FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Flash Platform Games Engine

Marco Silva

VERSION 1.1

Report of Project
Master in Informatics and Computing Engineering
Supervisor: Professor António Augusto Sousa

2008, June

Flash Platform Games Engine

Marco Silva

Report of Project
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Professor Jorge Alves da Silva

External Examiner: Professor António Ramirez Fernandes

Internal Examiner: Professor António Augusto Sousa

July 31st, 2008

Abstract

The game industry is a multimillion dollar industry that attracts a lot of fans. More and more people look for new forms of entertainment from house consoles to handheld devices and the World Wide Web. In any platform there are games and there are players to play them.

The number of users of the Web has been growing at an incredible pace. Web users nowadays resort to the Web not only to get information but also services, to access and create online communities and for entertainment. Because of the crescent number of Web users the Web became an appealing market for game development companies as well as any other multimedia company.

However creating a game is a very complex process and therefore very expensive. Companies need tools that help them to create games more rapidly and effectively, reducing costs and increasing the gamers' satisfaction, giving them a larger variety of games to choose from.

The aim of this project is to create a game engine that can help developer to create any 2D Web-based platform game they need. This game engine will enable developers to create game in a more agile manner and can change aspects of the game interface and mechanics, easily and rapidly.

Resumo

A indústria de videojogos é uma indústria multimilionária que atrai um grande número de fãs. Mais e mais pessoas procuram novas formas de entretenimento desde consolas para casa a dispositivos móveis e passando pela Internet. Em qualquer plataforma, existem jogos e jogadores para jogá-los. O número de utilizadores da Internet tem vindo a crescer a um ritmo incrível. Os utilizadores hoje recorrem à Internet não só para obter informações mas também serviços, acesso e criação de comunidades on-line e procurando entretenimento. Devido ao crescente número de utilizadores da Internet esta tornou-se um mercado atraente para empresas de desenvolvimento de jogos e conteúdos multimédia.

No entanto, a criação de um jogo é um processo muito complexo e, por isso, muito caro. As empresas precisam de ferramentas que as ajudem a criar jogos mais rapidamente e de forma eficaz, reduzindo custos e aumentando a satisfação dos jogadores, dando-lhes uma maior variedade de jogos para escolher.

O objectivo deste projecto é criar um motor de jogo que ajude a criar qualquer jogo de plataformas em 2D para colocar na Internet. O motor de jogo permitirá que os programadores criem um jogo de forma ágil e tornará fácil a alteração de aspectos da interface do jogo e da sua mecânica.

Acknowledgements

To Prof. Augusto Sousa, my supervisor, for all the help and support he gave me during all the years at FEUP, always proposing new challenges and creating opportunities.

To Pedro Gaspar, my coordinator at FullSix, for all his help and support during this project.

To Tiago de Andrade e Silva, CTO at Fullsix, for believing in me and giving me this great opportunity.

To my family and friends that always supported me and help me giving me encouragement and ideas so that I could go further.

To my girlfriend Susana Vilaça, for all the support and incentive that she gave me that allowed me the best that I could be.

Marco Silva

Contents

List of Figures

List of Tables

Abbreviations

2D	Two Dimensions
3D	Three Dimensions
AI	Artificial Intelligence
AIR	Adobe Integrated Runtime
AJAX	Asynchronous JavaScript and XML
CLR	Common Language Runtime
DOM	Document Object Model
FPS	Frames per Second
GUI	Graphic User Interface
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
OS	Operating System
RIA	Rich Internet Applications
SUS	System Usability Scale
SWF	Shockwave Flash (a compiled Flash file)
UI	User Interface
WPF	Windows Presentation Foundation
XAML	eXtendible Application Markup Language
XML	eXtendible Markup Language

Chapter 1

Introduction

The Internet has been experiencing a huge growth not only in number of users but also in number of contents and available technologies. As the number of users is growing, so are their needs, which create great market opportunities for Web companies.

Nowadays, Internet users are not limited to accessing the Web's information; they can also create their own information. Users have new tools and technologies that make it possible for anyone to publish information on the Web. One of the great examples of the mass usage of the Web and its growth are Blogs. A Blog is a tool that allows the common user to publish his ideas and thoughts on the Web without having any knowledge about Web development. Blogs can be seen as a form of service provided to users over the Web, and as any service it evolves to meet the user's needs. Other Web tools followed this technology wave that hit Web users like YouTube, a tool that allows users to post online their videos, or Flicker, a tool that allows users to create online photo albums and share them with others, Wikipedia, an online encyclopaedia that users can consult and even add content to it.

The increasing usage and technological boost that is felt all over the Web is a result of a new trend in the World Wide Web technology and design called Web 2.0. This new trend's goal is to enhance creativity and information sharing but most of all, its goal is to increase collaboration among Web users. This trend led to the development of new user centred tools and services, with it, new market opportunities and a better understanding of the Web users needs appeared. With this rapid growth of the number of Internet users, companies started to give more importance to placing online contents and services for users.

With more and more users using the Internet and companies wanting more complex and innovative Web sites and Web services, Web developers also started to need better tools and technology so that they could create better products for their clients. To satisfy the developers' needs, new and more powerful tools have been created, tools that allow them to create more interactive and creative Web products.

1.1 Scope

This project is integrated in two main areas:

- Game Development
- Rich Internet Application Development

The development of Web services applications like videogames is very similar to build any other type of interactive Web application. The major difference is that games are focused even more on the interactivity part of the application.

Game development is the process of creating a game. This process includes the programming of the game's logic and mechanics, the development of its visual, sound and written contents, and the testing.

In the other hand, Web games are a form of rich Internet applications. A rich Internet application is a Web application that, unlike normal Web pages, gives the user a more interactive and graphical environment. Just like in desktop applications, rich Internet applications feature things like animations and are highly interactive. Because a Web game is considered a type of rich Internet application, its development will follow the same guidelines and processes.

This project is developed for the company FullSix, which is an international company that develops interactive marketing solutions for a number of clients. In Portugal, FullSix works with clients like TMN, Nokia, McDonald's, Super Bock, Santander among others; it has grown into a medium sized company, with a work force of more than 100 employees.

1.2 Project

FullSix develops a great number of interactive marketing campaigns. Some of these interactive campaigns involve the creation of Web games that are included in the campaign's Web site. Because game development is a very complex and expensive part of a project, FullSix needs tools that can help developers and reduce the cost and risk of including games in their projects.

Video games have a large number of different types, from puzzle games to role playing games, each type having specific structures and mechanics. Because of this fact this project is focused only on the 2D platform game type.

What FullSix needs is a tool that can be integrated in the company's workflow to easily build any type of 2D platform game. For this purpose the tool is composed by two parts. The first is a dialect that is used to define all the elements and mechanics of the game. The second part is a game engine that understands the game dialect and that creates the game based in the information defined by the developer using the dialect. With this tool developers can focus only on the definition of the game and instead of putting effort into the implementation details.

1.3 Motivation and Goals

As Web-based video games continue to grow, new and more specialized tools are needed to help developers to create better gaming experiences. A video game is a very complex piece of software and takes some time to be developed, which makes it expensive. With tools that reduce the development time to a minimum a company can invest more in creating more immersive and innovative game experiences that can add even more value to the game, and tools like the one that was developed in this project pretend to accomplish that.

The goal of this project is to produce a tool to help developers at FullSix to easily create 2D platform games. Developers should be able to easily implement and change any of the game's mechanics or graphical elements, creating a dynamic development environment.

1.4 Document Structure

This document is composed by eight chapters, being the first this one.

The second chapter describes the tools and technologies available to develop rich Internet applications and Web-based games.

Chapter three describes the project's requirements and the proposed architecture, which is divided into two parts; the first describes the XML dialect built to deal with some of the project's requirements, and the second one describes the architecture of the developed game engine.

Chapter four describes the main aspects of the project's implementation, as well has the project's planning and the main libraries and design patterns that were used in the development to solve some requirement and development problems.

Chapter five describes the tests made to the game engine. It is divided into four main sections, the first describing an usability scale used to evaluate the user's satisfaction concerning the system's usability and each of the others describing one of the tests that were made and the respective results obtained. The second section describes the system's usability tests made with developers, the third is concerning the system's usability tests made with designers and, finally, the last section describes the performance tests made to the system.

Chapter six describes the conclusions drawn based on the test results described in the previous chapter and the system's fitness analysis for implementing a game described as a proof of concept for this work. Also a critical analysis of the project is made, based on the requirements that were defined, the conclusions that were drawn and all the research made through the project.

Finally chapter seven describes the future work that is expected to be made to improve the project. The future work described is based on the problems raised as a consequence of the system's tests and the conclusions that were drawn in the previous chapter.

Chapter 2

Interactive Web-based Technologies

Nowadays, interactivity is a very important aspect of any type of application. Technology is not only used by experts but by the common users as well. People see technology nowadays as the most efficient way to execute everyday tasks. We use interactive technologies on our cell phones, our personal computers, our cars, our televisions and on the Internet.

Tim O'Reilly defined this new generation of Web technology and software the Web2.0. This new concept defines a new generation of software that takes into account new aspects that were overlooked like user experience, interactivity, usability and new design patterns. These new needs of users created the need of new development platforms that enable software developers and designers to create adequate applications for these users [ORe05].

With the creation of new technologies for developing more interactive Web applications, came the creation of a new type of applications, the Rich Internet Applications. This new market opportunity boosted the development of more interactive Web-based technologies and applications for both, developers and end users.

These new technologies created also a new market for games. With the creation of highly interactive Web-based technologies, games could now be developed for a new platform, the Web [Sol07].

2.1 Rich Internet Applications

The term Rich Internet Applications (RIA) was introduced in a March 2002 whitepaper by Macromedia (now a part of Adobe). Although the concept behind RIA was also referred with other names like *Remote Scripting*, *X Internet* or *Rich Web Applications*, the name *Rich Internet Applications* is the most commonly used.

RIA are Web applications that, intend to be just like a desktop application. The idea is to bring to a Web environment all the functionalities and features that the user is used to see in desktop applications. This includes the user interface, the dynamic interactive environment, the user experience issues and the business logic issues. This system's architecture has its benefits and restrictions as shown in the following subsections [Sol07].

Just as interactivity opened the door for computer games in the desktop environment, the same happened with RIA. The creation of new development platforms for interactive Web-based application opened the door for Web games and so the developer of these Web games will have to understand the benefits and restrictions of RIA.

There is a large set of development platforms for RIA. According to O'Rourke [ORo04] here are some examples:

Java Applets – Java applets can have access, to the screen (to draw inside a designated area of the HTML page), and to the keyboard and mouse of the computer where the Web page is opened.

Google Web Toolkit Framework – The Google Web Toolkit (GWT) allows developers to create JavaScript and AJAX (Asynchronous JavaScript and XML) based RIA, using Java language. This toolkit transforms the Java code for the user interface into cross-browser-compatible JavaScript that can be embedded in the Web page.

JavaScript and AJAX – Nowadays JavaScript is installed in almost every browser. This scripting language allows DOM (Document Object Model) manipulation and with the use of AJAX creates the development platform for RIA.

Microsoft Active X controls – A way to create a RIA is to embed Active X controls into a HTML page. The problem with the usage of Active X controls is that the only browser that recognizes them is Internet Explorer.

Adobe Flash and Flex – Flash and Flex are the two Adobe RIA development platforms. Flash was the first development framework to be launched (at the time it was launched by Macromedia). More recently Flex was launched, which is a new framework that uses MXML (an XML dialect created by Adobe) to define graphical elements. Like Flash, Flex is a framework for developing RIA, although more powerful than Flash. Adobe also released Adobe AIR (Adobe Integrated Runtime), a runtime platform that allows AJAX, Flash and Flex applications to be installed as desktop applications. Adobe Flash is described in more detail in section 2.2.2 - Adobe Flash.

Microsoft Silverlight – Microsoft Silverlight is a subset of Windows Presentation Foundation (WPF) that allows the development of RIA. Silverlight uses XAML (an XML dialect created by Microsoft) to describe graphical elements and animations. The client has to install a Silverlight plug-in to view Silverlight embedded Web pages. Microsoft Silverlight will be described in more detail in section 2.2.3 - Microsoft Silverlight.

2.1.1 Benefits of RIA

Although a Web browser was not originally designed to support the type of usage intended with a Rich Internet Application (which raises some development issues that do not exist when developing regular desktop applications), there are some aspects that justify the effort [Nod05]. These are:

Distribution and Installation – The fact that users access the application directly through the Web site drastically reduces the distribution and installation costs. This also makes it easier for inexperienced users to use software because they don't need to install it.

Updates and Upgrades – Because the software is accessed over the Web, any update is made on the Web server, thus being totally transparent to the user and at the same time guaranteeing that all the users are using the same software version. This is the same for big system upgrades (like changing the type of Database used in the system or expanding the system adding more computers), they can be done without the user even knowing about it.

Delocalization – The fact that the user does not have to install any piece of software in his machine makes it possible for him to use the software from any computer. This allows the user to carry the information he needs with him, but also to carry the software he usually uses as well.

Operating System Independence – Because the Web was created in a way that it could be platform independent, the application built on the Web will also be platform independent. The only thing that the user will need in order to use the application is a Web browser compatible with the operating system he is using. The developer will have to be careful so to prepare the application for multi-browser support.

There are also some development and performance benefits of using a client engine, such as:

Network Efficiency and Client/Server Performance – The fact that the client-server communications are reduced to the minimum reduces the network traffic. This alone can improve the systems performance (reducing the server response time).

Asynchronous Communication – The developer can use asynchronous calls to the server to get data that the user did not ask for but that the software estimates that he will ask for (like pre-loading the targets of the links in the current page). This will reduce the response time and give a better user experience.

All of these points that are introduced by RIA, creating a new form to see and sell software, not as a product but as a service that companies give other companies or even end-level users.

2.1.2 Restrictions of RIA

There are some restrictions from the user and the developer's point of view when talking about RIA. These are [Nod05]:

Enable Scripting – Often RIA applications require some type of scripting language to be processed by the Web browser (like JavaScript for example). For this reason the user must have previously enabled scripting on his browser or the application will not work properly.

Script Download Time – The fact that the Web page has script files embedded on it makes the download time longer and because the application's logic is partially on those script files, the overhead download time implied is important.

Loss of Integrity – The fact that the Web pages were originally designed for static contents makes it harder for RIA to dynamically change the content. The only way is for Web browsers to allow for scripts to alter the Document Object Model (DOM) of the Web page, making it possible for developers to create RIAs and impossible for the browser to guarantee integrity.

Web-Crawler Visibility – Because content can be dynamically altered and even embedded inside imported objects (such as Flash Player objects), Web-Crawlers cannot index the information on the Web site.

Internet Connection Dependency – The fact that the user has to have an Internet connection to access his information or the application can also be a problem. Although the developer can bypass this problem implementing an offline/online architecture and some synchronization functions, the user will have to connect himself, at some time, in order to trigger

the synchronisation. This type of bypass can also lead to synchronization issues that, themselves, can lead to loss of data.

Accessibility – RIA have a lot of accessibility problems. The main issue is the fact that screen readers have trouble reading inside Flash Player objects or detecting changes in the Web page.

Sandboxing – RIA run within a sandbox, which limits the access to the system's resources. A more detailed explanation on sandboxes can be found on subsection 2.1.3.

There are also some known issues when developing RIA. There are some service level management challenges that developers have to overcome if they wish to successfully deliver their online application. The aspects of RIA architectures that confuse the management process are:

Harder Development – The development of a RIA application is harder then developing a desktop application. Creating an application that will run on any Web browser is a concern that the developer must have, for example. Testing and debugging a Web application is also harder than when doing the same in a desktop environment. All of this lengthens the software development cycle.

The Web Page Paradigm – When developing a traditional Web site it is known that it is composed by a number of Web pages, each of which requires a distinct download (that is associated with a HTTP GET request). RIA completely breaks this paradigm as the application manages itself the content to download.

Asynchronous Communication – Managing asynchronous communications is hard and can lead to synchronization problems. When developing the server/client communication these problems have to be thought and addressed conveniently.

Performance Measuring – Because part of the application will be running on the clients end and also because of the asynchronous communications it is very hard to measure the application's performance.

2.1.3 The Sandbox

A Sandbox is a security mechanism for running unknown or untrusted applications. This security mechanism mainly consists on creating a controlled environment where the application is executing. The sandbox typically provides the program with a limited amount of resources (limited access to system's memory, cannot write to hard disk) and usually limited access to system's information, also network or input devices are disallowed or conditioned. In this sense, sandboxes can be seen as a form of virtualization. [Wik08]

Some examples of sandboxes are:

OS Virtual Machines – A virtual machine emulates a complete host computer, running on a native operating system. The access of the guest operating system to the host resources is limited because it can only access it from the emulator.

Java Applications – A java applet runs on a virtual machine that has limited access to the system resources. In the case of Java, the applet runs on the Java Virtual Machine that sandboxes the application.

JavaScript – JavaScript code is interpreted by the browser. The browser limits the script access to system's resources.

Adobe Flash Player Objects – A Flash object that is embedded in a Web page is sandboxed by the Flash Player running on the browser.

Silverlight Objects – A Silverlight object embedded in a Web page is also sandboxed by the Silverlight player plug-in running on the browser.

Understanding this type of security mechanism is very important when trying to develop RIA, mainly to better understand the platform limitations and to avoid future problems in the development and maintenance of the application.

2.2 Web-based Development Platforms

When creating a RIA such as a Web-based platform game, some decisions have to be made. The first decision is which technology to be used. When developing the game engine for the game, the same decision has to be made, because the technology used to develop the game engine will be, in principle, the same as used for the final game (at least for full compatibility).

