification of the user requirements that the system should fulfill. Then, there is the Use Case document and the Non-Functional Requirements specification.

User Requirement Specification

In order to supply a clear, concise and easy-to-track UR specification, each requirement will be presented as in Table 1.

Table 1. *User Requirements Template*

Identifier	Description	User Priority	Technical Priority	Stability
UR-n	A detailed description of the UR.	1 to 5	1 to 5	Current state

The fields in the template above will be filled with the following information.

- **Identifier.** A descriptive identifier for the requirement. The format used is UR-n, where n is a two-digit number.

- **Description.** A detailed description about the user requirement.

- **User Priority.** An integer that represents the user priority in a specific requirement. It is a numeric value 1–5, where 1 represents low priority and 5 means high priority.

- **Technical Priority.** An integer that represents the technical priority in a specific requirement. It is a numeric value 1–5, where 1 represents low priority and 5 means high priority.

- **Stability.** The current state of the requirement. Its value could be *Stable*, which means the requirement is not likely to change, or *Unstable*, which means that the requirement may be dependent on feedback on other user requirements or system requirements, and it could change in the future.

Table 2. *User Functional Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
UFR-01	The system shall add new users.	5	5	Stable
UFR-02	The system shall remove any user.	5	5	Stable
UFR-03	The system shall modify any information about a user.	5	5	Stable
UFR-04	The system shall provide the information about a user.	4	3	Stable
UFR-05	The system shall manage the synchronization points of a user.	5	5	Stable
UFR-06	The system shall be able to store all the information in the storage facility of the device.	3	5	Stable
UFR-07	The system should handle a world where the character moves around.	5	3	Stable
UFR-08	The system shall provide all the directions to move the character anytime.	5	5	Stable
UFR-09	The system shall manage the road hazards, such as cars or traffic lights.	5	2	Stable
UFR-10	The system should manage cars to follow road rules.	5	2	Stable

Use Case Document

Since, this is a small system, it is only necessary to group all the use cases in one package. The uses cases are based on the requirement defined in the previous section. The interaction in the system is between the user and the storage facility of the device, called *Database*, but this does not need to be a full-fledged DBMS.

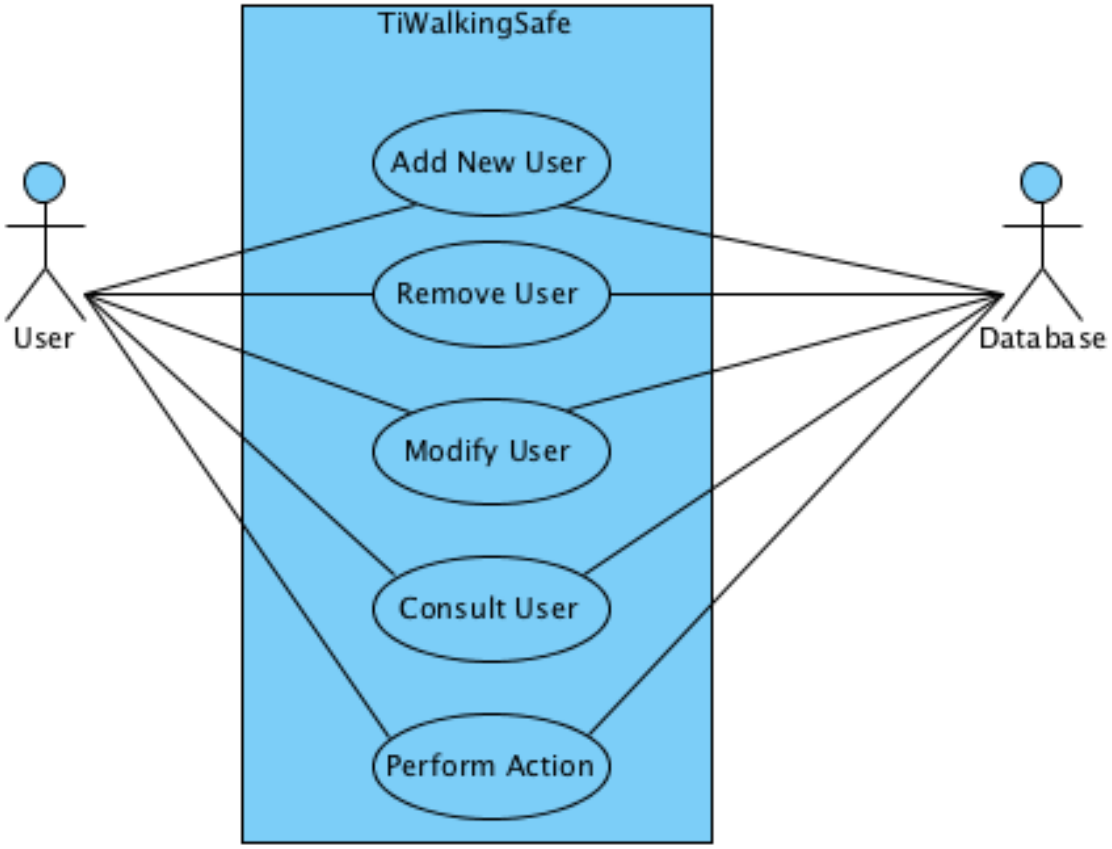

Figure 1. *Use Case Diagram*

Tables 3 to 7 depict the complete use case specification, exploring all the possibilities handled by each use case.

***Table 3**. Use Case – Add new user*

Version	Description	Modified by	Date
2.0	Third version	A. Garcia–Moya	04/04/2013

Name	Add New User
Brief Description:	The system should bring the possibility of adding a new user to the system.
Business Trigger:	-
Preconditions:	A new user wants to start using the system.

Basic Flow: The system processes all the information about the new user and stores it in the system database.		
Line	**System Actor Action**	**System Response**
1	The user adds the information about him.	The system checks that all the text fields have information and shows a confirmation message.
2	The user accepts to store the new file.	The system stores the information file in the system database.
Post Condition:	The system has a new file in its database.	

Alternate Flow (AF):		
If at line 1		
Line	**System Actor Action**	**System Response**
1	The user leaves any text field without information.	The system finds the incomplete fields and shows the error message.
2	The user introduces information in the empty fields and accepts to store the file.	The system stores the file.
The use case terminates when the system stores the file. / The use case does not restart in the basic flow as it ends on the Alternate Flow.		
Post Condition:	The system has a new file in its database	

Table 4. *Use Case – Remove user*

Version	Description	Modified by	Date
2.0	Third version	A. Garcia–Moya	11/04/2013

Name	Remove User
Brief Description:	The system should bring the possibility of removing user's file from the system.
Business Trigger:	-
Preconditions:	A user wants to finish using the system.

Basic Flow: The system looks in the database for the specific user's file.		
Line	**System Actor Action**	**System Response**
1	The user adds the user's nickame and password.	The system checks the information and shows the file.
2	The user accepts to remove it.	The system removes the file from the database.
Post Condition:	The system has no information about this user.	

Alternate Flow (AF):		
If at line 1		
Line	**System Actor Action**	**System Response**
1	The user adds an incorrect name or password.	The system shall not find the specific file and communicating to the user.
2	The user corrects the name or the password introduced.	The system removes the file from the database.
The use case terminates when the system removes the file. / The use case does not restart in the basic flow as it ends on the Alternate Flow.		
Post Condition:	The system has no information about this user.	

Table 5. *Use Case – Modify user*

Version	**Description**	**Modified by**	**Date**
1.5	Second version	A. Garcia–Moya	11/04/2013

Name	Modify User
Brief Description:	The system should bring the possibility of modifying the information about a user.
Business Trigger:	-
Preconditions:	A user wants to modify his information.

Basic Flow: The system checks the database looking for the user, and then, processes the new information to store it in the database.		
Line	**System Actor Action**	**System Response**
1	The user adds the name and password of the user.	The system looks for the user, checking the password, and shows the file.
2	The user modifies the information.	The system checks all the input information and shows a confirmation message.
3	The user accepts the modification to the file.	The system rewrites the information file.
Post Condition:	The system has a modified file in its database.	

Alternate Flow (AF):		
If at line 1		
Line	**System Actor Action**	**System Response**
1	The user adds an incorrect name or password.	The system shall not find the file and communicates this to the user.
2	The user corrects the name or password introduced.	The system shows the file.
The use case terminates when the system updates the file. / The use case does not restart in the basic flow as it ends on the Alternate Flow.		
Post Condition:	The system has a modified file in its database	

<table>
<tr><td colspan="3">Alternate Flow (AF):</td></tr>
<tr><td colspan="3">If at line 2</td></tr>
<tr><td>Line</td><td>System Actor Action</td><td>System Response</td></tr>
<tr><td>1</td><td>The user writes incomplete information.</td><td>The system finds the incomplete fields and shows the error message.</td></tr>
<tr><td>2</td><td>The user completes the empty fields and accepts to store the file.</td><td>The system stores the updated file.</td></tr>
<tr><td colspan="3">The use case terminates when the system updates the file. / The use case does not restart in the basic flow as it ends on the Alternate Flow.</td></tr>
<tr><td colspan="2">Post Condition:</td><td>The system has a modified file in its database</td></tr>
</table>

Table 6. *Use Case – Consult User*

Version	**Description**	**Modified by**	**Date**
1.5	Second version	A. Garcia–Moya	11/04/2013

Name	Consult User
Brief Description:	The system should bring the possibility of consulting the information about a user.
Business Trigger:	-
Preconditions:	A user wants to consult the information in the file.