To make this decision, a full analysis of main development platforms must be made to better decide which one is better suited for developing and deploying a Web-based video game. The two main development platforms in question are Adobe Flash and Microsoft Silverlight.

2.2.1 The Evaluation Matrix

To evaluate the different development platforms that are described in the next section an evaluation Matrix was used as an evaluation heuristic to support the decision of which platform is the best candidate for the development of this Platform Games Engine.

The evaluation will be based on a list of attributes of each development platform. These attributes are:

Platform Market Share – This attribute represents the number of Web users that have installed the plug-in necessary to view the contents developed using the given platform.

Platform Performance – This attribute represents the expected performance of the platform. The objective is to compare the different systems in terms of performance.

Plug-in Size – This attribute compares the size of the plug-in of each platform.

Multi-System Compatibility – This attribute compares the platforms in term of compatibility with different operating systems. The value of this attribute depends on the number of operating systems that are compatible with the each platform's plug-in.

Multi-Browser Compatibility – This attribute compares the platforms in term of compatibility with different Web browsers. The value of this attribute depends on the number of browsers that are compatible with the each platform's plug-in.

Support for Mobile Devices – This attribute compares the platforms in term of Mobile Devices Support. The value of this attribute depends on the availability plug-ins for that platform.

Programming Languages – This attribute compares the platforms in terms of the power of the programming languages they support. Aspects like Object-Oriented programming, number of compatible programming languages, compiler error detection power and language debugging capabilities are taken into consideration when evaluating this attribute.

Integrated XML Support – This attribute compares the platforms in terms of integrated XML processing support. The value for this attribute depends if the platform has "easy-to-use" tools for XML processing.

Integration with Other Systems – This attribute evaluates the platform in terms of easy integration with other systems or platforms, for example integrated with a data base or a server side application.

Acceptance at FullSix – This attribute measures the acceptance of the platform at FullSix enterprise. The number of people that know how to work with the platform and the number of licences the company has for the development tools needed are taken into account when evaluating this attribute.

The evaluation of each of the attributes will be made using a 0 to 5 scale (0 meaning that the system totally fails to fill the needs defined in the attribute and 5 meaning that all the needs are met).

Because the different attributes have different importance weights in the success of this project, the contribution given by each attribute to the total score must take into account this importance. For this reason, the attributes where divided into six levels of importance, being level 1 the most important and level 6 the least important. The attributes are distributed in the following manner:

Level 1 – This level is composed by the attributes Market Share, Multi-System Compatibility and Multi-Browser Compatibility. The reason for these attributes to be considered the most important is the fact that they condition the users that will effectively be able to play a game made with the Game Engine.

Level 2 – This level is composed by the attribute Performance. This attribute is also very important because the platform must have good performance values so that the Game Engine can also have a good performance. Video games have high performance demands and the platform has to be able to satisfy these needs.

Level 3 – This level is composed by the attribute Mobile Support. This attribute is important when considering that the mobile entertainment market is rising as well as its Web usage. This can be a great opportunity in the future if the developed Game Engine can be used to develop for these devices.

Level 4 – This level is composed by the attribute Acceptance at FullSix. This factor conditions the re-usage of the platform and its integration in the company's workflow.

Level 5 – This level is composed by the attributes Programming Languages and Integrated XML Support. The system's support of multiple programming languages and integrated XML processing tools can help in the development process of the Game Engine as well as future upgrades.

Level 6 – This level is composed by the attributes Plug-in Size and Integration with Other Systems. The size of the Plug-in can be seen as an obstacle in the growth of market share because users do not usually download and install big software packages in order to see some content on a Web page. Another thing considered is if it is easy or not to integrate other technologies with the given system. Both of these attributes have a small weigh in the Game Engine's success.

The weight distribution to each attributes is so as shown in Table 2.1.

Table 2.1: Evaluation Matrix Weight of each attribute

	Weight
Market Share	20.00%
Performance	15.00%
Plug-in Size	2.00%
Multi-System Compatibility	20.00%
Multi-Browser Compatibility	20.00%
Mobile Support	10.00%
Programming Languages	3.00%
Integrated XML Support	3.00%
Integration With other Systems	2.00%
Acceptance at FullSix	5.00%
Total	100.00%

2.2.2 Adobe Flash

Adobe Flash, previously known as Macromedia Flash and Shockwave Flash, is a multimedia development platform that is distributed by Adobe Systems. This technology was introduced by Macromedia in 1996 and since then has become the most commonly used way of adding animation and interactivity to Web pages. At first, Flash was used to add animated banners and advertisements to Web sites; now, Flash is used to create animated menus, contents and also to create the entire Web site or Web application.

Nowadays Adobe Flash player is a very common Web format. Adobe estimates that 99.3% of the Internet desktop users have some version of flash player installed on their machines. The player is also compatible with multiple operating systems and has a version for handheld devices called Adobe Flash Lite. [Hal02]

The massive use of this technology allowed for the Web to become more and more interactive, and rich Internet applications started to become more common, followed by interactive Web-based games.

Flash's interactivity is obtained by joining the graphical part of the platform, that allows for the creation of the graphical elements that the user interacts with, and a powerful scripting language that allows developers to create the interaction logic of the application.

All of this growth and demand pushed Flash to evolve, and newer and more complex versions of the software have been released.

In the following section the key aspects of the Flash platform are described, namely the scripting language used in the platform. Finally the last section evaluates this platform based on the evaluation matrix described in section 2.2.1.

2.2.2.1 Adobe Flash's scripting languages

When creating rich Internet applications using Adobe Flash Web developers have to use Flash's scripting language to implement the system logic. This language is called ActionScript.

ActionScript is a form of ECMAScript that allows developers to manipulate the elements design in Flash allowing the development of rich Internet applications and Web-based games. [Moo03]

As Flash evolved, so did its scripting language, having three different versions released throughout these years. Given that the first version of ActionScript is completely out of date, only the two most recent versions are described in the next sections.

2.2.2.1.1 ActionScript 2.0

This version of ActionScript was released in September 2003. Because of the crescent use of ActionScript for more complex Web applications, the developers needed a better equipped language; this led to new features to be added, like compile-time code checking (to help developer detect syntax problem in the code) and a class-oriented based syntax. Although ActionScript 2.0 allows developers to define classes and object-oriented inheritance mechanisms such as super classes and class interfaces, when the code is compiled it is transformed in the prototype-based approach of the previous version (ActionScript 1.0). The reason for ActionScript 2.0 to still use the same based architecture as the 1.0 version is that the new Flash player needed to be compatible with the previous version so that Web users could still view all the contents produced with both versions.

Another feature added in this version was the possibility for developers to specify the types of the variables, allowing type mismatch errors to be detected in compiling-time. [Moo04]

2.2.2.1.2 ActionScript 3.0

This ActionScript version was released in June 2006. This new version constituted a fundamental restructuring of the ActionScript language so much that a totally new virtual machine was created to support this language. This fact allowed Adobe to really boost the system's performance and create a totally new programming model that is more adapted to the developers needs when creating rich Internet applications.

This new version added new features to the language such as:

- Compile-time and runtime type checking;
- Improved performance;
- Support for packages, namespaces and regular expressions;
- Unified event handling system based on the DOM event handling standard.

All these changes in this version made it not compatible with previous versions. To solve this fact the new flash player comes equipped with both virtual machines so that the user can still view content that was produced in a previous version of the platform. The Flash Player browser plug-in that supports this version of ActionScript has a total size of 1,5MB. [Moo07]

2.2.2.2 Evaluation Matrix

Analysing this development platform using the evaluation matrix defined in section 2.2.1, the following score was given:

Table 2.2: Evaluation Matrix for Adobe Flash

	Value	Weight	Weighted Score
Market Share	5	20.00%	1
Performance	4	15.00%	0.6
Plug-in Size	5	2.00%	0.1
Multi-System Compatibility	5	20.00%	1
Multi-Browser Compatibility	5	20.00%	1
Mobile Support	4	10.00%	0.4
Programming Languages	3	3.00%	0.09
Integrated XML Support	5	3.00%	0.15
Integration With other Systems	4	2.00%	0.08
Acceptance at FullSix	5	5.00%	0.25
Total	43	100.00%	93.40%

One important point has to be explained at this moment, the fact that the Acceptance at FullSix parameter got maximum score is due to the fact that FullSix uses Flash as the primary RIA development technology. Another important fact to point out is that FullSix's developers still use ActionScript 2.0, although the company has intentions of fully integrating the new version of ActionScript there are no users that use this technology.

2.2.3 Microsoft Silverlight

Microsoft Silverlight is the new tool created by Microsoft that allows Web developers to create Rich Internet Applications. Microsoft released the first version of Silverlight on April of 2008 and is preparing a second version (the latest preview is dated June 2008). [Wen07]

Silverlight is based on another Microsoft technology called Windows Presentation Foundation (WPF) that is a graphical subset of the .Net Framework 3.0. WPF comes already installed with Windows Vista and provides a clean separation between the user interface (UI) and the system logic. This separation is achieved by the use of XAML, a XML dialect that is used to define the aspect of the UI. Because all the system's logic is implemented in another language (for example C#), the UI's aspect is totally separate. Because Silverlight is a subset of WPF, not all the features included in WPF are available in Silverlight [Mor06].

Although Silverlight is a subset of WPF, to view its content, users do not need to have WPF installed on their computers. Microsoft created a plug-in for users to install that integrates with their browser allowing them to view the Silverlight content in Web pages. The Silverlight is a 2.0MB plug-in available for free download but it is only compatible with Windows and Mac OSX platforms. Also the plug-in is compatible only with some of the browsers used in these platforms. The compatible browsers are Internet Explorer, Mozilla Firefox, and Safari. Microsoft is also developing a version of the plug-in that will be compatible with Windows Mobile 6 [Wen07].

Microsoft also developed some authoring tools to be used by developers and designers when developing UI using XAML. These tools make up Microsoft Expression Studio. In Expression Studio, designers can find a visual way to create XAML based UIs.

Although Silverlight is a very recent platform, it is already very well adapted to the Web user's needs. Still Adobe Flash is in the market since 1996 and has a huge market penetration that took a lot of year to get, although Silverlight has shown a lot of potential it still needs more time so that users can start to see it, also Microsoft needs to convince developers to start to produce RIA using this technology.

In the following sections some of the key aspects of this technology are explained, such as how the XAML dialect works and which development languages are compatible with Silverlight. The authoring tools that Microsoft created so that Web developers could use to design their UI for their RIA are also described in section 2.2.3.1. Finally, in section 2.2.3.4, the technology is evaluated using the evaluation matrix described previously in section 2.2.1.

2.2.3.1 Silverlight's Authoring Tools

With the development of user interfaces being made using XAML and as part of Microsoft's investment in the Windows Presentation Foundation, Microsoft released the Expression Studio.

Expression Studio is a group of multimedia development applications that Microsoft has put at the disposal of multimedia developers to help them create their interactive applications using this new technology. Expression Studio in composed of five tools that are [Mic08]:

Expression Media – This tool is mainly a media manager that provides users with an integrated way to manage their media content.

Expression Web – This tool is intended for Web developers. It is mainly a What You See Is What You Get (WYSIWYG) Web site developer, much like Microsoft's FrontPage, although Expression Web brings a lot of new features.

Expression Encoder – This tool is a video encoder that is designed to help users encode videos for Silverlight Web streaming.

Expression Design – This tool is a vector graphic editor for users to design XAML interfaces and elements.

Expression Blend – This tool is a visual interface builder for building Windows Presentation Foundation and Silverlight applications.

The tools of Expression Studio that are relevant for this work are Expression Design and Expression Blend, because no development of Web pages or video content encoding will be made.

Microsoft Expression Blend and Design can be compared with Adobe Flash and Illustrator tools (although Expression Design and Illustrator have some differences), since they serve the same purpose, in spite of being very different. In the case of Expression Design and Abode Illustrator, both can be used by multimedia designers and their goal is to help them develop graphic element to be included in multimedia applications. Both Expression Blend and Adobe Flash are visual interface builders that allow multimedia developers to create rich Internet applications.

2.2.3.2 Silverlight Development Languages

Silverlight, in its stable release version (version 1.0 released April 2008), is only compatible with one development language, JavaScript. In this version, developers can use JavaScript to program the user interface logic using event triggering, manipulating element properties and adding and removing element instances. JavaScript is very well adapted to this technology because the

interface is described in a XAML file (which is a form of XML), and the Silverlight plug-in uses a form of DOM to represent the graphical elements of the application. It uses JavaScript like in HTML to manipulate the DOM of the plug-in [Wen07].

In the upcoming release of Silverlight 2 (formally known as version 1.1) the platform will implement the Common Language Runtime (CLR) of .Net Framework 3.0, making Silverlight compatible with any of the .Net programming languages, including VB.Net and C# for example [Wik082].

2.2.3.3 The XAML Dialect

XAML (which is an acronym for eXtensible Application Markup Language) is a declarative XML-based dialect developed by Microsoft and it is used in Windows Presentation Foundation framework. XAML is used to structurally declare the objects that compose a user interface and their properties.

One of the great aspects of XAML is the fact that it allows designers to produce the user interface of the application independently of the developers system's logic, due to the fact that the designer develops the XAML file and the developer the code files. This also has the advantage of allowing the production of totally different and new user interfaces, at any time, without having to alter anything in the system's logic.

XAML can be used in Microsoft Silverlight, Windows Presentation Foundation (WPF) and Windows Workflow Foundation, although when used in Silverlight 1.0 not all features and elements are available. With XAML, users can declare interface elements such as text boxes, buttons and input text boxes, as well as graphical elements such as rectangles, ellipses and lines. Using XAML designers can also declare animations of their interface elements, which are called storyboards in XAML [Mac06].

2.2.3.4 Evaluation Matrix

Analysing this development platform using the evaluation matrix defined in section 2.2.1, the following score was given:

Table 2.3: Evaluation Matrix for Microsoft Silverlight

	Value	Weight	Weighted Score
Market Share	3	20.00%	0.6
Performance	5	15.00%	0.75
Plug-in Size	4	2.00%	0.08
Multi-System Compatibility	2	20.00%	0.4
Multi-Browser Compatibility	2	20.00%	0.4
Mobile Support	1	10.00%	0.1
Programming Languages	5	3.00%	0.15
Integrated XML Support	5	3.00%	0.15
Integration with other Systems	5	2.00%	0.1
Acceptance at FullSix	2	5.00%	0.1
Total	34	100.00%	56.60%

Like it was said in section 2.2.2.2, FullSix uses Adobe Flash as its primary RIA development technology, nevertheless the company is starting to explore the potentials of Silverlight, but there are no ongoing projects at the moment that uses this technology.

2.3 High-level Frameworks for Web-based Development Platforms

There are several high-level framework based on the previously explained development platforms. The frameworks that are more interesting to this project are the ones that are used for game development. This is why the next sections are about Microsoft Popfly Game Creator and Papervision3D. Both platforms are used for the development of Web-based games. In the case of Popfly the games are based on the Microsoft Silverlight platform, as in the case of Papervision3D the games use Adobe Flash.

2.3.1 Microsoft Popfly Game Creator

Microsoft Popfly Game Creator is a new tool added to Microsoft Popfly which is a Web site that is part of Windows Live services. With Popfly, users have access to a number of tools for creating rich Internet applications using Silverlight. The goal of Popfly is to enable non-technical users to create their own applications, experiences and ideas and share them with their friends and even with the entire community [Lot08].

Using the Popfly tools a user can create a number of different contents like:

- Mashups
- Web pages
- Web games

Although Popfly itself was made available on May 2007, it is still in beta version and new features have been added since then. One of the latest features to be added was the possibility to create Web games using the Popfly Game Creator Tool. The tool is still in Alpha version but it is already available for any user for free (like any other Popfly tool)

This tool allows users to create any 2D game they want with a simple and intuitive What You See Is What You Get user interface as shown in Figure 2.1.

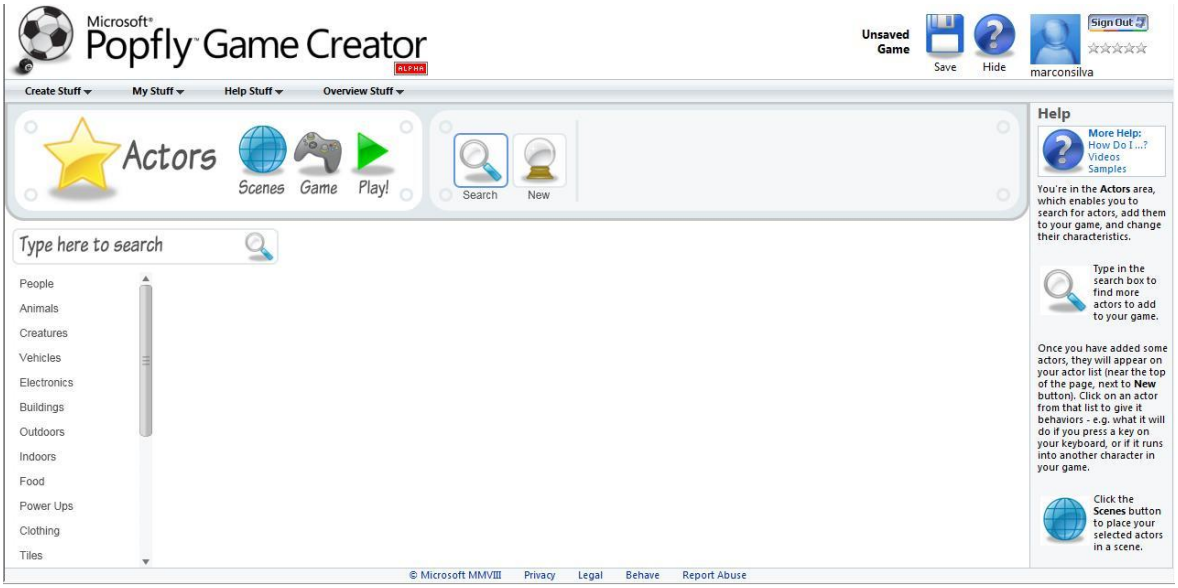


Figure 2.1: Popfly Game Creator's Homepage

To create a game, the user simply has to follow three steps. The first is to add the game elements (all the characters, objects, enemies) and add behaviours to them if needed (for example if implementing a soccer game you will need to add behaviours to the field goals such as if a collision with the ball element is detected then update the score board). Next, the user creates the games scenes. The scenes are like the screens of the game (for example, intro, game over and victory screens). Finally the player can create and manage the variables of the game (for example initialize the score variable for the game with zero, or create another score variable for a two player game). When everything is ready the user can click play and test the game.