Basic Flow: The system checks the database and presents the file.		
Line	**System Actor Action**	**System Response**
1	The user adds name of the user.	The system checks the name and shows the file.
2	The user consults the file.	-
Post Condition:	The system keeps the user's file unchanced.	

Alternate Flow (AF):		
If at line 1		
Line	**System Actor Action**	**System Response**
1	The user adds an incorrect name.	The system shall not find any file with the specific name.
2	The user checks and corrects the name.	The system shows the specific file.
3	The user consults the file	-
The use case terminates when the system shows the file. / The use case does not restart in the basic flow as it ends on the Alternate Flow.		
Post Condition:	The system keeps the user's file unchanced.	

Table 7. *Use Case – Perform Action*

Version	Description	Modified by	Date
2.0	Third version	A. Garcia–Moya	11/04/2013

Name	Perform action
Brief Description:	The system should allow the user perform any movement on the main character.
Business Trigger:	-
Preconditions:	A user wants to move the character.

Basic Flow: The system processes the action selected and performs it.		
Line	**System Actor Action**	**System Response**
1	The user selects an action available.	The system checks the action and performs it.
2	The user observed the action realization.	The system stores the new state of the character in the game.
Post Condition:	The system is in a new state of the game.	

Alternate Flow (AF):		
If at line 1		
Line	**System Actor Action**	**System Response**
1	The user tries to perform a non-possible action.	The system checks the action and shows an error message.
2	The user chooses a correct action.	The system performs the new action.
The use case terminates when the system performs the action in the game. / The use case does not restart in the basic flow as it ends on the Alternate Flow.		
Post Condition:	The system is in a new state of the game.	

Non-Functional Requirement Specification

In order to supply a clear, concise and easy-to-track requirements specification, each requirement will be presented in the same table-like template as the user requirements. Tables 8 to 16 depict the Non-Functional Requirement Specification of the system.

Table 8. *Performance Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFPR-01	The system shall be able to process any valid user's action, in less than 15 seconds.	5	5	Stable

Table 9. *Security Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFSR-01	The system should protect the user's information.	5	4	Stable
NFSR-02	The user's profile shall only be accessible from the user's mobile device.	3	4	Stable

Table 10. *Safety Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFSAR-01	The system should be able to make a backup with all its data stored avoiding data losses in case it is necessary.	3	5	Stable
NFSAR-02	The system shall avoid revealing the user's password to others or allowing them to change it.	5	5	Stable

Table 11. *Usability Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFUR-01	The system shall have common and big icons to provide a friendly navigation control.	4	3	Stable
NFUR-02	The system shall allow the user consults his profile any time.	5	2	Stable
NFUR-03	The system shall need a low learning period of time allowing the user starts use it soon, e.g. less than one hour.	4	4	Stable
NFUR-04	The system's UI shall not change all over time, to keep be friendly to old users.	4	1	Stable

Table 12. *Portability Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFPOR-01	The system shall be able to run over Android IDE and iPhone IDE without changes or errors.	3	5	Stable

Table 13. *Reliability Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFRR-01	The system shall have a low response time in order to avoid user wasting time, e.g., less than 5 sec.	5	5	Stable
NFRR-02	The system shall always keep in order all the information stored in the database.	3	4	Stable

Table 14. *Maintainability Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFMR-01	The system's maintenance should keep the system's functionality in the initial response time: under 50 ms.	5	5	Stable

Table 15. *Technical Environment Constraints*

Identifier	Description	User Priority	Technical Priority	Stability
NFTR-01	The system's interaction with the user shall be done through the touch-screen of the device.	5	5	Stable

Table 16. *Policy Requirements*

Identifier	Description	User Priority	Technical Priority	Stability
NFPOLR-01	The data model of the system should follow the guidelines of relational databases.	5	5	Stable

APPENDIX 2: SYSTEM DESIGN DOCUMENT

This section provides description of the System Design Document. It begins with the games prototypes, the diagram of the system, the explanation of the architecture selected, then the component and deployment diagrams of the system. Finally, there is detailed system decomposition and the traceability matrix.

Game Prototypes

This section provides a set of examples of the design of the game could look like. Those designs are the first draft, so they will suffer some changes till the final implemented version of the application. Figures 2 and 3 illustrate the first version of the design. Here the idea was to create a general view of the world, and at each crosswalk, the game would show a view of the crosswalk, and the user would have to decide when to cross (Figure 2). Figures 4 to 7 are representations of the second version of the design, where there are already interfaces for the application and also the general view of the world has been remodeled. Figure 8 depicts the design of the game. It is going to be a general world, but the user would see a portion of that world, that fits in the dimensions of the device. As the user moves the character outside that portion, the game provides the view that corresponds to the new section.