One aspect that is particularly interesting is the interface for creating and managing element behaviours. This interface is shown in Figure 2.2.

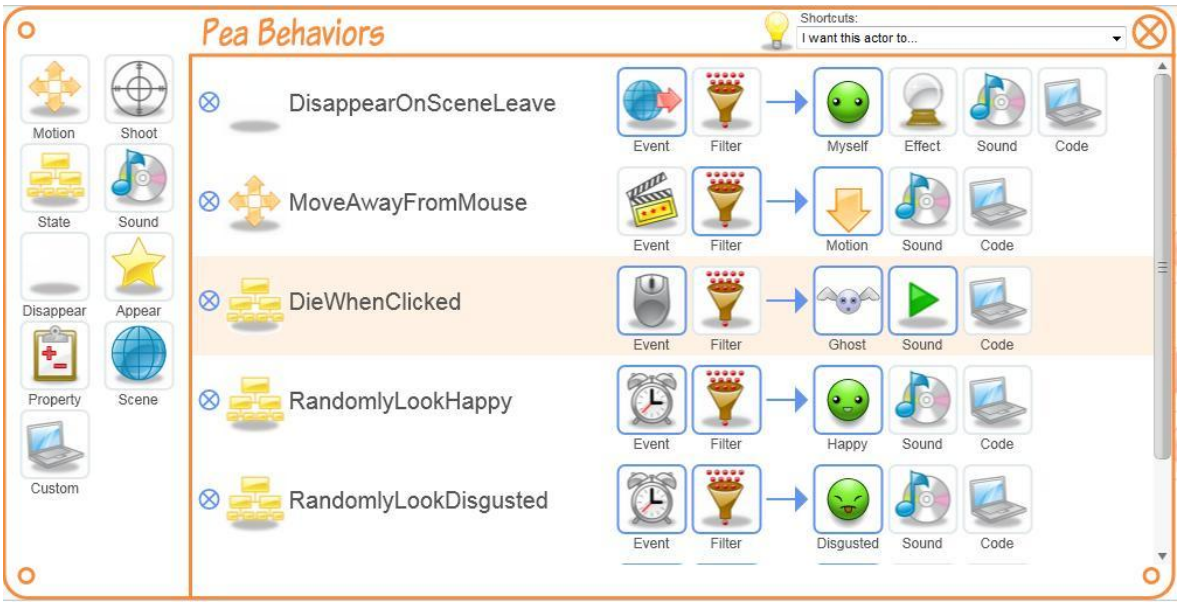


Figure 2.2: Popfly Game Creator Behaviour Manage Window

On the left of this window the user can see the different types of behaviours that he can add, and on the centre, a list of the element behaviours. Each of the behaviours has an event that triggers it and a filter, so that users can setup some exceptions (for example moving an element when the player presses a key, but filtered if the element is at the edges of the screen). Behaviours have also an action defined that is of the type of the behaviour (for example motion events have actions that move the element) and a sound that the user can define to be played when the behaviour is triggered.

This event triggering method is a very simple and clean approach for describing the mechanics of a game and makes it very easy for the user to understand.

2.3.2 Papervision3D

Papervision3D is an open source Flash-based 3D render engine developed by Carlos Ulloa, John Grden, Ralph Hauwert and Tim Knip. This engine consists of an ActionScript library that developers can use to create full 3D experiences in Adobe Flash. The library is very used and a lot of other platforms have been built on top of it. An example of it is the First Person 3D engine called Paradox [Fir07].

Many developers have also started to use this library to produce 3D Flash games and even 3D Web sites.

The next section shows some example of games developed using Papervision3D. Also there are some examples of what the Paradox engine can do.

2.3.2.1 Papervision3D examples

Papervision3D created the opportunity for Web-based game developers to create real 3D games and put them on a Web site for users to play. One of the downsides of playing Web games was the fact that all of them had to be 2D games. Nowadays users prefer 3D games like the ones they play in their desktop PC and living room consoles. [Spi08]

One example of a game Papervision3D-based game is the Bowling Buddies game that is available in Facebook. The fact that a large Web social network like Facebook is using this technology shows that more and more users like this type of 3D rich Internet applications. The game has a very simple look, as can be seen in Figure 2.3.

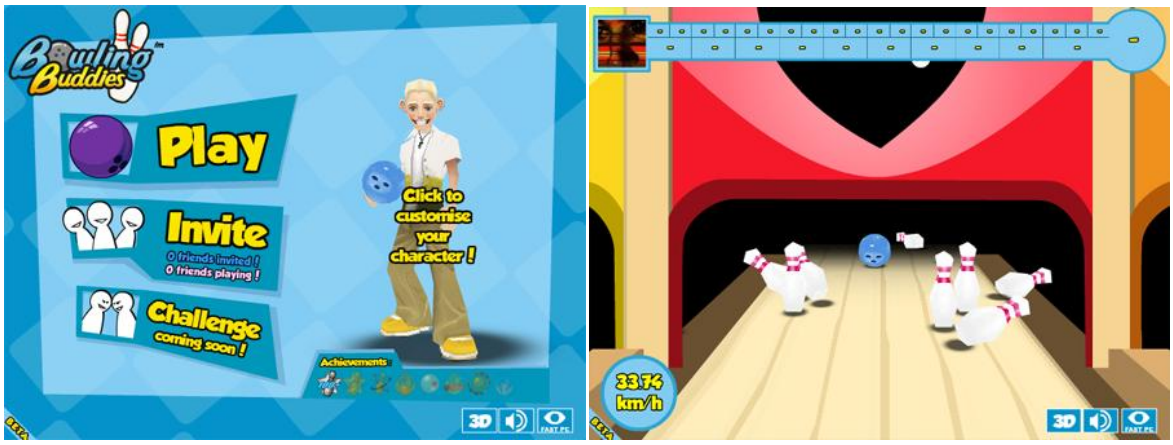


Figure 2.3: Facebook's Bowling Buddies Game

Another example of development based on Papervision3D technology is the Paradox first person 3D engine. This engine is still in development but two demos have been made available and the results are shown in Figure 2.4.

Figure 2.4: Screenshots of Paradox's demo projects

Here we can see some of the render features that Papervision3D has, like 3D lighting, 3D model rendering and texturing.

2.4 Technological Cross-Analysis

When deciding the technology to use in the development of this project, two considerations have to be made. The first one is which of the development platforms (Adobe Flash or Microsoft Silverlight) should be used, the second is to use one of the already existing frameworks or not and develop directly over the development platform.

Starting by selecting the development platform for the project some considerations have to be made. The first is to compare the evolution matrix results for each platform. The Adobe Flash platform got a score of 93.40% as Microsoft Silverlight got a score of 56.60%. Looking at both scores, only a decision is possible due to the large difference between them: the best choice is clearly the Adobe Flash platform.

Now that the platform is set, the decision left to take is either to use a Framework based on that technology or not. The fact that the selected platform was Flash leaves only the Papervision3D framework, because Popfly develops Silverlight games.

The papervision3D is a very good framework and some games have already been developed with it. The only question is the fact that the game, by specification, is a 2D platform game, and the biggest advantage of Papervision3D is the fact that it gives the possibility of working in a 3D environment. Given this fact, Papervision3D is not a good starting point for this project's development.

Although the development platform has already been selected, some aspects of the Popfly Game Creator are important to be mentioned and analyzed. Popfly is a great solution for a user friendly game creator, but it is not oriented for the development of platform games. Another obstacle to its usage in this work is the fact that Popfly is not an open-source project. Users are free to use its tools and features and create their own games, also they can download the source of the created game or embed the Silverlight element in their own Web page, but the Game Creator tool is not available for download and cannot be altered by the user. Nevertheless the study that was made about the platform brought some interesting perspectives about the way to create and describe a

game and its elements like the event-based architecture of the game that is described in section 2.3.1.

So the best decision is to develop the Platform Games Engine using Adobe Flash. The only thing that has to be selected is which of the ActionScript versions to use in the development. At FullSix developers still work on the 2.0 version of the language due to the fact that the company has not yet licensed the new version. One of the main goals of this project is that this tool is successfully integrated in the company's workflow. For this the tool used must be available at the company, so the best choice is the version 2.0 of ActionScript, for which the company has the licences.

2.5 Summary

The Web has been growing at an incredible rate. Each day the number of users and Web sites grow. The fact that the Web is based on HTML technology limits the usability and the type of content that could be made available. Users started to expect more from the Web; they needed a more interactive interface, with animation and dynamic content. To satisfy this need RIAs were created. These applications use new technologies that allow Web developers to create new designs for their Web contents, adding animation, interactivity, and a better user experience. Users and developers were very pleased with the potentials of RIAs and a large number of services started emerging.

Users started to use the Web not only to search for content but also for services (for example online shopping) and even for entertainment. This created the opportunity for the gaming industry to enter this market and create entertainment contents for these users. Web games were only possible thanks to the evolution of RIA technologies. At first games were limited to 2D, but nowadays there are new frameworks like Papervision3D, which allow the creation of 3D Web-based environments. Other frameworks allow the user to create its own game, using a simple game creation tool, like Popfly Game Creator.

The success of RIAs created the need for more advanced technologies, allowing developers to create more complex applications. This fact boosted the development and update of tools like Adobe Flash and Microsoft Silverlight. Both these technologies allow developers to create interactive Web-based applications. An evaluation of both technologies was made, to determine which is more adequate to develop this project. The result of this evaluation showed that Adobe Flash is far more adequate for this project than Silverlight. Although Flash has a totally new version with a lot of new features, FullSix is not yet using it due to licensing issues, and, so that the project can be used in the company's workflow, it had to be developed in a previous version, supporting ActionScript 2.0.

Chapter 3

Requirements and Architecture

The Client for this solution is FullSix Portugal. FullSix is an international group composed by 9 offices spread around the world and a team of more than 500 people. The company's core business is the design, development and execution of marketing programs for client brands. As a marketing specialist FullSix believes that innovative direct consumer interaction is the way to effectively boost results [Ful08].

In Portugal the FullSix group, composed by FullSix Portugal and SixandCo companies. The group is one of the most successful interactive marketing groups, having won a great number of awards (for example the Sapo Advertising Awards). [Sap07]

FullSix, as an interactive marketing company, produces marketing contents for a number of clients. These contents include company Web sites, banners, and even entire marketing campaigns (online, radio, TV ads, and other marketing campaigns). Among different products, they produce Web videogames that are put on client's Web sites or used on marketing campaigns (for example a campaign that has prizes for the best high score in the game).

Although the game may not be the centre part of the marketing campaign its development has a great impact in the project's lifecycle and has a big weight in the overall time and cost.

3.1 Problem Description and Scope

FullSix produces a lot of different games for their clients, some are action games, some are puzzle games and some are platform games. FullSix tries to recycle their game projects reusing some of the common parts of each project, such as some generic code classes or overall project architecture.

Although there are numerous types of videogames, this project's scope is the 2D platform games. The problem is that when creating a platform game most of the components were made to fit that specific project and little or even none of the work could be easily reused in other projects.

In a short sentence what FullSix needed was a tool to create platform games that could reduce their implementation lifecycle.

This tool should meet a group of requirements, like:

Expandability – They wanted a solution that they could expand if needed (For example easily adding a new module to the solution).

Reusability – The solution should be generic enough so that they could reuse it in any platform game.

Design Independent – The solution also had to be design independent so that they could implement the design template for any marketing campaign.

Web Page Integration – The solution must allow for the final game to be easily integrated into a Web page.

To solve this problem and meet all these requirements the proposed solution is the creation of a specific game engine for Web platform games that FullSix could use to build any game they need.

3.2 The Platform Game Dialect

The first part of the solution was to create a structured way to define all elements and mechanics of a platform game, so that a game engine could understand and create the desired game. This structure should be generic enough to allow for the definition of the entire game (all the elements and mechanics that compose the game). This definition is very important because it will be the means of communication between the developer, the designer and the game designer. All of them have to communicate with each other and determine how to put the pieces all together and make the game. Because of the nature of the problem and the fact that different skilled people (namely programmers and non programmers) will have to understand this dialect, the best technology to define the dialect is XML.

Using XML to define the game has a group of advantages, namely: [Har04]

- **Text-Based Language;**
- **Not Compiled;**
- **Well Defined Structure;**
- **Easily Parsed;**
- **Easily Updated and Changed.**

Although there are many advantages in the use of XML in this particular problem, there are also some disadvantages to consider, namely: [Har04]

- **Heavy Syntax;**
- **Hierarchical Structure;**
- **Documentation.**

Although disadvantages for the use of XML exist, they can be overcome with careful consideration when designing the game engine and with tools and documentation supporting users when using the defined dialect.

Taking now a closer look at what composes a game, we can see a group of levels, each one very different from the other. While defining the game dialect, this fact has to be taken into

consideration. The fact that the main part of the game mechanics and element definition is a part of each level implies that their definition should be made in the same XML file, which would make the game dialect very hard to read and very long. One solution is to create two, almost independent, dialects one to define the game and all is global elements and the other to define a game level. The game dialect will then only have to reference the level XML files and the game engine parser will have to be prepared to understand and process this architecture. The advantage of having each level in its own XML file is that when someone needs to make a change in a level only the elements related to that level are contained in the XML file, making it faster and easier to make the changes.

To define these dialects, a XML Schema was created for each one. These schemas purpose is to help the developer to understand the dialects as well as make it easier for him to validate a XML document before using it in the game engine. The full definition of these Schema documents can be found in appendix A: Game XML Dialect Schema Definition and appendix B: Level XML Dialect Schema Definition. [Vli02]

3.2.1 The Game Dialect

A platform game, as any game, is mainly composed by three big types of definitions [Ada03]:

System Configurations – The game has defined a group of configurations, those configurations are input keys and start-up variables, and they are the same for all the game levels.

List of Levels – A game is composed by a sequence of levels.

List of Player Characters – A game is composed by a group of characters that the player will use to overcome the levels obstacles and enemies. The characters can be different or common on each level, so all characters are defined as part of the game and not as part of a level.

These three main elements constitute the starting point for the definition of the game dialect, so the game dialect structure will look like the representation in Figure 3.1.

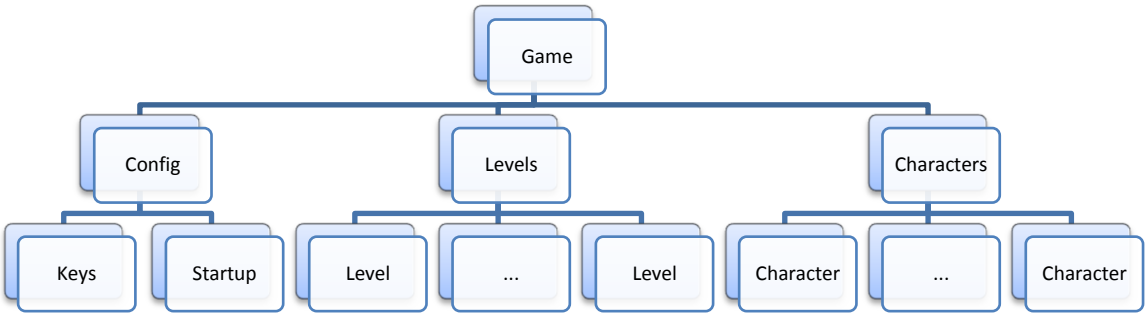


Figure 3.1: Structure overview of Game Dialect

This XML architecture contains all the elements of the game and groups them up into the 3 defined groups. Each group however has its own elements and some of them can be quite complex. In the following sections a brief summary of the composition of the three main elements is made. However, for a more detailed explanation, please refer to appendix A: Game XML Dialect Schema Definition

3.2.1.1 The Config Element Definition

The Config element is responsible for the definition of all generic system configurations. When parsing this node, the game engine should configure itself according to the settings defined in it. This element has two child elements, the Startup element and the Keys element, as shown in Figure 3.1.

The Startup element contains a group of other elements that are all optional and that define the corresponding variables of the game. Each element has an "amount" attribute that defines the value for the corresponding variable. The possible nodes that can compose this element are as shown in Figure 3.2.