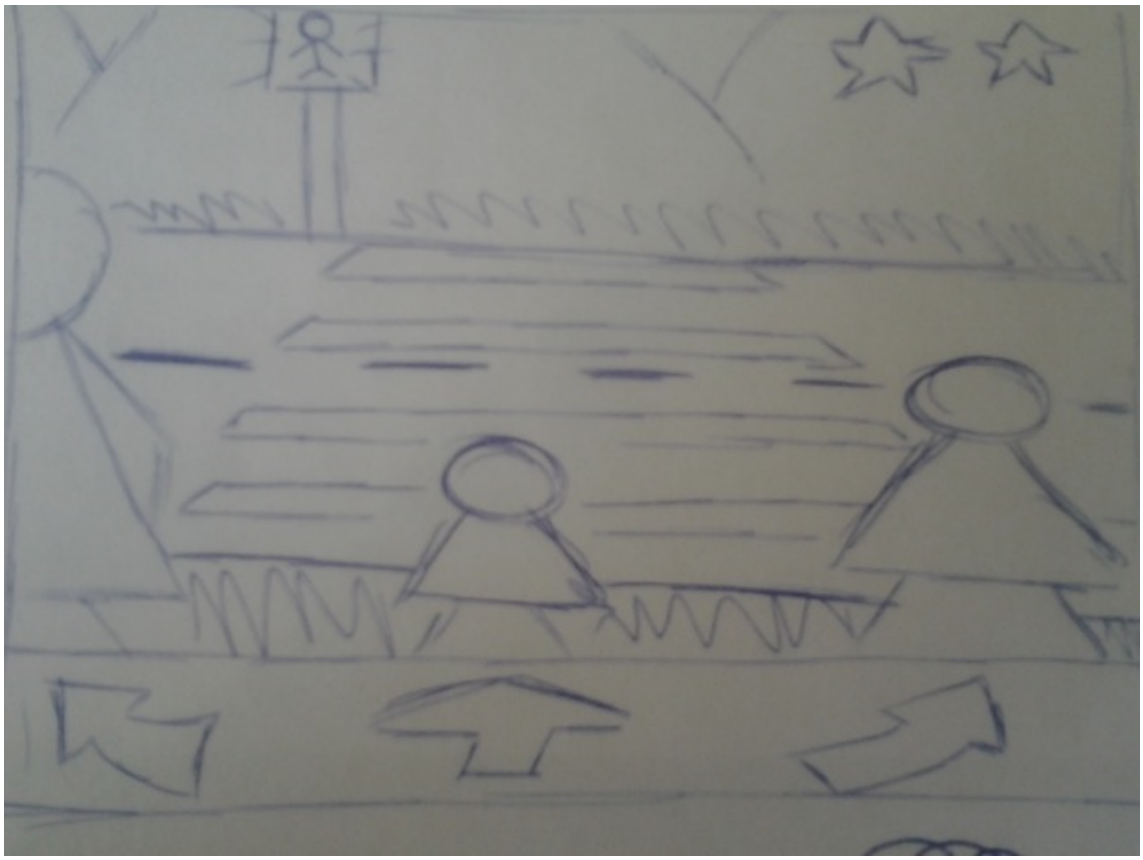

Figure 2. *First prototype of the game – Level view*

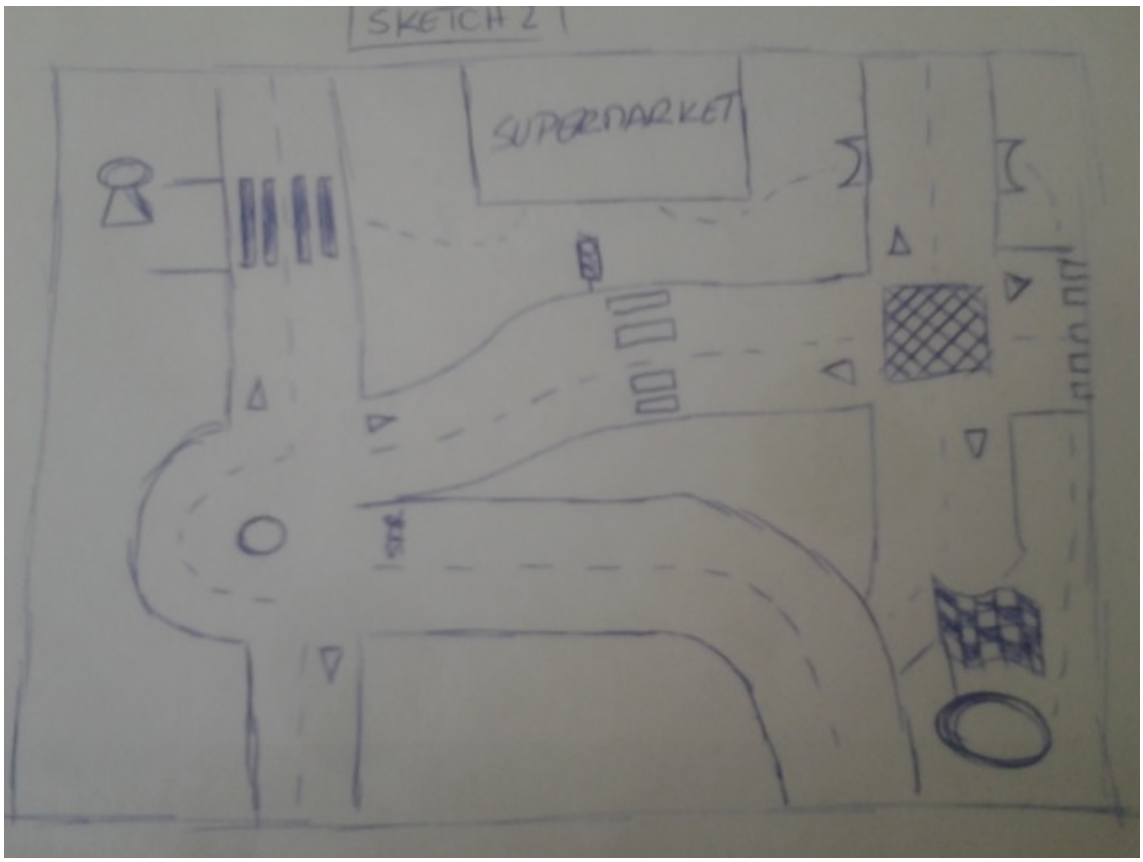

Figure 3*. First prototype of the game – World view*

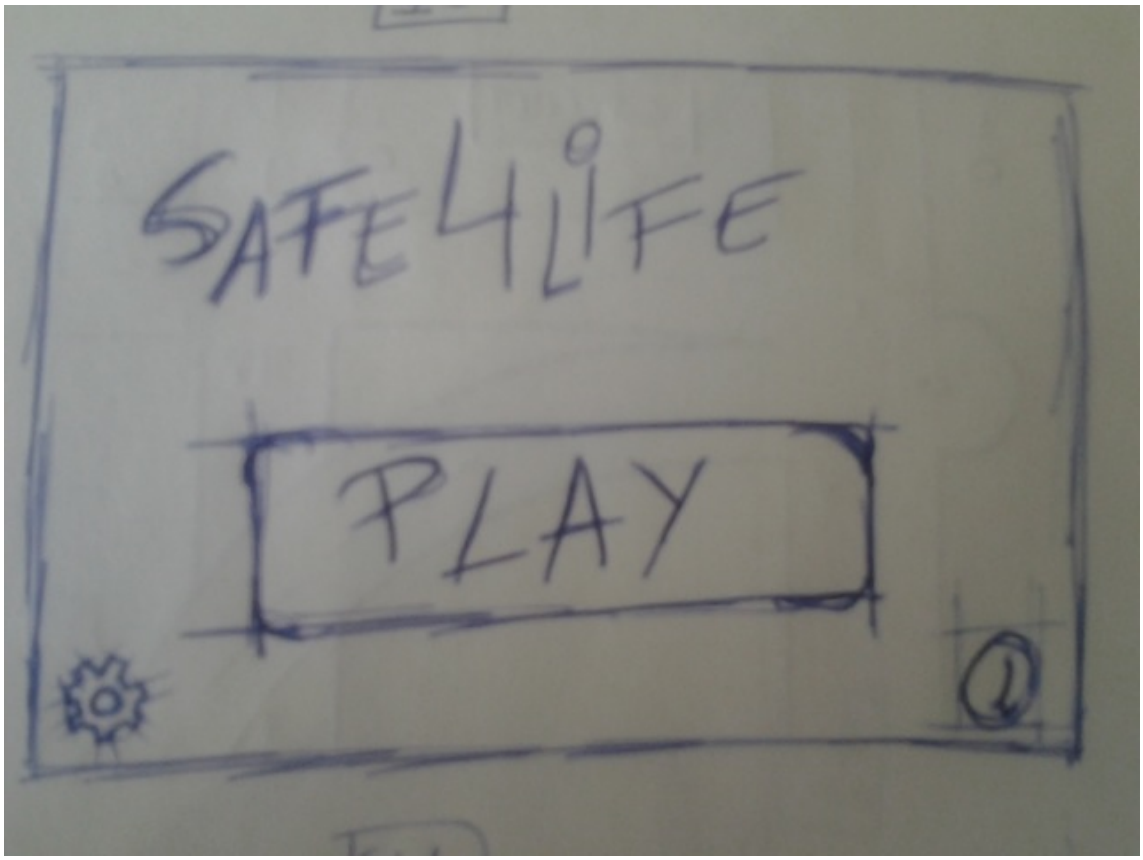

Figure 4. *Second prototype of the game – First interface*

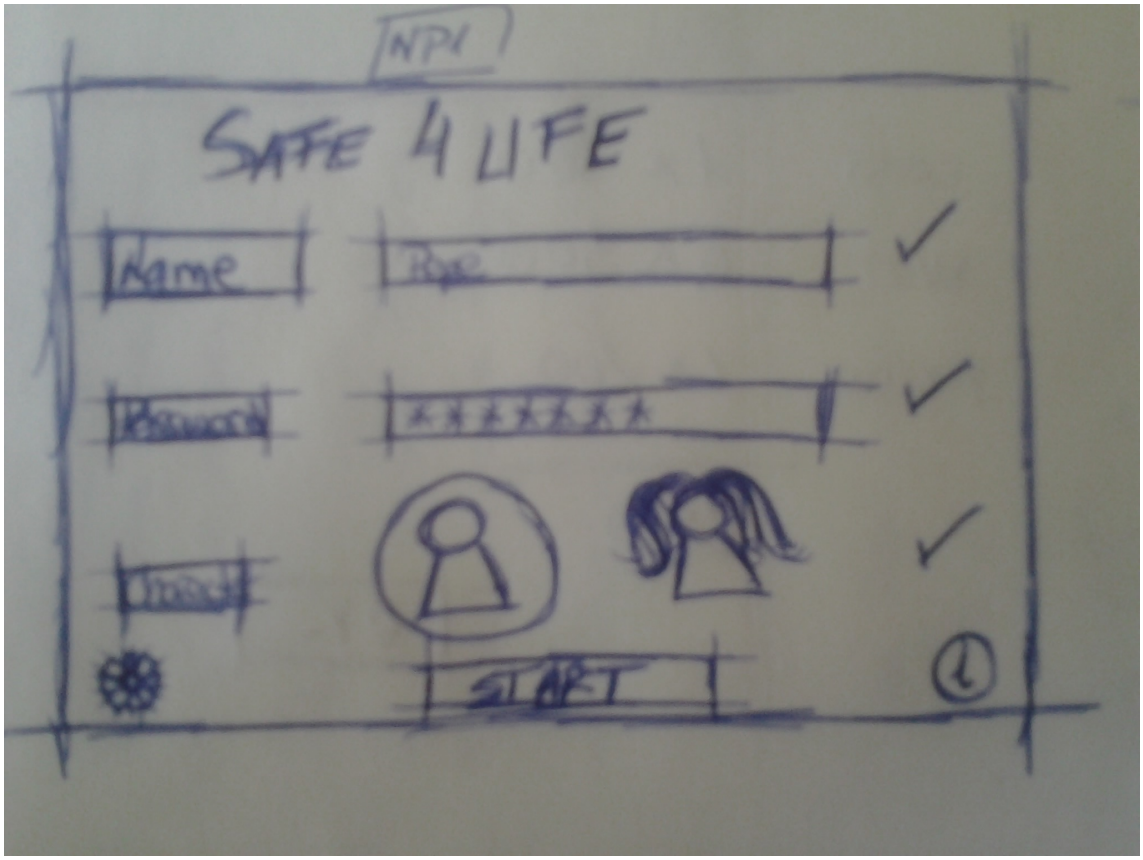

Figure 5. *Second prototype of the game – Manage user interface*

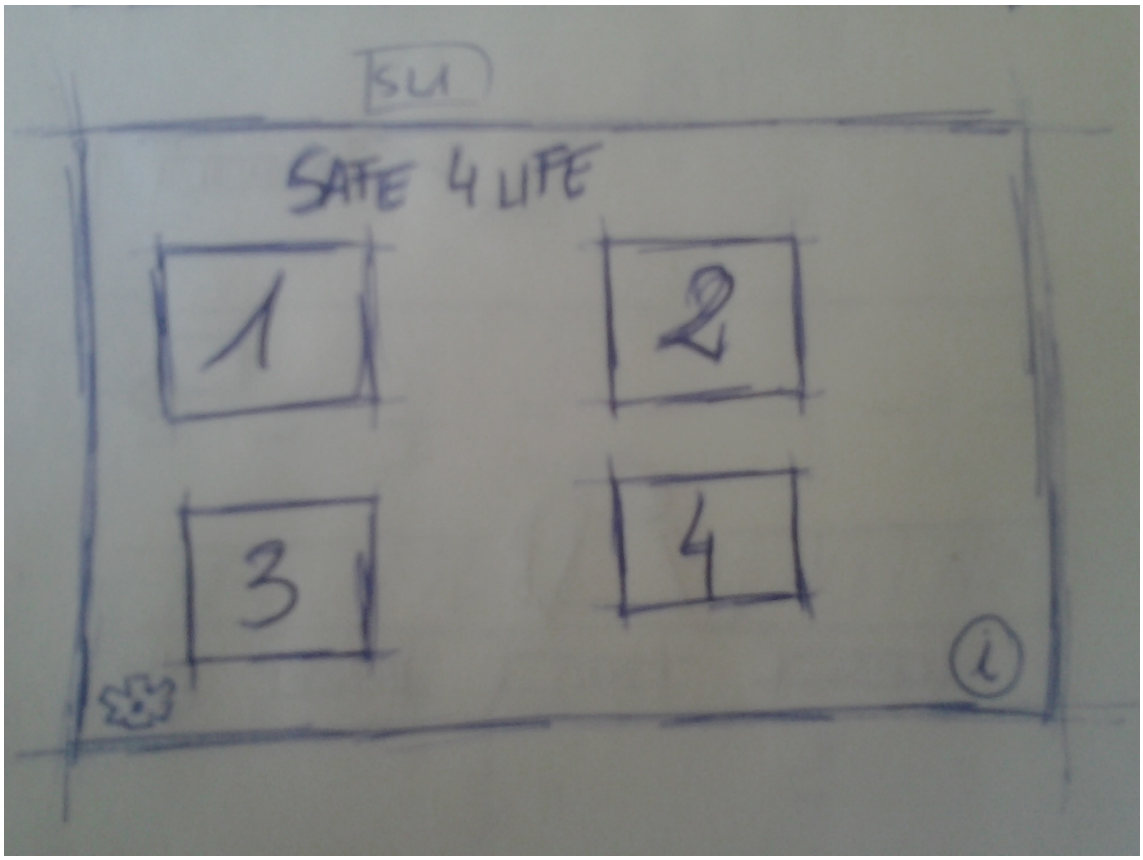

***Figure 6**. Second prototype of the game –Level interface*

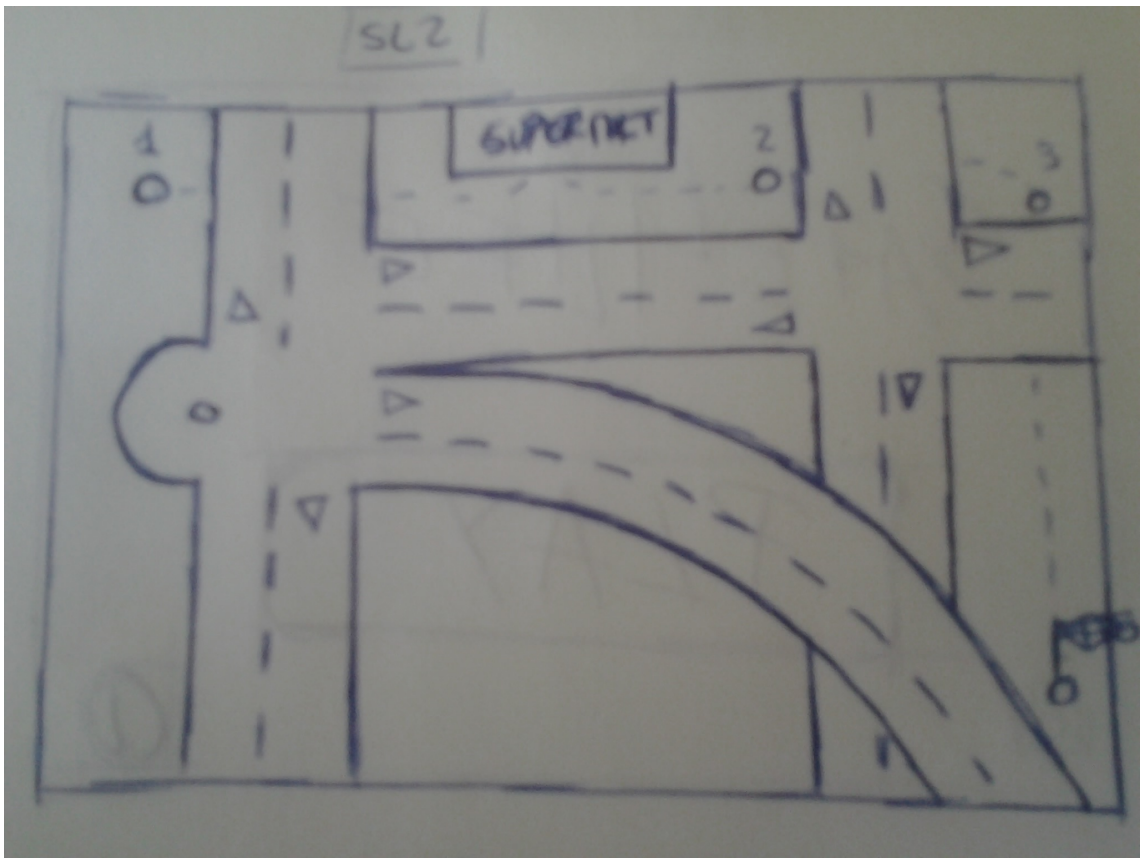

***Figure 7**. Second prototype of the game – World view*

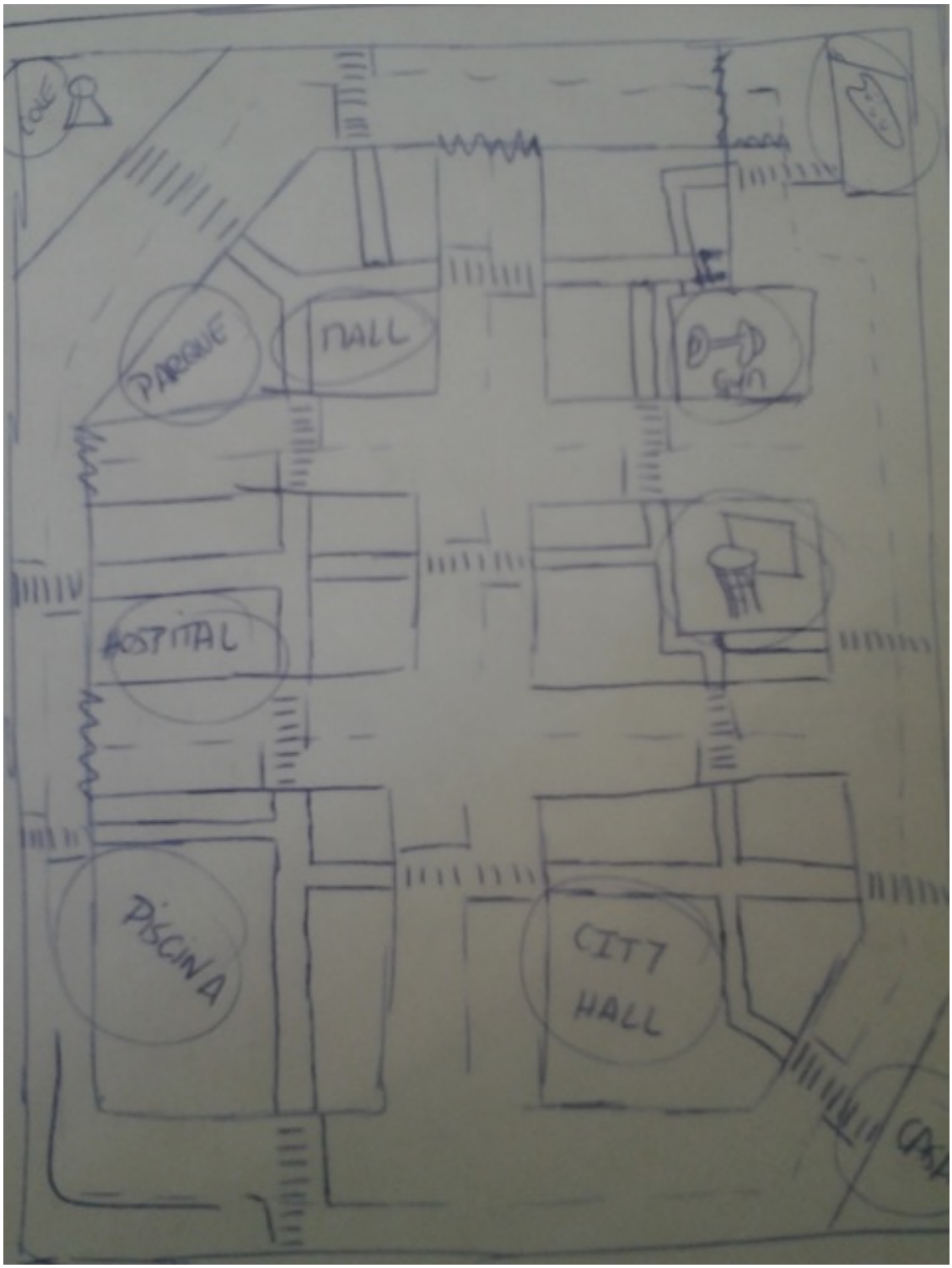

Figure 8. *Third prototype of the game – World view*

Class Diagram

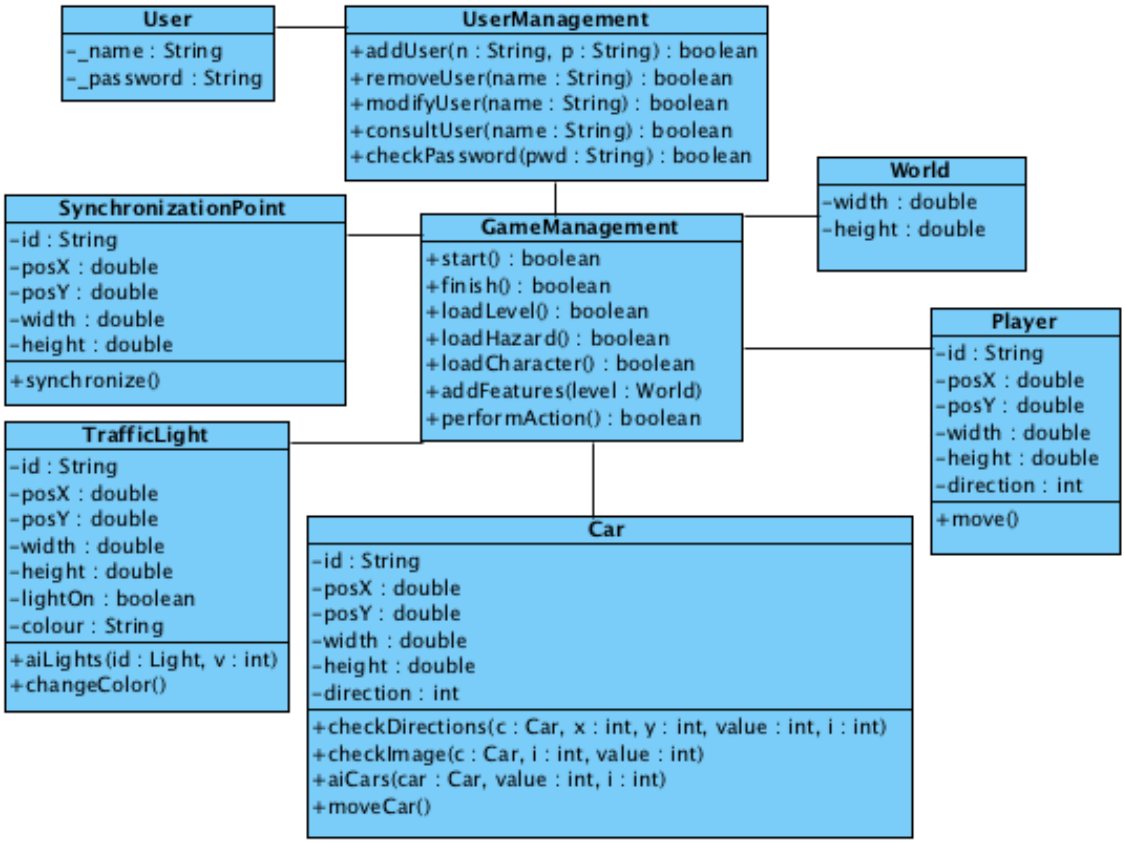

Figure 9. *Class Diagram*

Figure 9 illustrates the class diagram of the system. As one of the first steps in the system design, this can be changed or completely redesigned in case the development process cannot fulfill the statements of the present diagram. In case the programming paradigm does not support classes and methods, the diagram will be implemented in some other way, but with the same result.

The system is based on the entity *GameManagement*. It handles the rest of the entities, in order to load and process the world of the game and any request the user makes. This entity has relations with all the entities that manage the user, the road hazards, such as cars and traffic lights, the player, and the synchronization points. Entity *Player* handles the main character of the game, the size of the image and its position in the game, and is able to perform actions with the character, such as walking or interacting with synchronization points. Entity *World* represents the world created outside the system

and loaded for the game, and it contains the size of it. Entity *Car* handles each car of the game, the size of the image, its position in the world, and the direction that the car is moving. It is also in charge of moving the car on the roads and respects the traffic lights. *TrafficLight* represents each of the traffic lights of the game. It handles the size of the image, its position, if it is working or not, and the color of the light. It also manages the behavior of the traffic light. *SynchronizationPoint* represents the synchronization point of each level of the game. It handles the size of the image and its position. It is in charge of checking if the player is in the correct location when the user clicks the point on the screen. Entity *UserManagement* is in charge of handling information about the user. It performs functions such as adding, removing, updating or consulting information of the entity *User*.

System Architecture

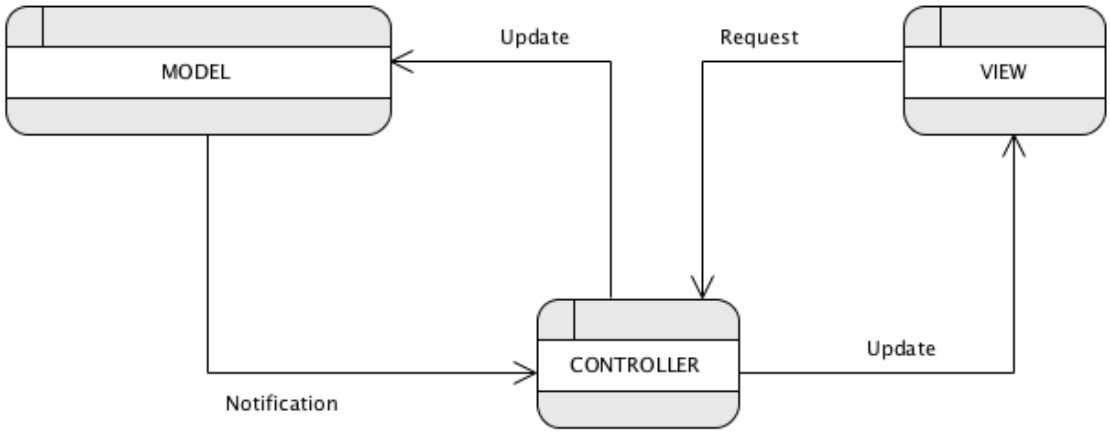

Figure 10. *Model-View-Controller Architecture (Adapted from Reenskaug et al. 2009)*

Figure 10 illustrates the Model-View-Controller architecture implemented in the present system. This architecture is divided in three components, with different functionalities.

- *Model* is the specific representation of the information that the system interacts with.

- *View* interacts with the user, receiving all the input requests that the system has to handle.

- *Controller* is the component between *Model* and *View*. It is in charge of performing actions according to the user's requests, updating the data stored in the system. Then, it transforms the data to be presented in the *View* interface.