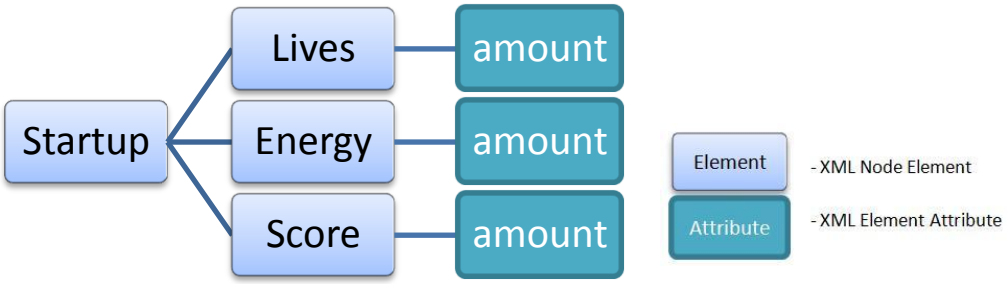


Figure 3.2: Structure overview of the Startup element

The Keys element contains the definition of the keys that generate events as well has the names of the events so that they can be referred in other parts of the document (namely in the character element).

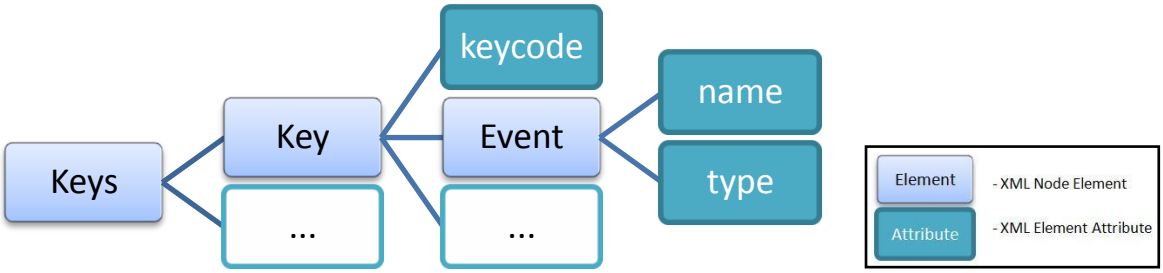


Figure 3.3: Structure overview of the Keys element

This element is composed by a sequence of Key elements (one element for each key the developer wants to configure). Each Key element has a "keycode" attribute and a sequence of Event elements. Each Event element has a "name" and a "type" attributes. The "name" represents the name of the event that should be launched and the "type" represents the type of key event the engine should be listening. The "type" attribute can have three distinct values, which are:

onKeyDown – The event should trigger only once, when the key is pressed.
onKeyUp – The event should trigger only once, when the key is released.

isKeyDown – The event should trigger several times (the number of times is decided by the game input processing loop) while the key is down.

This type of definition allows for the same key to launch different events depending on the type of key action that is received (for example to have an event when the jump key is pressed that puts the player character in jumping position and another event when the key is released that makes the player character jump). It also gives more control of the input to the developer that is creating the game, not only by making it easier to add, remove or change keys and startup variables, but also to control and better setup the hey the game gives visual feedback to the player.

3.2.1.2 The Levels Element Definition

The Levels element contains the definition of the sequence of levels that compose the game. Each level element points to the XML file where the game engine should find the complete definition of the level as well as to the character defined later in file. The order in which the levels are defined will be the order in which they will be played.

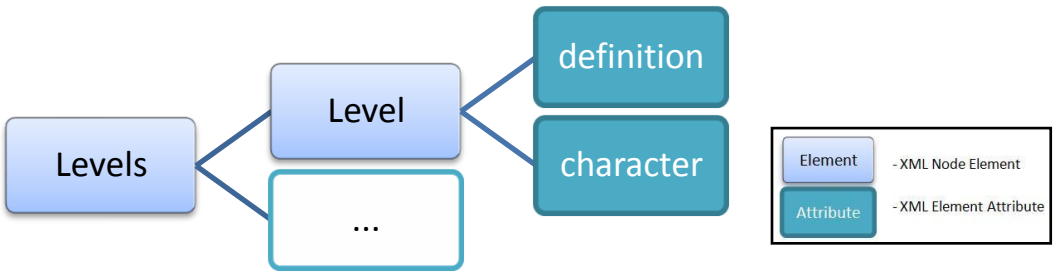


Figure 3.4: Structure definition of the Levels element

Because there is no way to force that the path for the level XML file has to be valid, the parser should be prepared in case the path is not valid and notify the developer with a error message.

3.2.1.3 The Characters Element Definition

The Characters element contains the definition of all the player characters that compose the game. This element is composed by a group of Character elements, one for each player character. These characters must be referred in the "character" attribute of a Level element so that the game engine knows on which level to use the character.

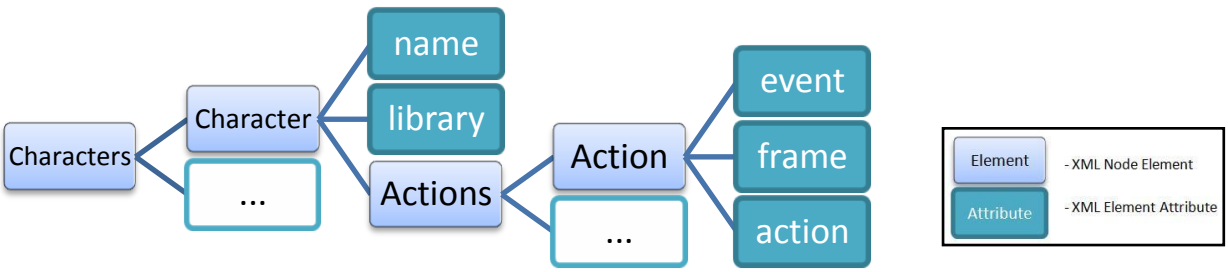


Figure 3.5: Structure overview of the Characters element

As shown in Figure 3.5 each character is composed by two attributes and a group of actions. The first attribute is the "name" that defines a unique identifier for the character so that it can be referred in other parts of the XML file (namely in the level element). The second attribute is the "library" and it defines the path to the file containing the graphical representation of the character for the game engine to import. The character can also do actions, which should be defined in the list of actions of the character. Each action is defined by the event that triggers it (in the "event" attribute), the frame of the graphical element that should be shown while the character is doing the action (in the "frame" attribute) and the type of action (in the "action" attribute). The types of actions are limited because they are pre-programmed in the game engine. Therefore the possible values for the attribute are also limited to guarantee that the game engine can understand them. The defined types are:

moveLeft – Gives the Game Engine the command to move the player's character left.
MoveRight– Gives the Game Engine the command to move the player's character right.
jump – Gives the Game Engine the command to make the player's character jump.

Although now there are only three possible actions for the character to use, this structure is expandable and, at any time, new actions can be defined and added to the game engine and game dialect. Also the fact that each character has its own actions definition allows for the developer to create totally different types of character, for example one that cannot jump or one that as different keys from other characters.

3.2.2 The Level Dialect

When looking at a platform game level, although each one is unique, we can find some general common aspects that define it, namely:

Level Mechanics – A level has its own mechanics and that is one of the parts that makes it unique. For example one level can have auto scrolling that forces the player to move constantly and another one can have free scrolling mechanic here the player can go back and pickup items he missed.
List of Game Elements – A level is also defined by the list of elements that compose it. These elements are things like the enemies, the items, the objects, and even the start and ending points of the level.

These general aspects of a level served as a guideline in the creation of a complete dialect that fully defines it. Some other aspects were also included in this dialect so that the game's developer had always the means to personalize every aspect of the level, which made is structure as shown in Figure 3.6

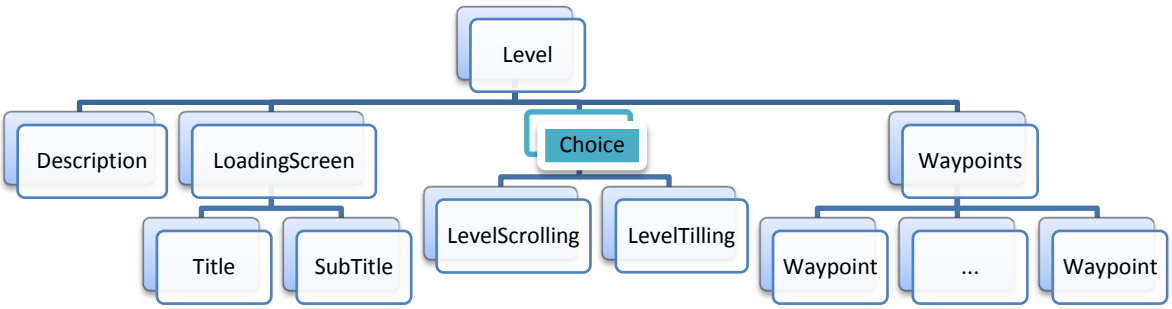


Figure 3.6: Structure overview for the Level dialect

The XML structure divides the level definition into four parts. The first is the level's description, next the loading, followed by the level mechanics part (which can be of the scrolling or tiling type), and finally the list of waypoints that defines all the elements in the game.

Both the Description element and the LoadingScreen element are optional and are not important for the understanding of the architecture of this dialect. For this reason, in the following sections, only the structure for the two remaining parts will be explained. A more complete description of this dialect can be found in appendix B:Level XML Dialect Schema Definition.

3.2.2.1 The Level's Mechanics Definition

When defining the mechanics of any game a full understanding of the game is needed. In this particular case of a 2D platform game, the level mechanics can be defined by a set of properties:

Direction – The level can be vertical or horizontal.

Navigation – The navigation made by the player character is important in the level mechanics. For example, if the level is horizontal, the player is presumed to navigate the level to the left, to the right, or both.

Automatic Scrolling – This aspect may create a different type of game mechanics, creating a challenge for the player when he plays the level. The challenge can be also defined by the speed set to the automatic scrolling.

Total Length – The total length of the level is also an important aspect to be defined because it will limit where the player can go in the level.

Type – There are mainly two types of 2D platform levels, the scrolling level and the tilling level. In the scrolling level, the camera is always centred on the player character and the background scrolls as a continued strip. In a tilling level, the scrolling occurs only when the player touches the edges of the screen creating an almost comic book like experience.

The possible combinations with all of these properties give a lot power to the developer and game designer to define a great number of levels. Because there are mainly two types of levels (the scrolling and tiling types) it is important to define two distinct XML elements that the developer will use to define the level's mechanics. Because a level cannot be both scrolling and tilling the

dialect has to be structured so that only one of the two types can be defined. In the next sections a brief definition of both elements is made.

3.2.2.1.1 The LevelScrolling Element

A scrolling type level has a group of properties that define his mechanics, thus defining the element structure as shown in Figure 3.7.

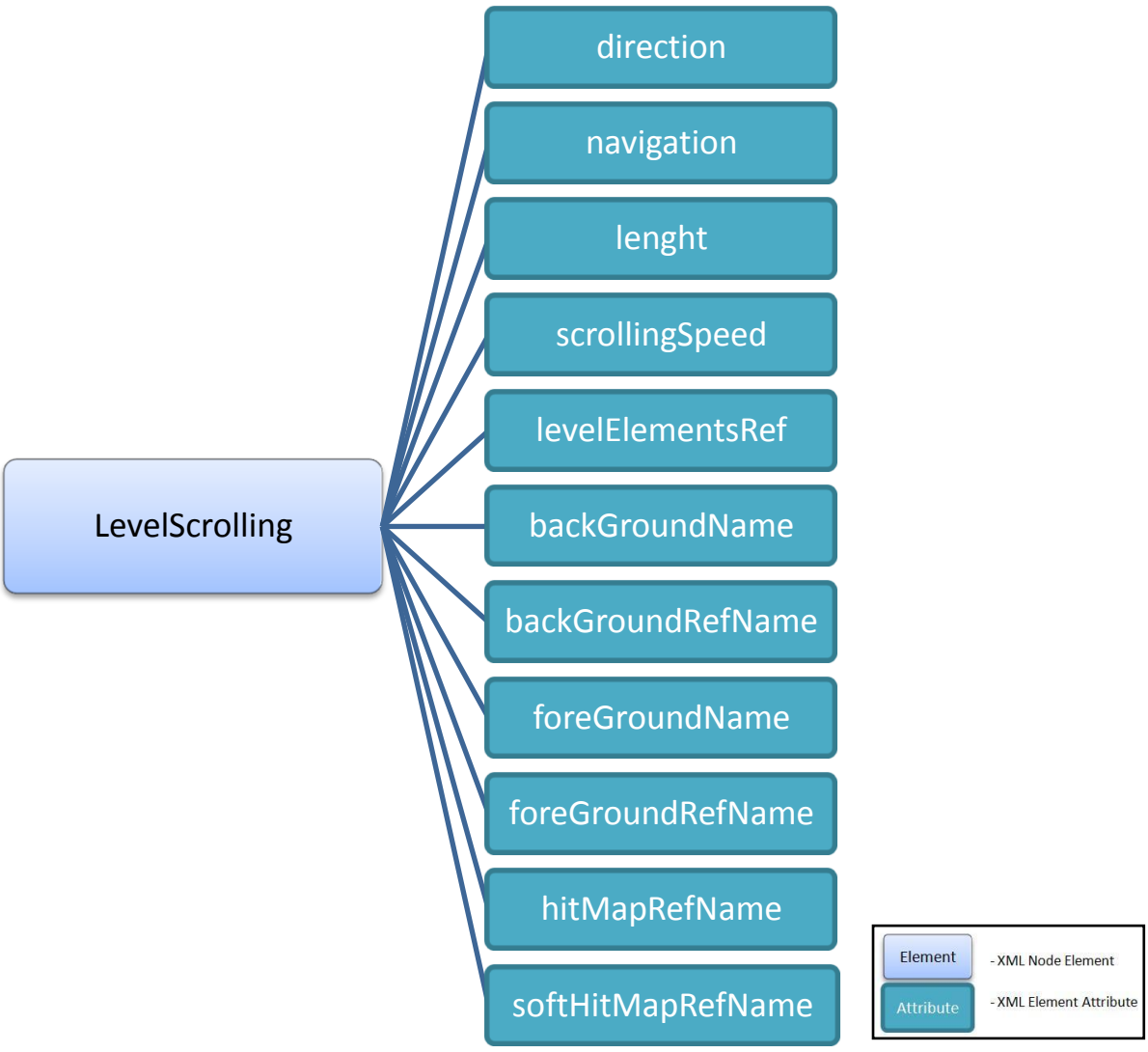


Figure 3.7: Structure overview of the LevelScrolling element

This element is only composed of a group of attributes, which are shown in Figure 3.7. The level scrolling mechanics are defined in the attributes “direction”, “navigation”, “length” and “scrollingSpeed”. The rest of the attributes define the objects in the main SWF file and the paths to the corresponding graphical elements to import, for example the “backGroundName” defines the name of the SWF object that should contain the background and the “backGroundRefName” defines the path to the file containing the image to load has the background.

This element allows the developer to create is level mechanics with little effort. One of the biggest advantages of using this element is that the game engine has all of these mechanics ready to

use and the developer only has to be concentrate on the way that the level should be built so that it gives the player a great gameplay experience.

3.2.2.1.2 The LevelTilling Element

A tilling level is composed by a group of tiles, and the player will go through the level, tile by tile. One of the main characteristics of this type of level mechanics is the fact that the level changes tiles when the player touches the edge of the screen. This element contains a group of properties that define the level's mechanics as well as all the tiles that makeup the level, as shown in Figure 3.8.

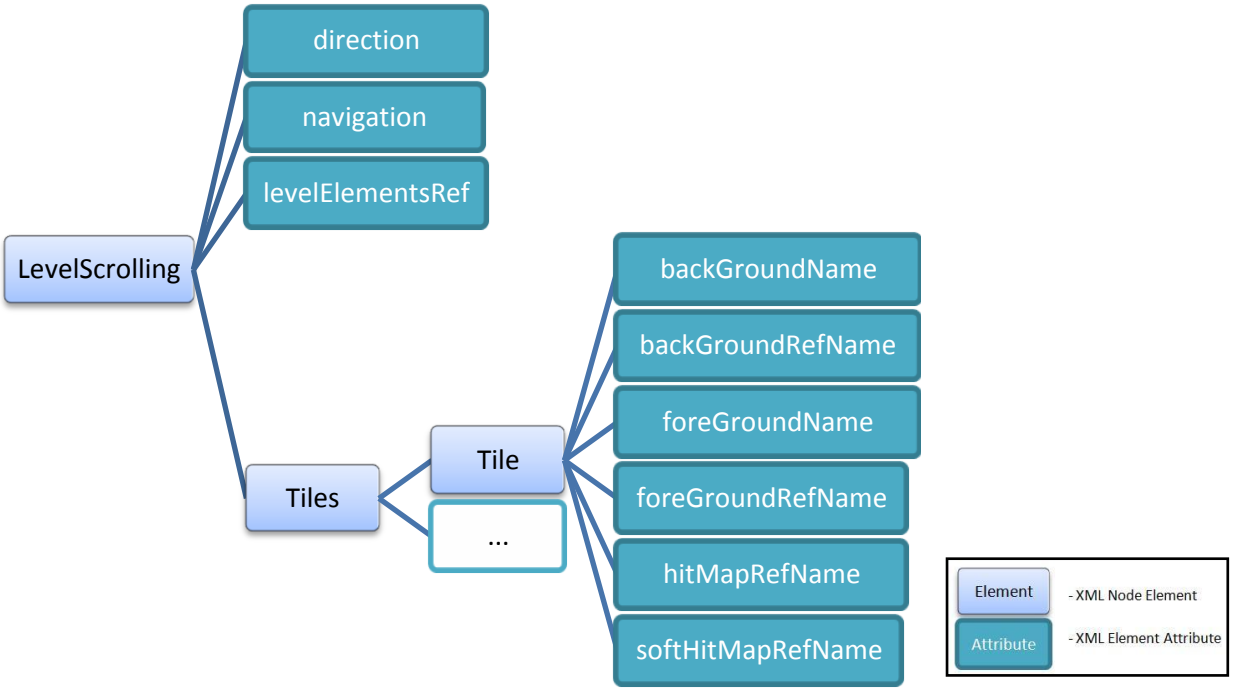


Figure 3.8: Structure overview for the LevelTilling element

Similarly to the LevelScrolling element, this element has a group of attributes that define some of the key aspects of the level's mechanics. The scrolling mechanics are defined in the "direction" and "navigation" attributes, followed by a list of tiles. Each tile has a set of attributes that, like in the LevelScrolling element, define the name of the some important objects in the main SWF file and the path to the graphical elements that should be loaded into these objects. The order of the tiles will determine the order in with the player will scroll through them.