Component Diagram

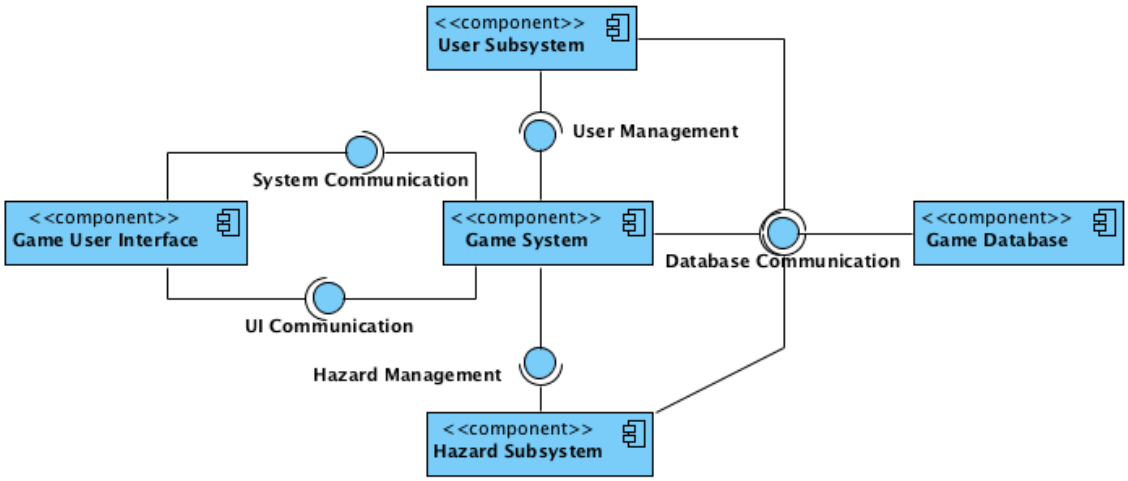

Figure 11. *Component Diagram*

Figure 11 depicts the component diagram of the system. This diagram groups the class entities into components following same functionalities. The main component is *Game System,* which handles the rest of the components of the system. It contains the following class entities: *GameManagement, Player, World* and *SynchronizationPoint.* The component *Game System* is connected to *Game User Interface* through the *System Communication* interface. *Game User Interface* contains all the statements of the user interface of the system. It is connected with *Game System* through *UI Communication* interface. *User Subsystems* component contains the class entities *User* and its manager. It is connected with *Game System* through the interface *User Management. Hazard Subsystems* component contains the class entities *Car* and *TrafficLight.* It is connected with *Game System* through *Hazard Management* interface. *Game Database* component is connected with *Game, User,* and *Hazard Subsystem* components, through the *Database Communication* interface.

Deployment Diagram

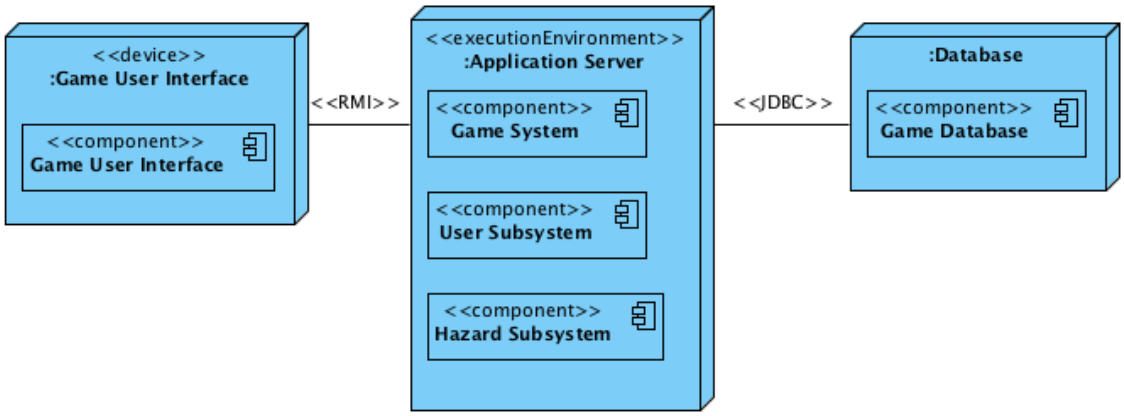

Figure 12. *Deployment Diagram*

Figure 12 depicts the deployment diagram of the system. The deployment diagram has three nodes. The first node is the *Game User Interface*. It provides communication between the system and the user. The second node *Application server*, contains the entire system engine, thus all the subsystem components are in this node. Finally, the third node is called *Database.* It represents the database of the device that supports the *Game Database* component.

Detailed System Decomposition

In this section, the detailed system decomposition is provided. The focus will be on the software component of the system, which may be implemented grouping the classes of the system class diagram. In order to group in components, there should consider the specific related functionality that each class provides.

COMPONENT CO-01: Game User Interface

This component should provide to the mobile device graphical interfaces where users can interact with the system. This component is part of the interface layer.

Purpose

This component is based on the following functional requirements: UFR-01, UFR-02, UFR-03, UFR-04, UFR-05, and UFR-08.

Function

This component should provide human users interaction through the mobile device.

Subordinates

This component is not decomposed into any other component.

Dependencies

This component does not depend on any other component.

Interfaces

This component provides the User Interface Communication through the mobile device.

Resources

This component should use resources from the mobile device and the programming language Javascript.

References

User Requirements specification

System Design specification

Processing

This component processes its inputs following two ways.

- User Inputs.
    - The UI component may receive orders from human users.
    - The UI component shall send them to the Game Subsystem.
- System Inputs.
    - The UI component may receive orders from Game Subsystem.
    - The UI component shall show them through the mobile device.

Data

This component deals with all the user's information and orders that are introduced through the mobile device and the feedback information from the system to the user.

COMPONENT CO-02: Game System

This component should provide the main management of all the aspects related to the game and the system. It is the component in charge of handles the rest of the smaller components. The classes comprising it: Player, World, SynchronizationPoint and GameManagement.

Purpose

This component is based on the following functional requirements: UFR-05, UFR-07, UFR-08, UFR-09, and UFR-10.

Function

This component should provide the system the following functionalities.

- Manage users.
- Manage road hazards.
- Manage the world.
- Manage main character.

Subordinates

This component is not decomposed into any other component.

Dependencies

This component depends on the Game user interface through System communication interface.

Interfaces

This component provides the User Interface Communication with the user interface.

Resources

This component should use resources from the mobile device and the programming language Javascript.

References

User Requirements specification

System Design specification

Processing

This component processes its inputs as follows.

- Receive request from Game user interface.
    - The Game subsystem component receives requests from Game user interface component.
    - The Game subsystem component communicates them to the specific component to complete the command.

Data

This component deals with information obtained from the component in the interface layer. This information sent, typically, is about managing users and their commands, hazards, or levels of the game, and the users shall require the information obtained.

COMPONENT CO-03: User Subsystem

This component should provide the management of the users of the system and the internal interaction between the system and the users. The following classes comprise it: User and UserManagement.

Purpose

This component is based on the following functional requirements: UFR-01, UFR-02, UFR-03, and UFR-04.

Function

This component should provide the system the following functionality.

- Manage users.

Subordinates

This component is not decomposed into any other component.

Dependencies

This component does not depend on any other component.

Interfaces

This component provides the User Management communication with the Game Subsystem.

Resources

This component should use resources from the mobile device and the programming language Javascript.

References

User Requirements specification

System Design specification

Processing

This component processes its inputs as follows.

- Manage users.
    - The User subsystem component may receive information about users.
    - The User subsystem component would communicate with Game Subsystem component or Game Database component to complete the request.

Data

This component deals with information obtained from and typically about the user.

COMPONENT CO-04: Hazard Subsystem

This component should provide the management of the hazards of the game, such as cars, and traffic lights, each with its own features. It is composed of the following classes: Car and TrafficLight.

Purpose

This component is based on the following functional requirements: UFR-09 and UFR-10.

Function

This component should provide the system the following functionality.

- Manage road hazards.

Subordinates

This component is not decomposed into any other component.

Dependencies

This component does not depend on any other component.

Interfaces

This component provides the Hazard Management communication with the Game Subsystem.

Resources

This component should use resources from the mobile device and the programming language Javascript.

References

User Requirements specification

System Design specification

Processing

This component processes its inputs as follows.

- Manage hazards, such as cars, trucks, traffic lights or other pedestrians.
    - The Hazard subsystem component may receive information about hazards.
    - The Hazard subsystem component would communicate with Game Subsystem component or Game Database component to perform the request.

Data

This component deals with information about hazards behavior.

COMPONENT CO-05: Game Database

This component should provide the system the management of the storage unit of the device. This component is part of the data layer.

Purpose

This component is based on the following functional requirements: UFR-01, UFR-02, UFR-03, UFR-04, UFR-05, UFR-06, UFR-07, UFR-08, UFR-09, and UFR-10.

Function

This component should provide the system the following functionalities.

- Manage data in the system database.

Subordinates

This component is not decomposed into any other component.

Dependencies

This component does not depend on any other component.

Interfaces

This component provides the Database communication with Game, User, Award and Hazard Subsystem components.

Resources

This component should use resources from the mobile device storage unit.