This element allows developers to easily customize the mechanics of a tilling type level. This type of level, not only adds more possible game mechanics to the dialect, but also adds the chance for the game to have a better system management, because the engine will only need to have loaded the tile that the player in on and its neighbours.

3.2.2.2 The Waypoints Element Definition

A level is composed of a group of elements that are placed throughout the level. These elements can be enemies, items or objects, each with a distinct behaviour and its own set of properties. One way to see these elements is as a set of waypoint in the level, each one representing an element that is of a specific type.

This leads us to the definition of the Waypoints element as shown in Figure 3.6, as a container of a group of waypoints. Each waypoint can be of a different type:

Enemy Waypoint – Represents where to initially place a given enemy.
Item Waypoint – Represents where to place a given item.
Object Waypoint – Represents where to place a given object.
Start Waypoint – Represent where to place the player at the start of the level.
End Waypoint – Represents where the level end.
Enemy Spawn Point Waypoint – Represents where to place a spawn point for enemies.

The fact that there are these different types of waypoints suggests that each of them is unique in some sort of way, and evidently the game engine will need to distinguish these waypoints as well.

This creates the need to describe each of the types separately making it easier for the game engine to process and clearer to read in the document. The best way to do this is to have an element for each for the types. This model also meets the expandability requirement for the platform: if at any time the need for a new type of element arises, the only change that needs to be done is to create a new element in the dialect specifically for that type.

A brief description of each of the dialect element that defines the mentioned waypoint types can be found in the following sections.

3.2.2.2.1 The EnemyWayPoint element Definition

This element defines a waypoint referring to an enemy in the level. The waypoint defines the enemy's properties as well as points to the SWF file containing the graphical element of that enemy and finally points to a Flash element that defines the position where this element should be loaded. The element's structure in the dialect is as shown in Figure 3.9.

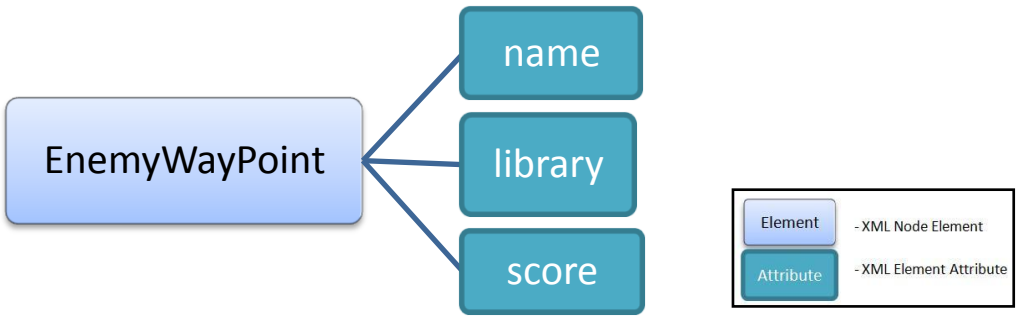


Figure 3.9: Structure of the EnemyWayPoint element

Using this method it is easy to add new enemies to a level, just by duplication the EnemyWayPoint element and changing the "name" attribute value. The only other thing to do is positioning a new waypoint in the elements SWF file with the according names and adequate positions.

3.2.2.2.2 The ItemWayPoint element Definition

This element defines the waypoints referring to the items in the level. It gives information to the engine of both the waypoint and the item properties. This element is defined by a group of attributes, as shown in Figure 3.10.

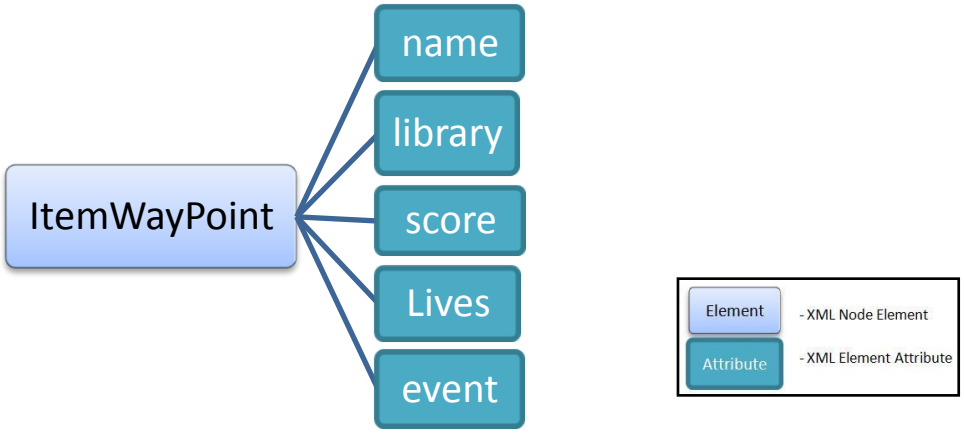


Figure 3.10: Structure of the ItemWayPoint element

This element can define a number of different items, from normal items (that give points to the player), to live boost items (that give lives to the player), and special items (that give power to the player). The fact that the attribute "event" is defined in the dialect allows for new gameplay dynamics to be implemented. At any time new events can be defined in the dialect that the engine can be programmed to handle and with this expand the possibilities of items in this game engine.

3.2.2.2.3 The ObjectWayPoint element Definition

This element defines not only the waypoint that it refers to but also the properties of the object. The game engine uses this information to create a graphical representation of the object based on the attribute values defined in the element.

In a level objects can be define as parts of the scenery (like doors or rocks), but with the difference that the player can interact with these parts. This interaction can be picking-up the object to throw at something or someone, or to destroy it to get through the obstacle (for example a door blocking the way).

The structural definition of this element in the dialect is as shown in Figure 3.11.

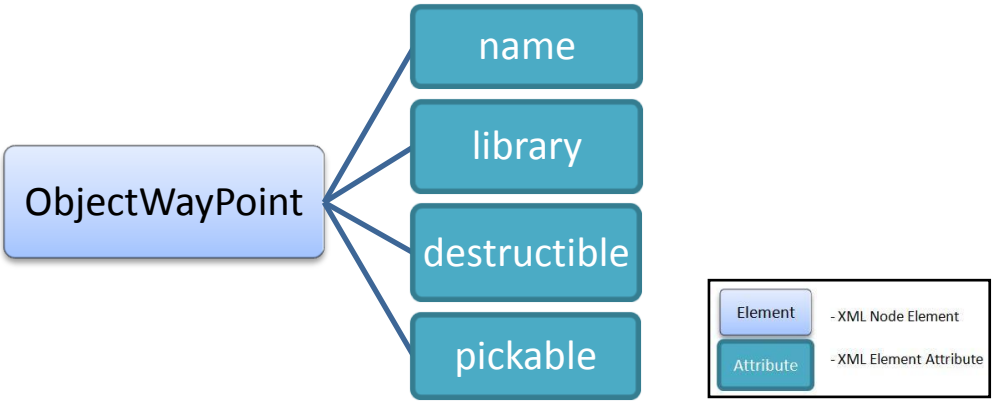


Figure 3.11: Structure of the ObjectWayPoint element

This element can be used to create puzzles in the level such as, creating some paths that are blocked by objects, and the player will have to overcome them by destroying the objects. This element can be a way to create different gameplays and contribute for a good user experience. The fact that this element is defined in this manner makes it very easy to add or remove object to a level, creating a dynamic development environment.

3.2.2.2.4 The LevelStartingPoint element Definition

The LevelStartingPoint gives the game engine information about where to place the player's character at the beginning of the level. The element can also define a new element to be placed at this point to represent the starting point (like a tree house that could represent the place here the level starts). The structure defined in the level dialect is as illustrated in Figure 3.12.

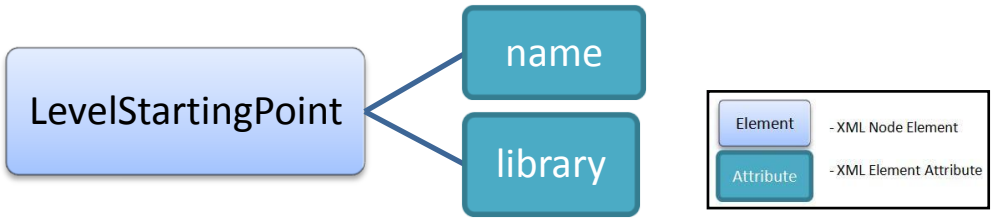


Figure 3.12: Structure of the LevelStartingPoint element

For the purpose of making the XML file easier to read, the first element in the list of waypoints has to be the LevelStartingPoint element that the level XML document can be valid.

3.2.2.2.5 The LevelEndPoint element Definition

The LevelEndPoint gives the game engine information where the level ends. This element defines an area that, if the player enters, launches an event to end the current level. This area can be visible to the player (for example a finish line, or a building) or not, simply determining its area and position. The structure of this element in the level dialect is as shown in Figure 3.13.

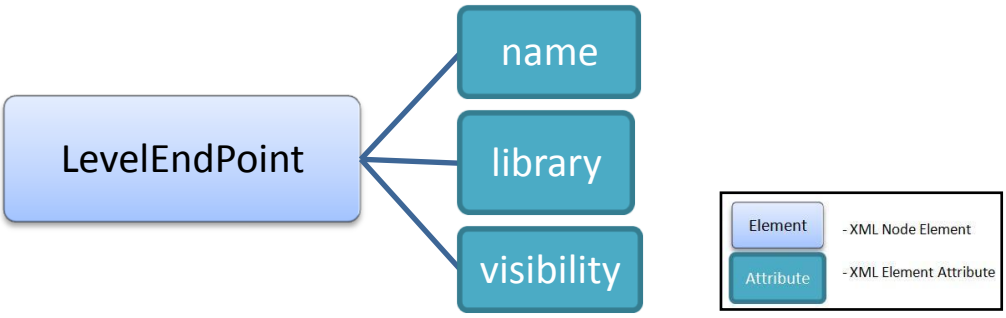


Figure 3.13: Structure of the LevelEndPoint element

This element has to be defined in the level XML file and must be the last of the Waypoints elements. This element exists with the purpose of giving more power to the developer in creating the end of the level, for example if the developer wanted the end of the level to be on top of a platform or behind an object he can do this.

3.2.2.2.6 The EnemySpawnPointWayPoint element Definition

The EnemySpawnPointWayPoint defines a point in the level that generates instances of enemies, and can have a graphical representation (for example a portal graphical element). Its structure in the dialect is as shown in Figure 3.14.

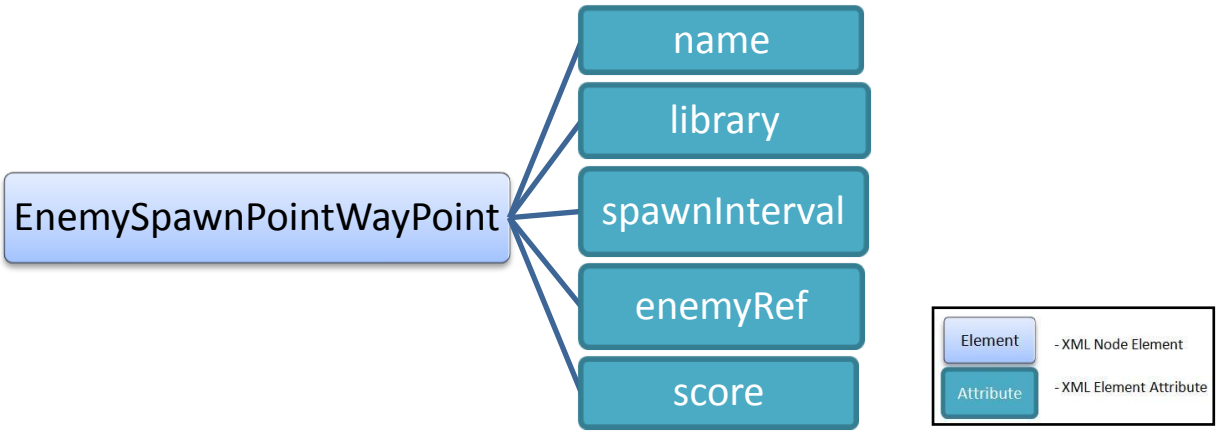


Figure 3.14: Structure of the EnemySpawnPointWayPoint element

This element gives the developer one more feature to add to a level. These spawn points can be easily added and removed or even reconfigured. The idea for creating these types of elements in the level is normally to create new challenges for the player to overcome.

3.3 The Game Engine Architecture

A game engine is composed of a group of elements that define the way the game behaves. Therefore, and according to Mönkkönen [Mön06], every game engine must have a group of features implemented, like:

Physics – In the case of a 2D platform game, the physics needed are basically gravity and collisions.

Graphic User Interface Management – A game engine has to give the developer the ability to manage the information shown on the screen.

Content Management – A game engine must have a form of content management implemented that the developer uses to load and unload content from his game.

Input Handling – Because a game has to be interactive, the game engine must implement an input handler so that the developer can use it to create his game mechanics.

All these elements are very independent and serve very specific purposes in a game. One way to design the architecture of this game engine is to treat each of the entities as an isolated model that solves a specific problem or feature.

The architecture was designed following the previously mentioned features structure (as shown in Figure 3.15).

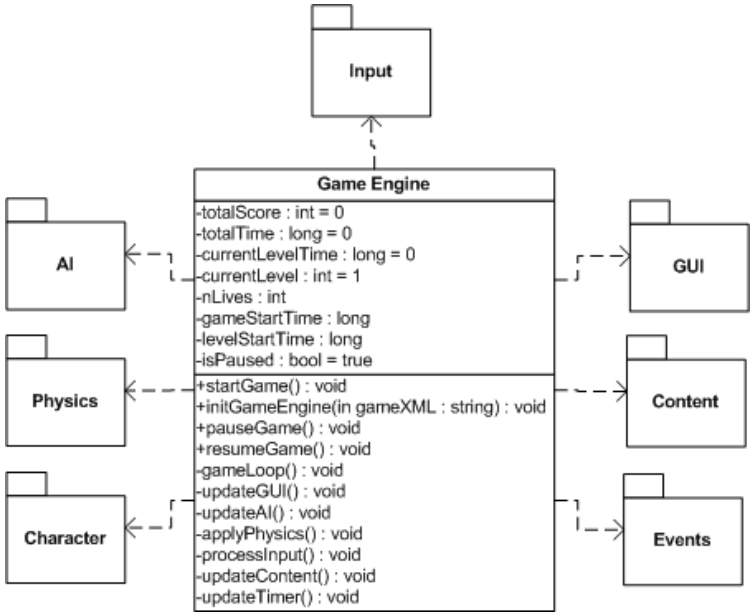


Figure 3.15: Game Engine's Class Diagram

This architecture is centred on the class GameEngine, which controls game issues. This class depends on other seven packages, each solving a different set of problems or features of a 2D platform game.

Because each of the features is isolated in each one of the packages, no connections are defined that puts them all together. To get a better performance and a structure that can easily be altered and expanded, these elements will communicate with each other using events. When an element changes, or needs something to change, he dispatches an event that all the other elements that are listening can receive and perform accordingly. This type of architecture leads to a type of programming called Event-Based Programming, which will be better explained in the following sections.

In the following section each one of the seven packages shown in Figure 3.15 will also be explained and their structure defined.

3.3.1 Event-Based Programming

Event-based programming (also known as event-driven programming) is a form of programming paradigm in which the flow of the program is not defined by the order of its code. When an event is dispatched, the program will run the code that is defined as the handler for the event (sometimes even interrupting the code that was running at the moment the event was dispatched). The other reason for the undefined program flow is the fact that the element that dispatches the event normally has no information about who is listening to that event and, because multiple elements can be listening, the program flow is unpredictable. This system is illustrated in Figure 3.16. Here, the mouse dispatches an event that an element, listening to it, catches and starts to perform actions; these actions made the element also dispatch an event that a group of other elements where listening to [Fai06].

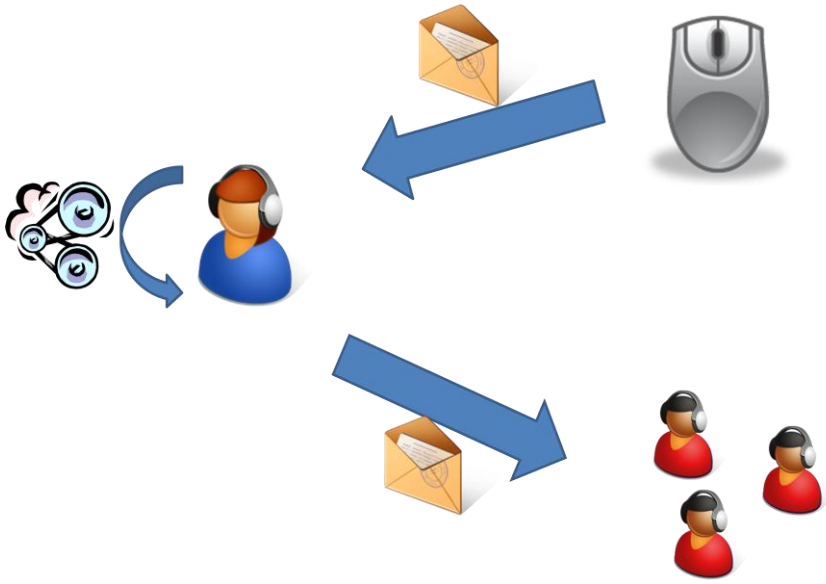
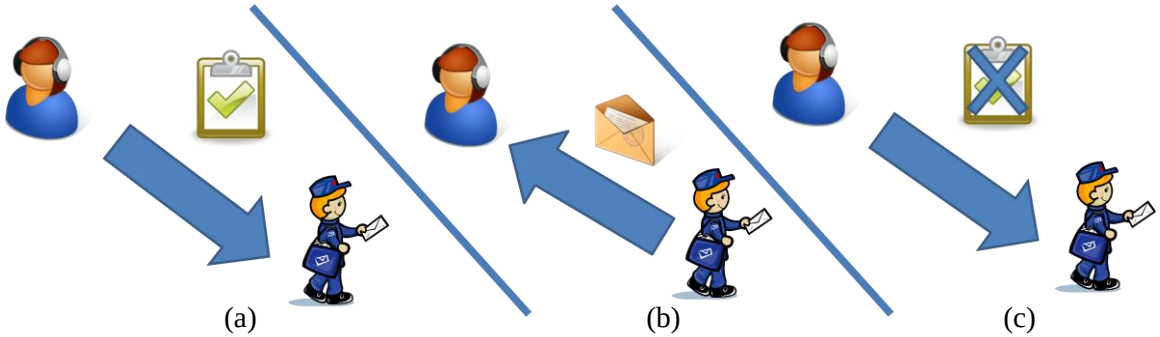


Figure 3.16: Message exchange example diagram

The advantage of this type of structure is that, from an architectural point of view, the elements are independent from each other. The element that dispatches an event does not need to know anything about the receiver and the receiver does not need to know anything about the sender; the only common point to both is the event message.

3.3.1.1 The Listener

It is important to understand the way a listener element work when developing an event-based program. The listener is responsible for informing the events manager of its intension to be notified when a given event occurs, this step being called registration (Figure 3.17 (a)). As shown in Figure 3.17, the next step occurs when an event arrives; this is when the event manager notifies the registered elements to call their handler functions (Figure 3.17 (b)). Finally, if at any time the element wishes to stop listening to the event, he has to notify the event manager to unregister him from listening to the event (Figure 3.17 (c)) [Fai06].



(a) (b) (c)

Figure 3.17: Illustration of the registration, handling and unregistration of a event listener

3.3.1.2 The Sender

The sender has also an important role in this architecture. When the sender wishes to dispatch an event, it has to inform the event manager, which will then notify every element that is registered for that event. One important issue is the fact that the sender does not know who receives the event or even if anyone receives it. The Figure 3.18 illustrates the event dispatch process from the point of view of the sender. [Fai06]

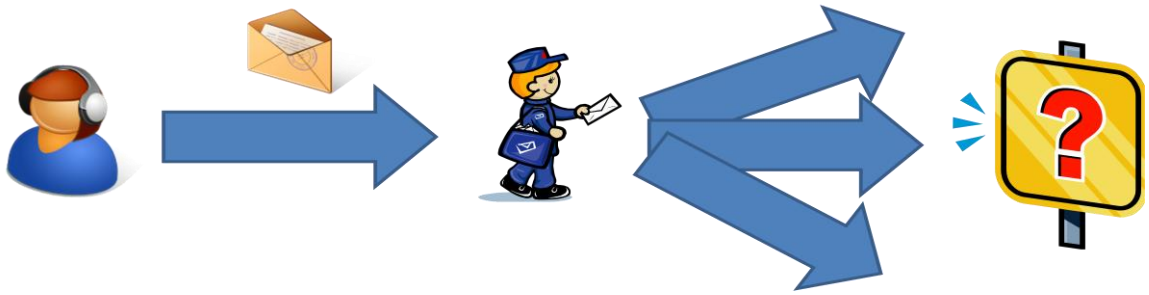


Figure 3.18: Illustration of the event dispatching from the senders point of view

3.3.2 Content Package

The Content package is the game engine's module that is responsible for managing the graphical elements of the game. Like all the other modules of the game engine, this one has its own manager, the ContentManager class.

The ContentManager class is responsible for listening to events that have graphical implications (for example when an enemy is killed). The architecture for this package is as shown in Figure 3.19.

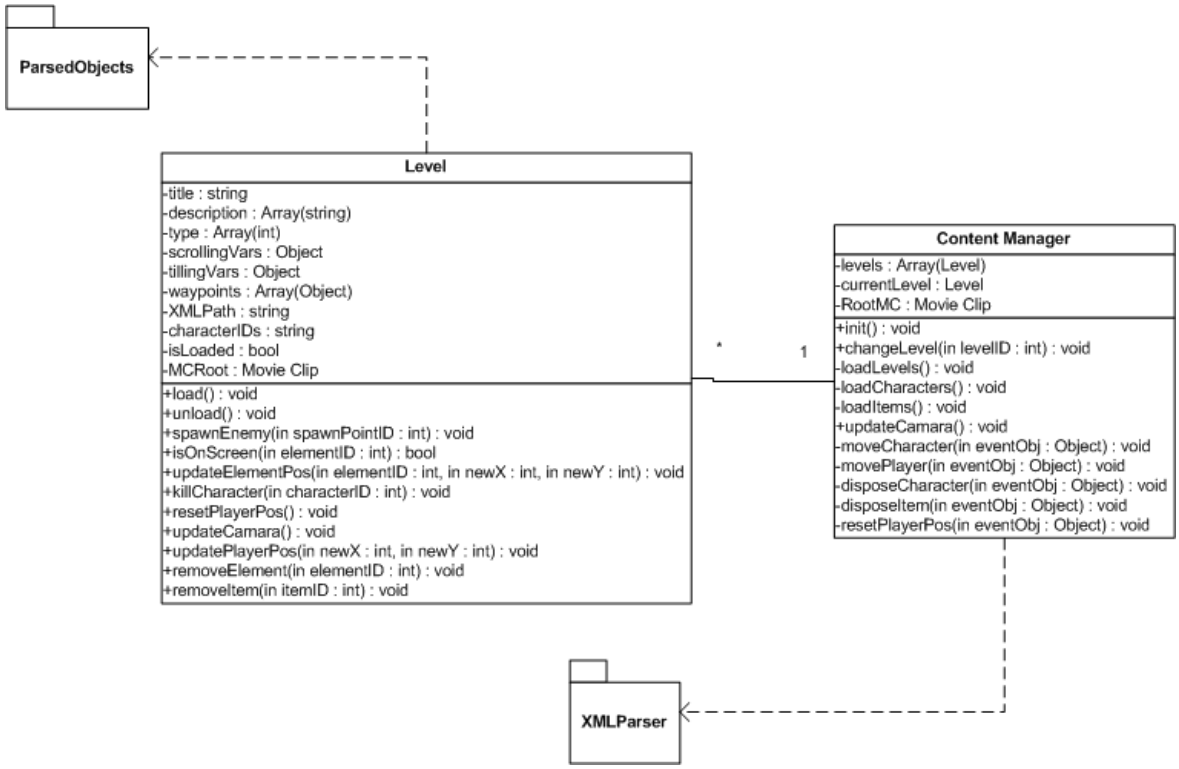


Figure 3.19: Content package's class diagram

The ContentManager class is the centre part of this architecture, being this the responsible for the connection of this package with the remaining parts of the game engine. The Content package is responsible for loading, updating and unloading graphical elements so it is also responsible for the parsing of the XML files.

The ContentManager has a list of Level elements that it manages. It uses the XMLParser package to parse XML documents and creates new instances of the Level class with the parsed data from the files. The Level class uses the ParsedObjects package for the interpretation of the parsed data (namely to process the different types of waypoints defined in section 3.2.2.2 The Waypoints Element Definition).

The fact that the ContentManager class is separated from the interpretation of the Level class makes it easier to update this module, adding new features to the level. Also the creation of the two packages to group the parser classes and the parsed objects classes facilitates the adding of new elements to the dialects.

In the following sections both the XMLParser and ParsedObject packages will be explained.

3.3.2.1 The XMLParser Package

The XML Parser Package is part of the Content package. This package holds the classes that are responsible for the parsing of the XML files that compose the game (the game XML file and all the level XML files that it refers to). The main class for this package is the XMLLoader class. The ContentManager class calls the methods of this class to load the XML files.

The package is composed by three classes and a library called XML2Object, as shown in Figure 3.20.

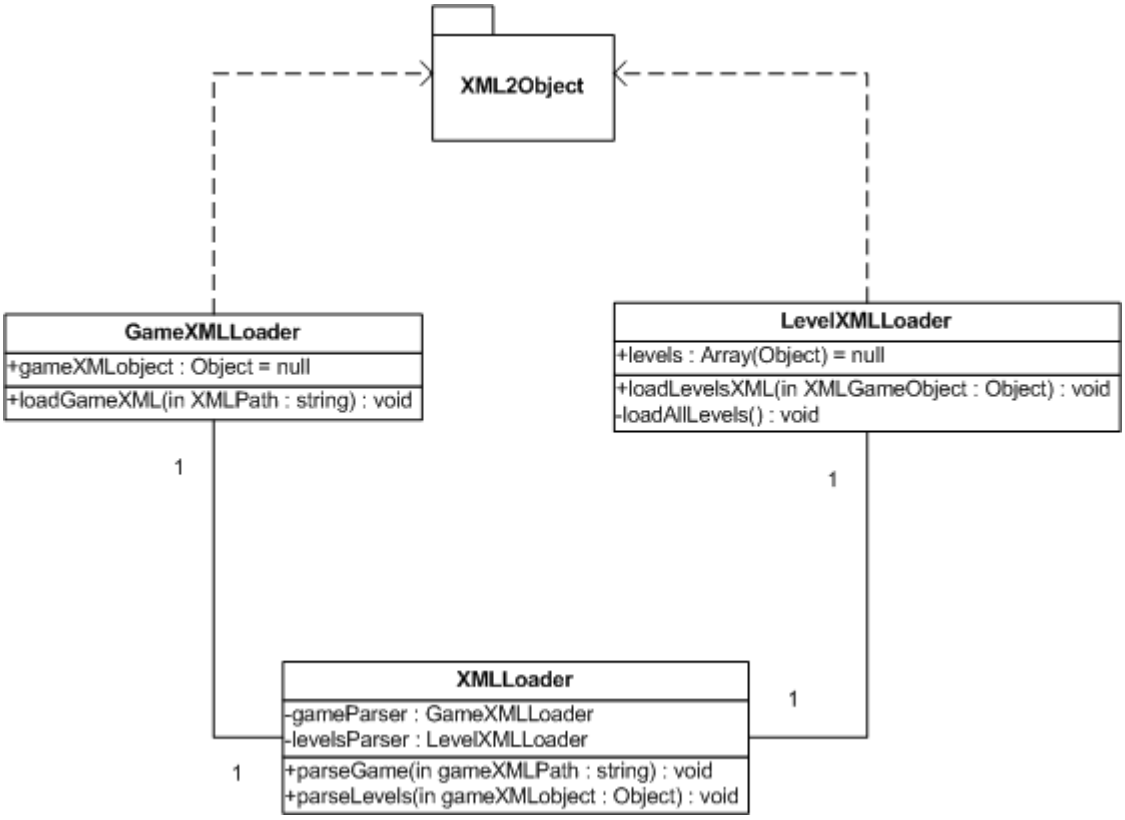


Figure 3.20: Content's XMLParser package's class diagram

The GameXMLLoader and LevelXMLLoader are the classes that are actually responsible for parsing the XML files. The reason for this architecture is that the game has two distinct dialects that can be changed at any time (as a requirement for the system), and this change as to be easily done. So all the parsing logic for the Game XML dialect is contained in the GameXMLLoader class, and for the Level XML dialect in the LevelXMLLoader class.

An important component of this architecture is the XML2Object library. This library is an open-source and free to use library that was made by Alessandro Crugnola and Phil Powell [Cru07]. It reads any XML file and transforms it into a Flash object representation. The object representation of the XML file is loaded in memory and makes it faster and easier to navigate through the XML file structure in object form. As shown in Figure 3.20 both of the file loader depend on the XML2Object library to parse the XML data.

3.3.2.2 The ParsedObjects Package

This package is mainly a container for the class representation of each of the waypoint types defined in the level XML dialect (see section 3.2.2.2 The Waypoints Element Definition). There are 6 classes defined in this package as shown in Figure 3.21.

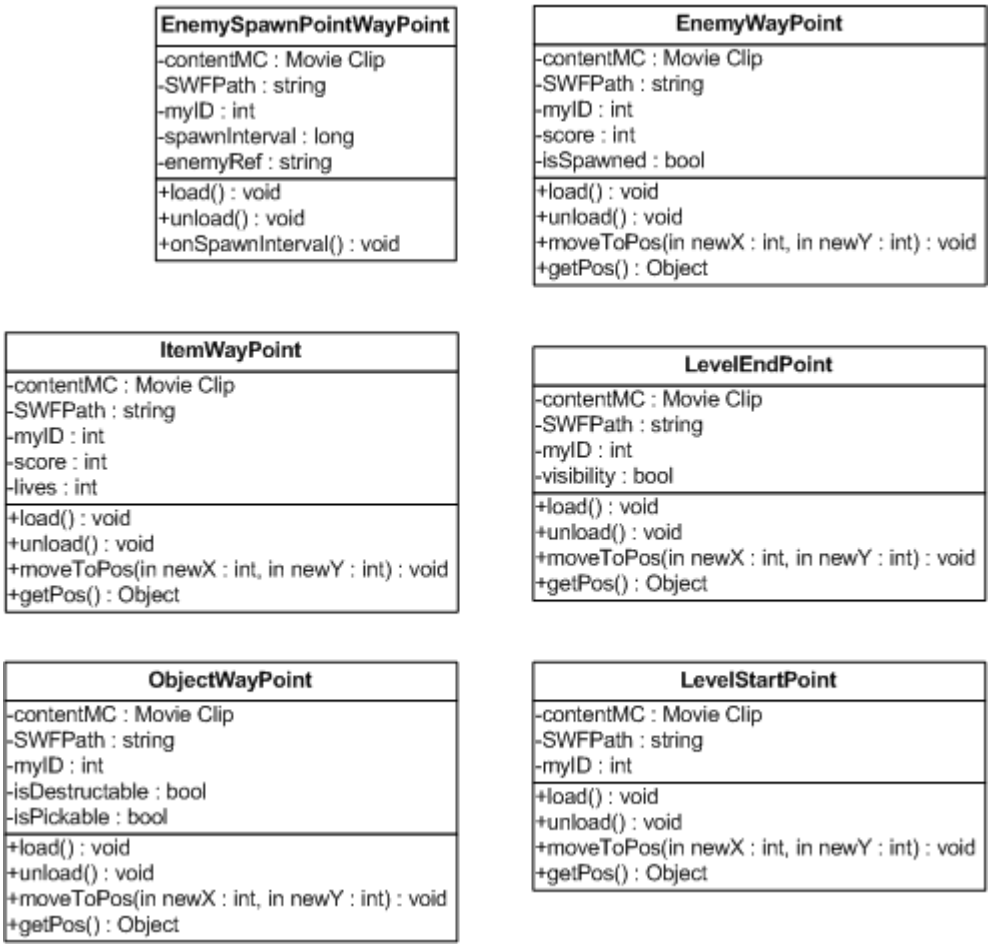


Figure 3.21: Content's ParsedObjects package's class diagram

Each of the classes contains the property values and logic of each waypoint types. Every element has methods for loading and unloading its contents. Also every element has a unique identifier that is given by the ContentManager and will be used by the game engine to unambiguously refer to the object at any time.

The fact that each waypoint type has its own class makes it easier to alter the mechanics of that type of element (adding or removing features to a type of element). Also, if at any time a new waypoint types is necessary to be added to the Level XML dialect it will only be necessary to add a new class with the elements properties and logic to this package.

3.3.3 Character Package

The Character package is responsible for the management of the player character. The CharacterManager class is the centre part of this package and is responsible for listening to the events that are launched from the Input package and process that information accordingly to what is defined in the XML file definition of each character. The architecture of this package is very simple, as shown in Figure 3.22.