References

User Requirements specification

System Design specification

Processing

This component processes its inputs as follows.

- General storage.
    - Game database component shall receive information that has to be stored, from components hosted on application unit layer.
    - Game database component should store each data received from the system, on the specific part of the storage unit reserved to it.

Data

This component deals with the relevant information about every aspect of the system.

System Design vs. Object Design Traceability Matrix

Table 17. *System Design vs. Object Design Traceability Matrix*

	CO-01	**CO-02**	**CO-03**	**CO-04**	**CO-05**
UFR-01	X		X		X
UFR-02	X		X		X
UFR-03	X		X		X
UFR-04	X		X		X
UFR-05	X	X			X
UFR-06					X
UFR-07		X			X
UFR-08	X	X			X
UFR-09		X		X	X
UFR-10		X		X	X

Figure 17 illustrates the traceability matrix of the functional requirements, and the components of the system. This matrix is a visual aid to understand which component or components perform the functionality of each requirement. As this is a small system, there are only five components that perform the ten functional requirements of the system.

APPENDIX 3: USER'S MANUAL

This chapter includes the user's manual about the mobile game called *TiWalkingSafe*. This manual details all the functionalities that the user can perform in the mobile application.

Home interface

Once installed in the device and started the system, it displays the home interface, as in Figure 13. Through this interface the user can access the user management view, pressing the button *New User,* or the game directly, pressing the button *Start*.

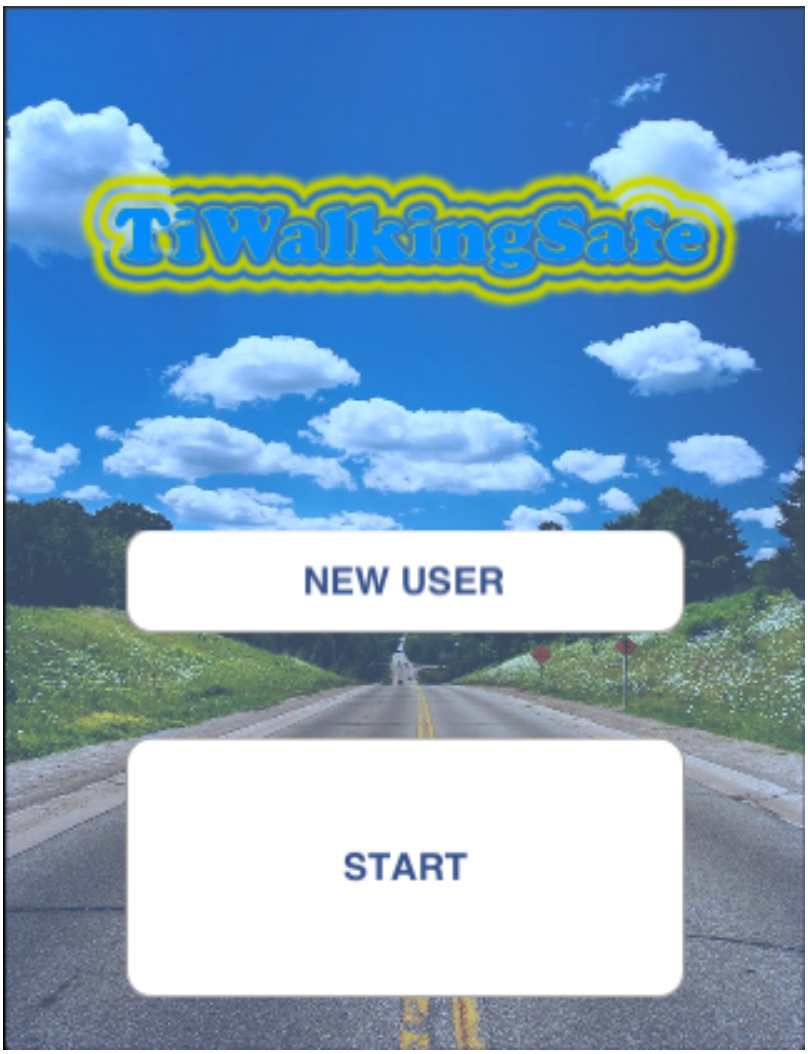

Figure 13. *Home interface*

New User interface

Figure 14 depicts the menu to create a new user. The user can write a nickname, such as *John Smith*, and a password like *JohnS1234*. The password characters hide when the user writes them down in the application. When it is complete, the user can press the *Accept* button and go straight to the introduction to the game.

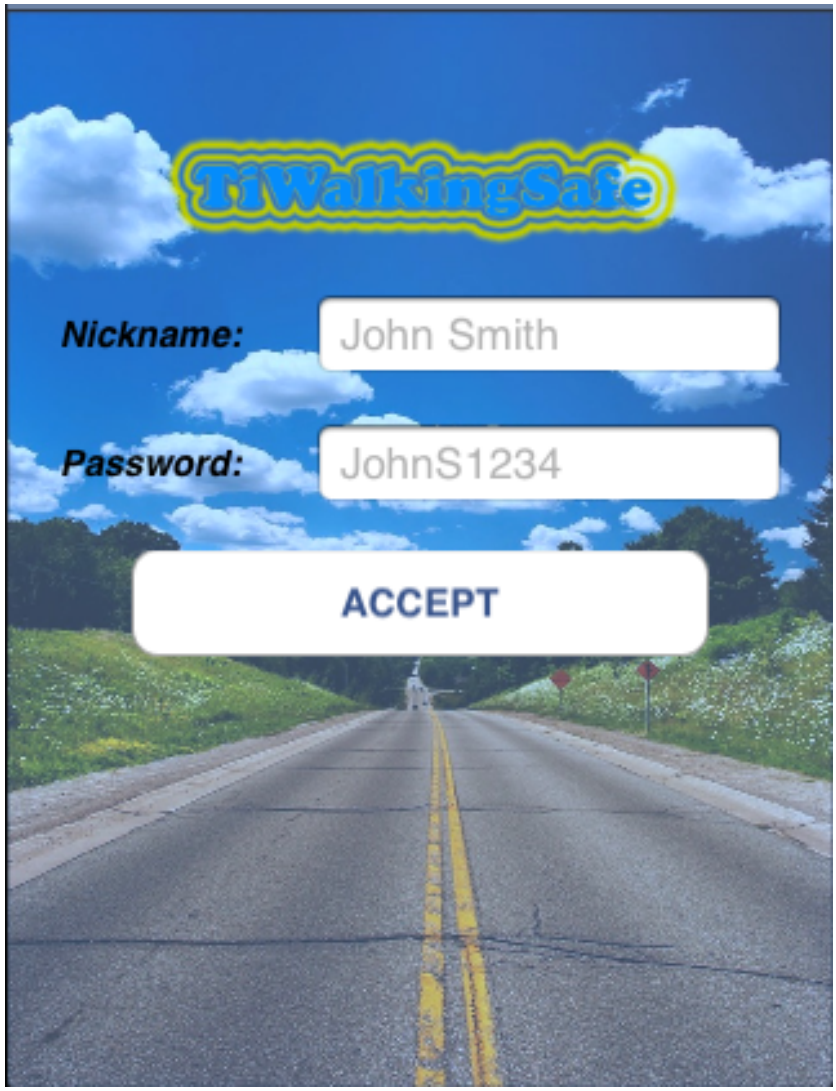

Figure 14. *New User interface*

Introduction interface

Figure 15 depicts the introduction to the game. In order to start playing the game, the user shall navigate to the game interface.