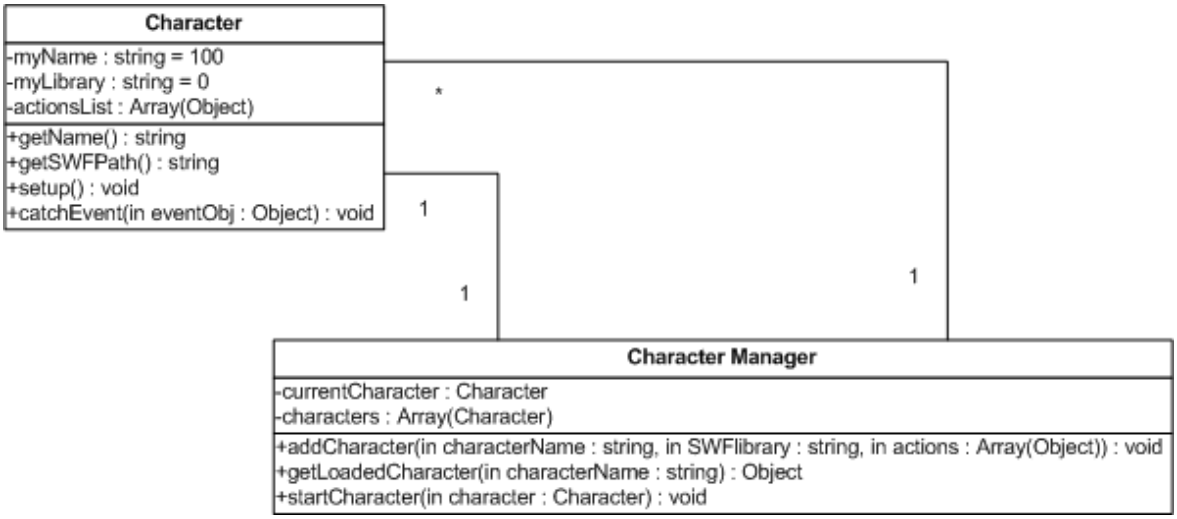


Figure 3.22: Character package's class diagram

This package is composed by the CharacterManager and the Character classes. The CharacterManager class is responsible for maintaining and managing the player characters. The game engine uses this class defined methods to add new instances of characters (after the interpretation of the game XML file) and to change the player character when needed. The CharacterManager class maintains a list of all loaded characters as a list of instances of the Character class and maintains a copy of the instance for the current character that is being used in the game for easier referencing and usage.

The Character class contains the general logic that can be used in a player character, more specifically the actions that can be done by a character.

The fact that the entire player's logic is implemented in the Character class makes it very easy for a developer to add, alter or remove features of characters; for example, if a new action is added to the XML dialect, the developer only has to implement the action in the Character class's "catchEvent" method.

3.3.4 Artificial Intelligence Package

This package is responsible for the management of all the artificial intelligence associated with the game's enemies. Because of the unique nature that each level and game enemy can have, this package architecture was designed to allow the developer to create his own strategy for each enemy, and in a fast and easy way integrate that strategy with the game engine. This architecture is as shown in Figure 3.23.

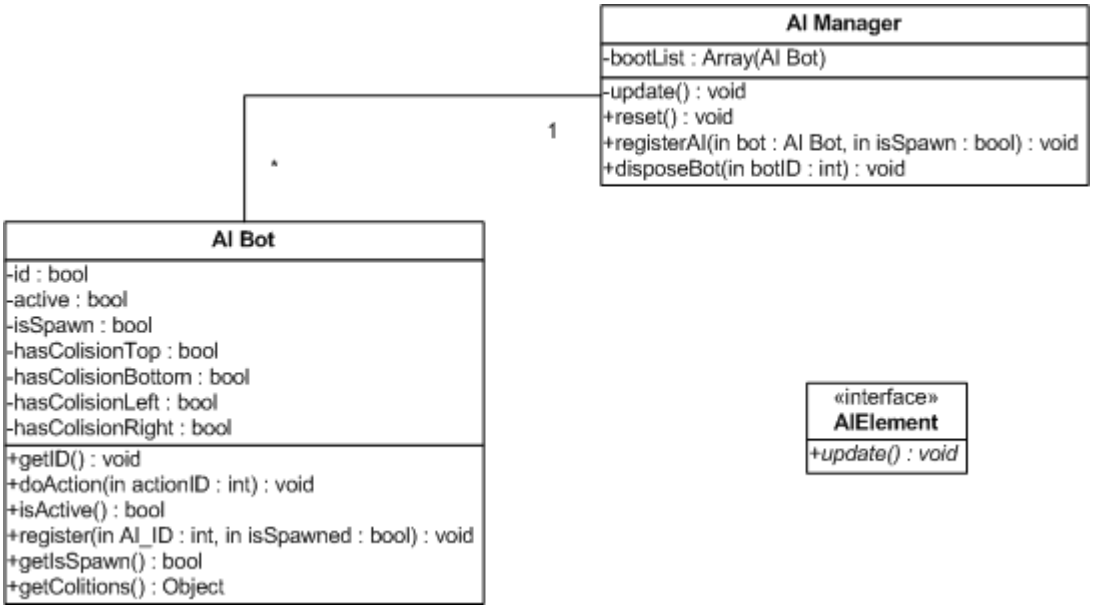


Figure 3.23: Artificial Intelligence Package's class diagram

This package is composed by two regular classes and an interface class. The AIManager class is responsible for the management of all the artificial intelligence (AI) for the game. The game engine calls the "update" method every time he decided that the AI can perform actions (this is called the logic or update loop). Because the AIManager only manages the instances of the enemy's AI classes, the only thing he does is to visit every AI element that is available to perform an action (AI elements can be deactivated for various reasons, namely if they are outside of the current viewport they cannot do any actions) and allows that element to perform the action.

As said before, each game enemy as his unique characteristics and AI logic. To allow for this game engine to implement any type of game the design architecture had to be dynamic enough to allow the developer to create his own strategy for each of the enemies. This is accomplished with the use of both the AIBot class and the AIElement interface class. The developer can create his own, costume made, AI logic classes as long as those classes implement the AIElement interface and extend the AIBot class. The AIElement interface will guarantee that the class has an update method defined (this method will be the one that the AIManager will call when updating all the AI elements). The fact that the class extends the AIBot class not only allows the developer access to information about the enemy (for example if the enemy is colliding with any wall), but also will guarantee that the registration with the AIManager is made (this is because a call to register the class is made on the AIBot class's constructor).

The perceptions that the AI logic can use are all defined in the AIBot class. If at any time the developer feels the need to have additional information about the world additional perceptions can be added to the class that give that information. Also the actions that an AIBot element can have are defined in his class and if a developer at any time feels the need for additional actuators they can easily be implemented.

This dynamic structure of the AI module of the game engine creates a means the developer needs to implement any game strategy he wants with little effort.

3.3.5 Input Package

The Input package is responsible for listening for the player's keyboard events.

This package is only composed by one class, the InputManager. The InputManager is responsible for listening to keyboard events and to dispatch the events that are defined in the Game XML file.

This manager has a addKeyListener method that allows the Game Engine to register the key events defined in the XML file. He also has two methods to check keys, the checkKeys and the checkKeysSpashScreens methods. The first is used to check if any of the defined key events occurs. The second method is used instead of the normal key checking when the game is not running (for example in the intro screen or the winner screen).

Finally this manager has a reset method that the Game Engine calls when a new level or character is loaded so that the new keys can be added.

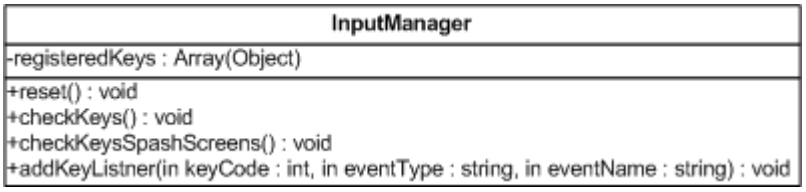


Figure 3.24: Input package's class diagram

3.3.6 Physics Package

The Physics package is a complex part of the game engine. This module is responsible for all the physics calculations in the game.

The architecture of this package, as shown in Figure 3.25, is centred on the PhysicsManager class. This class is responsible for all of the physics calculations and logic. The game engine calls the "update" method of the manager that then does all the necessary calculations and generates the necessary events (for example if a collision between the player and an enemy is detected).

This said there are mainly three steps when applying physics to the game, these are:

Applying Gravity – This is applied only to enemies and the player instances.

Calculating Hit-Map Collisions – This is applied only to enemies and the player instances.

Calculating Collisions with the Player – This is calculated between the player and the enemies (distinguishing when the player collides on top of the enemy or not), between the player and the items elements and finally between the player and the end of the level area.

There are mainly four entities that are important for this physics engine that are: the player, item, enemy character and the end of the level area. As shown in Figure 3.25, each of these elements is defined as a class linked to the PhysicsManager. The PhysicsManager therefore contains a list of all the enemy characters (as instances of the Character Class), a list of all the levels items (as an instance of the Item class), an instance of the player element (as defined in the Player Class) and an instance of the end of the level area (as defined in the EndLevelArea).

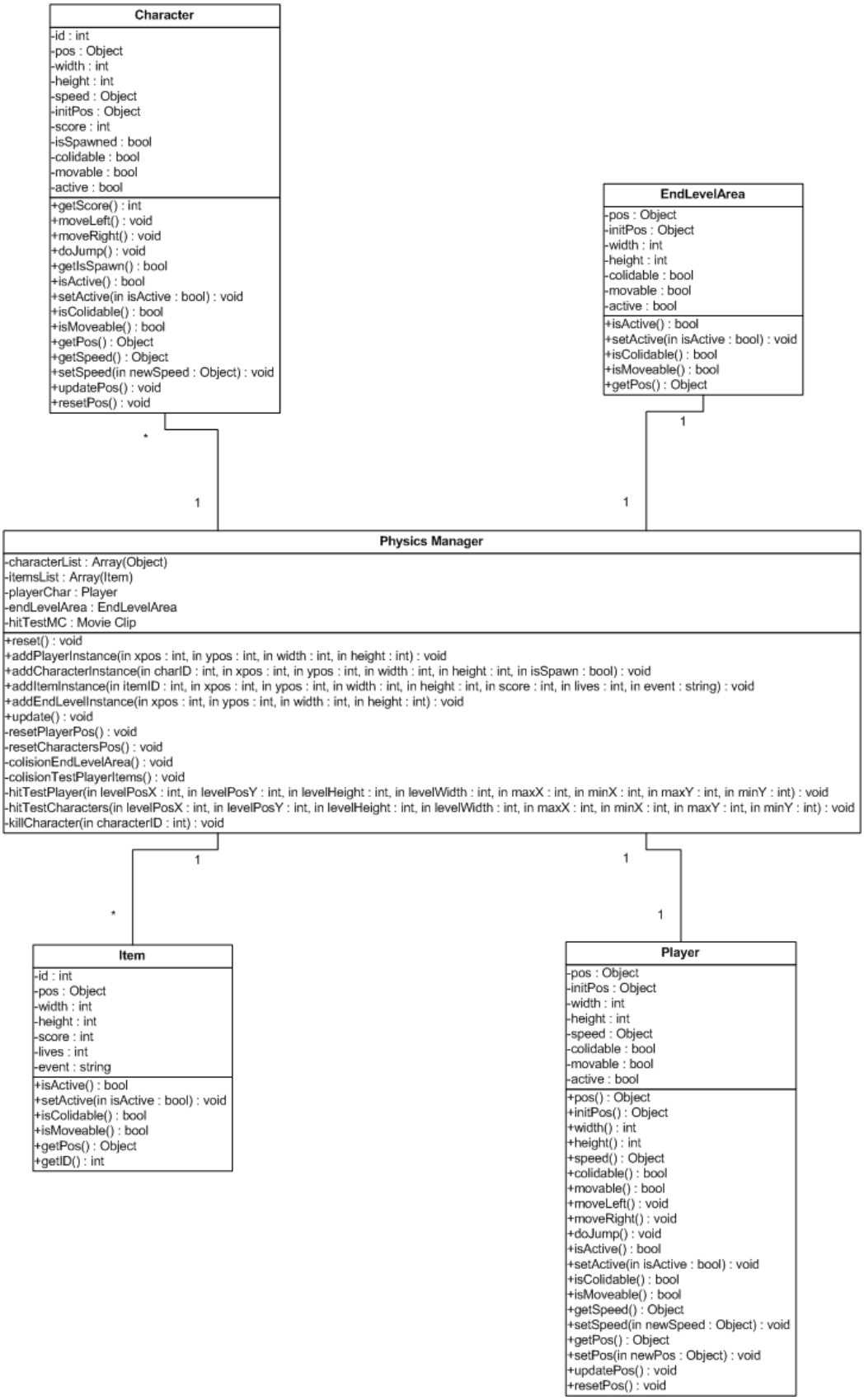


Figure 3.25: Physics package's class diagram

This architecture allows the developer to easily add new elements to the PhysicsManager, creating new game mechanics.

3.3.7 Graphic User Interface Package

The Graphic User Interface (GUI) package architecture is, in some aspects, very similar to the Artificial Intelligence one (section 3.3.4 Artificial Intelligence Package).

To allow the designer and the developer to easily create their own, costume made GUI, a very specific architecture was created. A designer should be able chose how to create the interface, for example create an interface where the number of lives are shown as a number on the screen, or as a group of graphical elements. This led to the definition of the architecture shown in Figure 3.26.

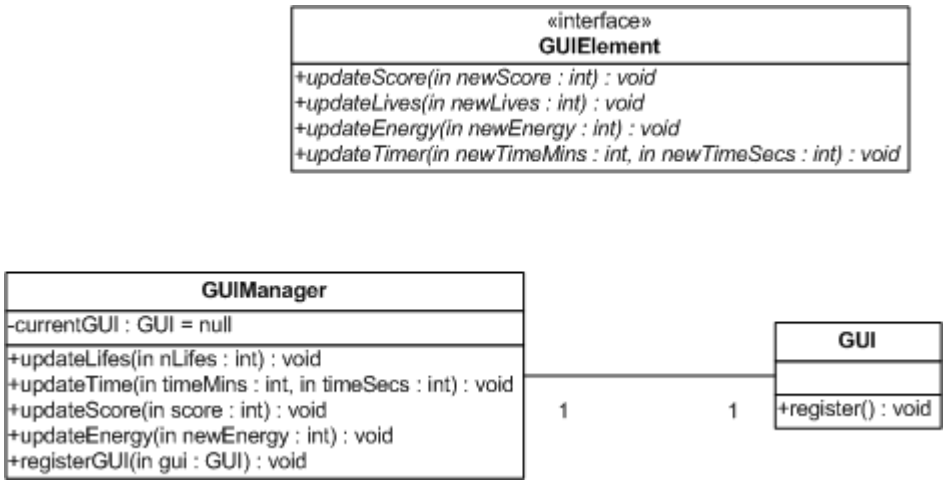


Figure 3.26: Graphic User Interface package's class diagram

Using this package, the developer can implement his own class to manage his GUI, as long has that class extends the GUI class and implement the GUIElement interface. The fact that the developer implements the GUIElement interface guarantees that is class has the necessary methods for the GUIManager to update the GUI data when necessary. Also by extending the GUI class the developer guarantees that the class is registered with the GUIManager (this registration is called by the GUI class constructor).

The largest advantage of this dynamic structure for the implementation of custom graphical user interfaces is that it gives the developers the power to create unique user experiences in each game he creates.

3.3.8 Events Package

The Events package is the part of the game engine that is responsible for managing the event messages that are exchanged between different parts. This package is composed of a single class called EventsManager as shown in Figure 3.27.

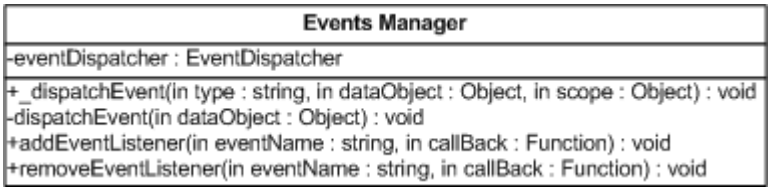


Figure 3.27: Events package's class diagram

The EventsManager class is responsible for dispatching event messages to other classes. This manager has an "addEventListener" method that any class can call to register a listener for a given event (the event is defined by its event name). When any class calls the "_dispatchEvent" method, the manager notifies every class that had registered a listener for that event.

This package architecture allows for an event-based programming architecture to be easily implemented. Also the fact that the events are defined by their names allows for the same names given in the XML files to be used in the game engine, as well as makes it very easy to add new events from every part of the game engine.

3.4 Summary

FullSix creates multimedia campaigns for a number of different clients. These campaigns sometimes include developing Web-based games. What FullSix realized is that these games are a significant part of the development process, and also that the pieces of each game are not reusable. What FullSix needed was a tool that helps developers to create these games easier and faster. Because there are a lot of game types, each with its own characteristics, FullSix decided to focus only on the development of platform games.

This work consists on the development of a tool to solve the problem proposed by FullSix. This tool is composed by two parts. The first part is a dialect that the developer can use to describe all the elements of the game and its mechanics. The second part is the development of a game engine specifically designed for 2D platform games, and able to understand the XML dialect.

The dialect used to define the elements and mechanics of the game consists of two XML dialects, one to define the game, and another to define its levels. On the game XML dialect the developer defines the global aspects of the game, like input keys and player characters. On the level XML file the developer defines the elements and mechanics of the level, like the type of elements or the level scrolling type.

The game engine is composed by a set of packages, each responsible for a different feature of the game. This architecture has the advantage of allowing the developer to easily add new features to the game engine at any time. The game engine also is designed to be adapted to both XML dialects so that they can work together.

Chapter 4

Project Implementation

In this chapter the key aspects of the project's implementation are described. The first section describes the planning made to create all the parts of the projects, from architecture to tests. Next some aspects like important libraries and programming patterns that were used are described. Finally the last section describes the way the main Flash project, which supports and runs the game engine, was implemented.

4.1 Project's Planning

This project started on the 18th of February 2008. From that day the project's total available time was 20 weeks. A planning was made that divided this time into four parts:

Research – 4 weeks
Requirements and Architecture definition – 6 weeks
Implementation and testing – 8 weeks
Project completion – 2 weeks.

Another important part of the planning is the writing of the projects report and maintaining and updating the project's Web page. This part has the duration of all the 20 weeks of the project and is planned to occur simultaneously with all the other defined parts of the project.