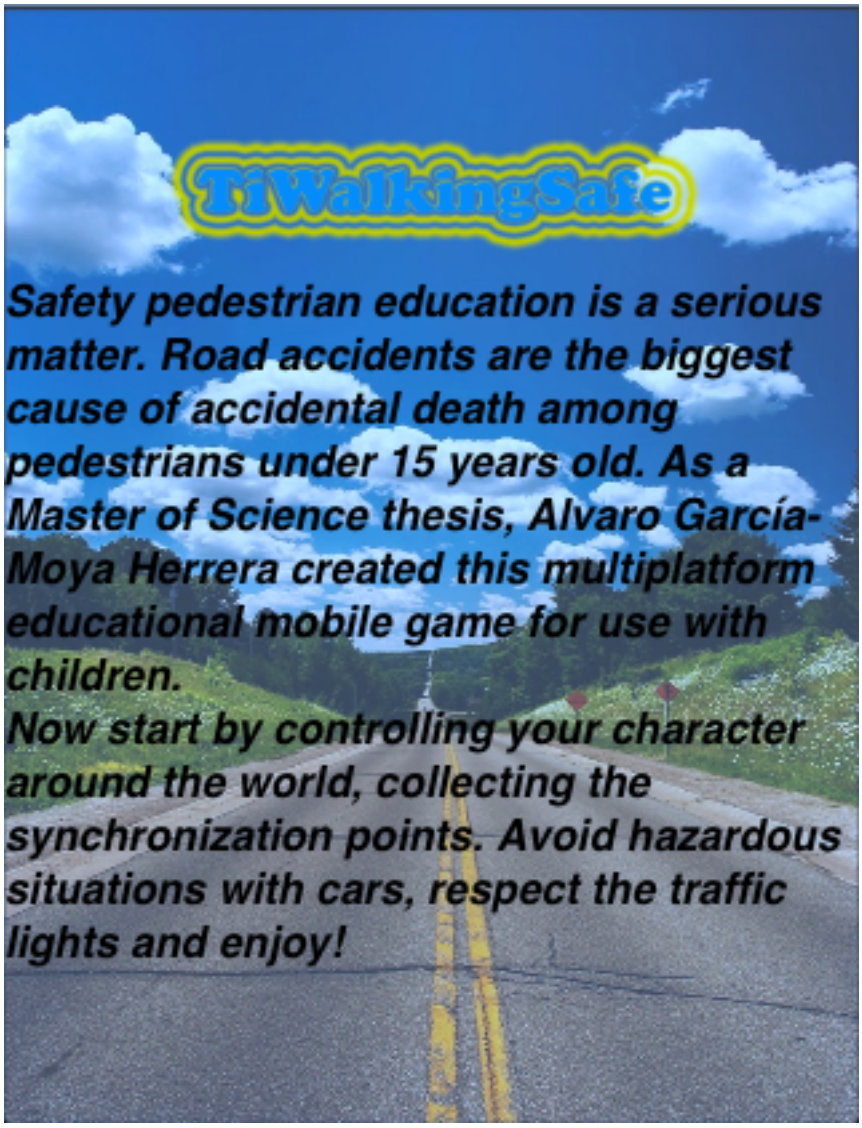

Figure 15. *Introduction interface*

Game interface

Figure 16 illustrates the game interface. This is the interface of the mobile application. The user uses the controller buttons at the bottom of the interface to direct the main character all over the world. Cars and traffic lights behave according to the application engine. The main goal of each level is to control the main character to reach the synchronization point in the world in the safest way. In the synchronization point, if the character is not located in the correct location, the synchronization with the point failed (Figure 17), and the user has to bring the character to the correct location to receive new instructions to go to a new location (Figure 18).

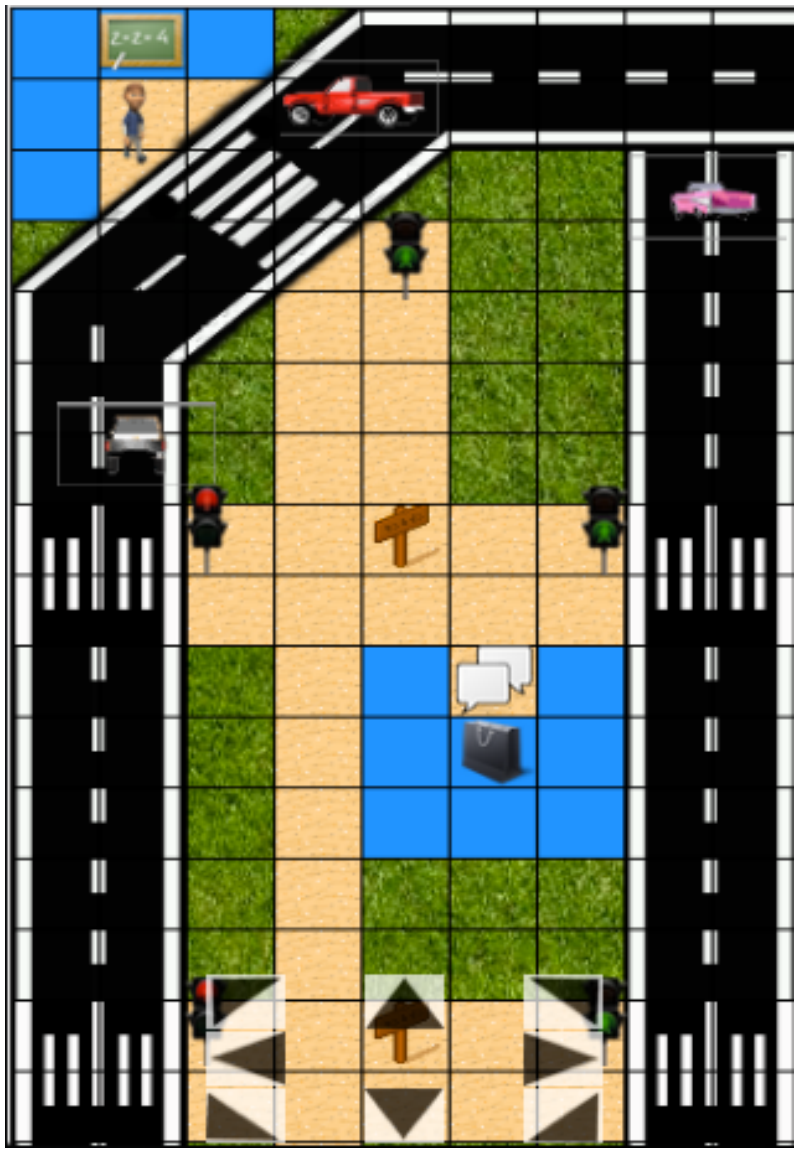

Figure 16. *Game interface*

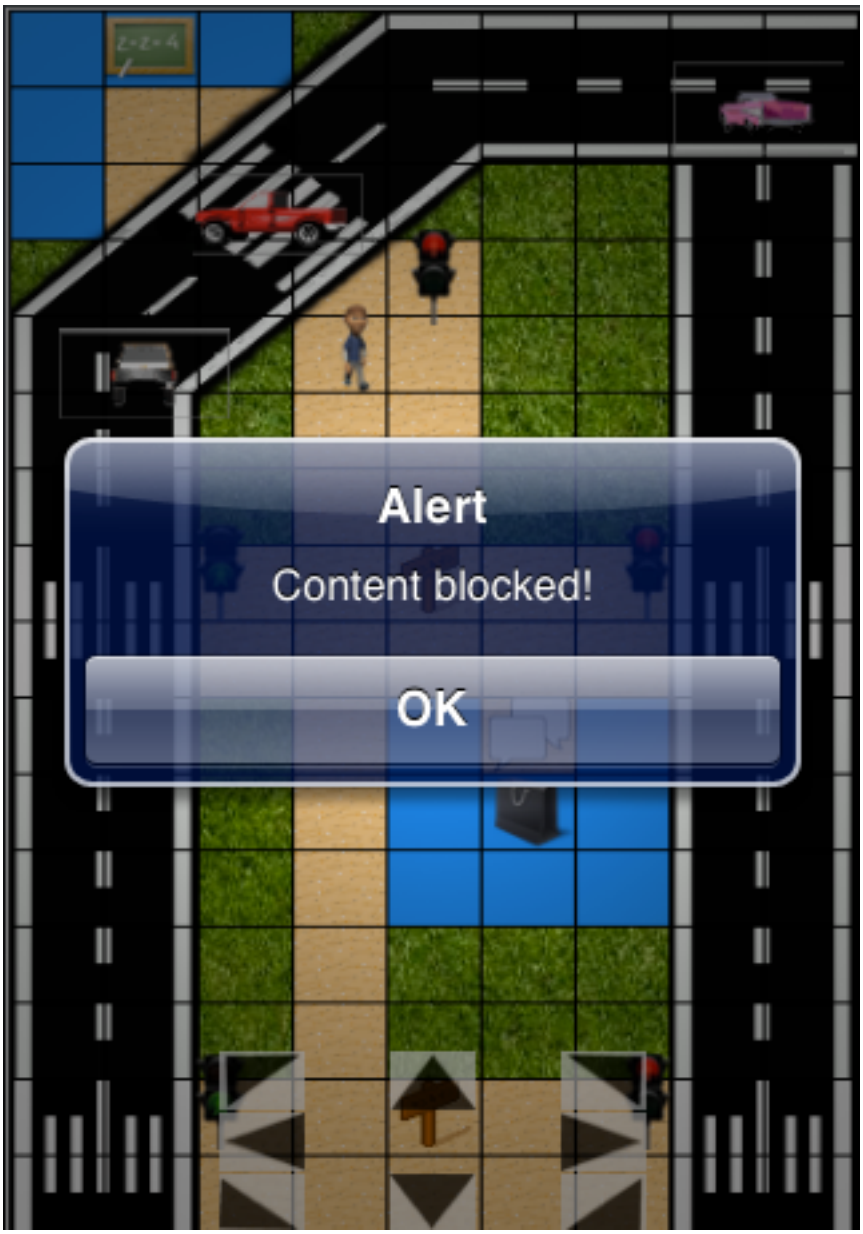

Figure 17. *Level incomplete interface*

Figure 18. *Level complete interface*