Each of these parts are composed by a number of tasks, these tasks are defined, for each of the planning parts, in the next sections. A complete description of the project's planning in the form of a Gant diagram can be found in appendix E: Project's Planning – Gant Diagram

4.1.1 Project's Research

This part consisted on researching books and technologies to better understand the problem and the options available to solve it.

This part is composed mainly by three tasks:

- Researching about Web-development platforms;
- Researching about existing frameworks for game development;
- Researching about game design and gameplay.

These tasks have no type of dependency between them so they can all occur simultaneously.

4.1.2 Project's Requirements and Architecture Definition

This part of the project occurs after the research is completed. This dependency on the research is due to the fact that the problem's options and possibilities have to be very well defined before trying to define its requirements and architecture.

The first task is to acquire a very good and clear definition of the requirements for the system. The next step is to build the system's architecture based on that requirements definition.

This part is then defined by the following tasks:

- Definition of Project's Requirements;
- Definition of Project's Architecture;
- Definition of Proof of Concept;
- Prototype Implementation.

After the architecture is defined the project needs for a way that it can be demonstrated and its potential tested. This is why the proof of concept is so important. For this task it was developed a game with the goal of proving that all the requirements are meet.

Finally a small prototype has to be implemented to correct any large scale architecture problems and to detect if there are any problems in the requirements interpretation.

4.1.3 Project's Implementation and Testing

The project's implementation part depends on the finalization of the requirements and architecture definition. This part can only begin after a good definition of what to implement, which features and how to set it all up.

This part is composed by the following tasks:

- Test definition
- Project implementation
- Usability test execution
- Performance text execution

The first task to be done it the test definition, all the other tasks can only start after all tests are defined. Next is the implementation task, and when the system is implemented the tests must be executed.

4.1.4 Project Completion

After the tests are executed some problems may be detected and changes have to be made to correct these problems. Also it is now time to create the proof of concept game so that the engine can be seen in action and all the requirements validated.

4.2 Flash's Event Dispatcher Library

One of the main parts of this platform's development is the Event Manager. This manager is one of the key elements of this system's architecture because a large number of element and manager register and dispatch events to each other.

Mainly there are three methods that the Event Manager has to provide the other classes with; these are the methods for registering, unregistering and dispatching events (event-based programming is explained in section 3.3.1).

The Event Dispatch Library that is included in ActionScript 2.0 is a great way to implement this architecture. This library already includes functions to register, unregister and dispatch events.

The implementation of the Event Manager consisted on creating the three methods mentioned in the previously, but these functions are already available in the Event Dispatch library. To use the functions the Event Manager Class has to import the "mx.events.EventDispatch" library. Next the class has to initialize the Event Dispatch library once before using it; this step is best done in the manager class's constructor method.

The EventDispatch's initialization method is a static method, makes it possible to call this method from using the class's name, like this:

EventDispatcher .initialize(this);

This method receives a parameter that is a class instance that defines the scoop of the library. This call will set the EventDispatcher's scoop to the Event Manager class so that the Event Manager can call all the methods of the EventDispatcher library.

The call to the initialize method installs the library methods addEventListener, removeEventListener and dispatchEvent. So that the code compiles without a problem these methods should be defined in the class but should be empty. The code for these methods is like this:

function dispatchEvent() {};

function addEventListener() {};

function removeEventListener() {};

Although at start this event-based architecture could seem a little hard to implement the EventDispatcher library proven to be a great help in simplifying the development of this key feature.

4.3 The Singleton Pattern

The fact that the system is all divided into independent packages made it very hard for each of the managers to communicate directly with each other. Therefore most of the system logic was implemented using event-based programming.

This solution solved most of the problem but still a problem existed, each manager had to be instantiated and that instance had to be made only once because there could be only one of each manager at any given time.

This description fits exactly the description of the Singleton design pattern in software engineering. This design patter is used to restrict the number of instances of a given class to one.

The Singleton pattern in implemented by creating a method in the desired singleton class that return an instance of that class, this instance should be created (and it should be saved for later use) if no other instance exists or the method should return the already existing instance. To guarantee that there is no way for other instances to be created the class's constructor should be set to be protected instead of public. The code for a singleton class should look something like this: [San07]

```
class MySingletonClass{
    private static var _instance: MySingletonClass = null;
    protected function MySingletonClass (){}
    public static function getInstance():MySingletonClass {
            if(_instance == null) {
                    _instance = new MySingletonClass ();
            }
            return _instance;
    }
}
```

This pattern is used in every manager, guaranteeing that only one instance of each exists at any time in the game engine.

4.4 The Flash Project

The game engine needs a flash project to run on. This project's architecture is important to guarantee that the game engine's behaviour is has expected.

This said one of the key aspects of the projects architecture is that the game elements need to be placed in a certain layer order so that none of them overlap elements that they shouldn't (for example the game elements overlapping the user interface or the background over lapping the game elements making it impossible for the player to play the game).

The Flash project should have a group of layer where the containers for the elements are defined. These layers should have the order represented in Figure 4.1.

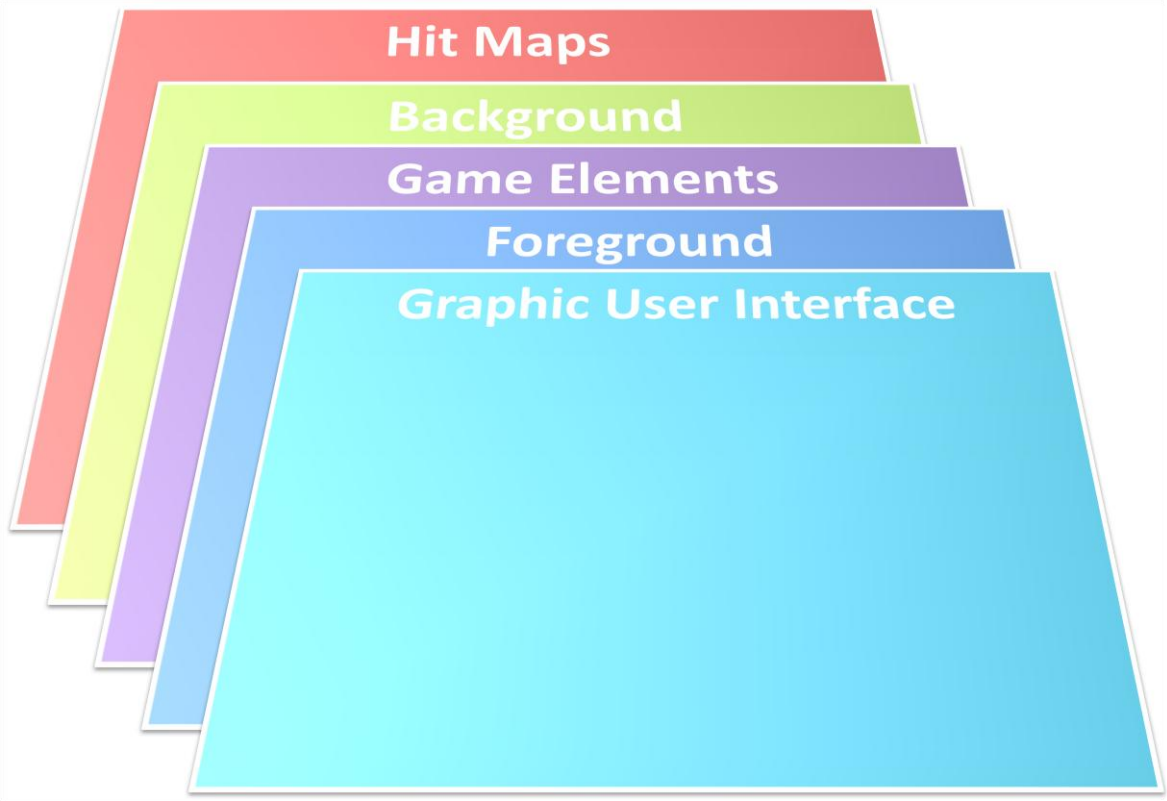


Figure 4.1: Illustration of the Layer Distribution of the Game Engine's Flash Project

The top layer should contain the Graphic User Interface (GUI) container, bellow this layer should be the foreground layer and bellow this one the elements layer. This guarantees that the elements are never on top of neither the foreground nor the GUI. Bellow the elements layer is the background layer guaranteeing that the game elements are always on top of the background and there for visible to the user. Finally the level's hit-maps should be on the bottom layer. Although the game engine render the hit-map elements invisible when they load, the fact that they are on the bottom layer guarantees that they will disturb the normal execution of the game (even if by any chance some error occurs in the code that prevents the object form becoming invisible).

Another important aspect of the Flash project implementation is the frames that exist in the project's timeline. The timeline is composed by a group of frames, on for each of the games screens and one more that the game engine for playing the game. This idea was inspired by the definition of scenes in the Popfly Game Creator tool explained in section 2.3.1. The idea is that the designer can create the splash screens for the levels into, and for the games winning and losing screens. Each screen is a frame in the root of the project and the game engine will scroll back and forward depending on the games conditions. With this implementation designer can easily customize the each game's screens to match the scenario and event the Web page's main theme.

4.5 XML Game Definition Prototype

During the development of the game engine a prototype of a game was created. The development of this prototype mainly consists on creating two XML files, one defining the game and another defining a level of the game, following the architecture defined in section 3.2.

The goal of creating a prototype during the game engine implementation was, not only to have a small demonstration of the game engine's potential, but also to test the architecture that was created.

In the next sections, the two files are described in more detail. The first section describes the game XML file definition and the second section describes the prototype's level for that game. A complete example of the game XML files can be found in appendix F.

4.5.1 Game XML Prototype

The main configurations of the prototype game are defined in the Game XML file, as shown in Figure 4.2.

```
<?xml version="1.0"?>
<Game name="Jogo de Teste">
  <Config>
    <Keys>
      <Key keycode="37">
        <Event name="moveLeft" type="isKeyDown"/>
      </Key>
      ...
    </Keys>
    <Startup>
      <Lives amout="3"/>
    </Startup>
  </Config>
  <Levels>
    <Level definition="level1.xml" character="hero1"/>
  </Levels>
  <Characters>
    <Character name="hero1" library="Libs\player.swf">
      <Actions>
        <Action event="moveLeft" frame="3" action="moveLeft"/>
        <Action event="moveRight" frame="4" action="moveRight"/>
        <Action event="jump" frame="5" action="jump"/>
      </Actions>
    </Character>
  </Characters>
</Game>
```

Figure 4.2: Game XML file example

First in the definition of the game XML file are the configurations; in this game there are three keys defined and one start-up variable. Starting with the game's keys, they are:

- Jump key (key code: 38, key event type: onKeyDown, throws event: jump)
- Move Left Key (key code: 37, key event type: isKeyDown, throws event: moveLeft)
- Move Right Key (key code: 39, key event type: isKeyDown, throws event: moveRight)

These keys throw events that are caught by the characters that are defined on the Characters element. The second part of the configurations definition is the start-up variables and, in this game, there is only one variable defined, the number of Lives the player has when the game starts. This variable is defined in the lives element and its value is three in this case.

Following the definition of the game configurations is the definition of the levels. This prototype is composed by a single level that is defined in the "level1.xml" file. The content of this file is explained in the next section.

Finally the character that the player uses in this prototype game is defined in the Characters element. The character is named "hero1" and his graphical representation can be found in "Libs\player.swf" file. This character has a list of actions that are associated with key events defined previously. Each action defines the behaviour of the character if a given event is thrown and the frame number that the game engine should play while the character is doing this given action.

4.5.2 Level XML Prototype

This game is composed by a single level; its main configurations are defined in the Level XML file, as shown in Figure 4.3.

```
<?xml version="1.0"?>
<Level title="Level 1">
  <Description>
    This is the first level of the prototype.
  </Description>
  <LevelScrolling levelElementsRef="Libs\elements1.swf" direction="horizontal"
lenght="3000" navigation="right" scrollingSpeed="2"  backGroundName="background"
backGroundRefName="Libs\background1.swf" foreGroundName="foreground"
foreGroundRefName="Libs\foreground1.swf" hitMapRefName="Libs\hitmap1.swf"/>
  <GUI name="gui" library="Libs\gui.swf"/>
  <WayPoints>
    <LevelStartingPoint name="start"/>
    <EnemyWayPoint name="enemy1" library="Libs\enemy1.swf" score="100"/>
    ...
    <ItemWayPoint name="item1" library="Libs\item1.swf" score="100"/>
    ...
    <EnemySpawnPointWayPoint name="spawn1" enemyRef="Libs\enemy1.swf"
library="Libs\portal.swf" spawnInterval="5000"/>
    <LevelEndPoint name="end" library="Libs\endPoint.swf" visibility="true"/>
  </WayPoints>
</Level>
```

Figure 4.3: Level XML file example

The first thing to define in a level is the type of scrolling that the game engine should apply. In the case of this level the scrolling should be done normally, which means that there is no tilling. This level mechanic is defined in the ScrollingType element. This element also defines that the scrolling should be done automatically at a rate of two pixels per second and with the direction set to right. The other aspects that are defined in this element are related to the files that contain the hit-map, element map, background and foreground for the level.

Next in the file definition is the GUI element. This element defines the GUI that should be used for this level and, in this game, the GUI can be found in the "Libs\gui.swf" file. The game engine should load the SWF file to the element defined in the name attribute of the GUI element.

Finally the XML file defines a list of waypoint that the game engine should interpret to build the level. The list of waypoints is composed by the start and end waypoints, a group of enemy waypoints, a group of item waypoints and two spawn points. The list starts with the level start

waypoint that defines where the player should be placed at the beginning of the level. Next there is a series of thirteen enemy waypoints that define where to place each enemy at the beginning of the level, each element also defines which SWF file has the graphical representation of the enemy and the points that the player should receive when he defeats it. Next, the definition of the level items can be found. In this level there are nine items defined, each defining the SWF file containing the graphical representation of the item and the points that each gives the player when picked up. The next defined elements are the two spawn points of this level. These two waypoints define the position of each spawn point, the enemy that should be created and its spawn interval. Finally the last waypoint is the end of the level. This waypoint defines the area that should be used to detect that the player has reached the end of the level, and the graphical element that should be shown (in this case the visibility of the end of the level waypoint is set to true, so the element will be shown).

4.6 Summary

To start with the implementation of the project, first it was necessary to plan how to do it. The project planning was divided in four activities:

- Research
- Requirements and Architecture Definition
- Implementation and Testing
- Completion

The project had a total development time of 20 weeks, which was divided between these activities. Among them the implementation took the most of the project's time, a total of eight weeks.

The game engine is composed by a group of packages, each responsible for one of its components. The proposed solution to put it all together was to implement an event-based architecture. Flash has a very useful library called Event Dispatcher. This library already has implemented the methods needed to build an event manager, like the one defined in this project's architecture.

Another important aspect of the architecture is that each package has a manager class. When the game engine initializes it has to create an instance of each manager and has to guarantee that no other instance is created, so that there are no unexpected errors. There is a design pattern called Singleton that solves this problem. Implementing this pattern in each manager will guarantee that only one instance can be created.

Next the game engine has to have a Flash project that compiles the code and has some pre defined elements. Because the game has different layers of view they have to be defined in this Flash project. It is important to understand this mechanic so that at any time developers can change the layer were an element is positioned. The game engine divides the elements into five layers from GUI, to background, and hit maps.

Finally during the implementation of the game engine a prototype game was created, composed by one level. The creation of this prototype consisted mainly on creating a XML file for the game and another one for its level.

Chapter 5

Tests and Results

In this chapter the tests that were made to the developed system will be described. The purpose of these tests is not only to detect some performance issues but also to determine usability problems from the system user's perspective. The users of the Flash Platform Games Engine are de developers and the designer at FullSix, they will be the ones using this system in their projects and they are the best people to test and give feedback about this system.

This said there will be three groups of test to be preformed, first the usability test with developers, testing the way a developer sees the platform and getting some feedback about what is good and what could be better in the system. Next the usability tests with designers, where designers have a chance to contribute with their unique perspective and opinions. Finally the system will undertake some performance tests that are required to assure the users that the system is robust.

To evaluate the system's usability from the developers and designers' point of view, a group of usability tests scenarios were prepared so that the users could interact with the system. At the end of the test sessions each user was invited to fill out questioner with the purpose of measuring the user's satisfaction with the system's usability. This questionnaire is based on the System Usability Scale defined by John Brooke.

In the next sections all of the tests will be described and the result obtained will be presented and analysed. Also in the next section the System Usability Scale will be explained.

5.1 System Usability Scale

The System Usability Scale (SUS) is a ten-item psychometric Likert scale intended to measure a user's satisfaction about a system's usability. This particular scale was created by John Brooke and has been widely used in the evaluation of different systems.

This scale is composed by a group of 10 questions, those questions are: [Bro]

1- I think that I would like to use this system frequently
2- I found the system unnecessarily complex
3- I thought the system was easy to use
4- I think that I would need the support of a technical person to be able to use this system
5- I found the various functions in this system were well integrated
6- I thought there was too much inconsistency in this system
7- I would imagine that most people would learn to use this system very quickly
8- I found the system very cumbersome to use
9- I felt very confident using the system
10- I needed to learn a lot of things before I could get going with this system

So to each of the ten questions the user will give a value (from 1 to 5) according to a Likert scale with the following values: [Wik081]

1 – Strongly disagree
2 – Disagree
3 – Neither agree nor disagree
4 – Agree
5 – Strongly agree

Next comes the evaluation part of this method. To evaluate the user's satisfaction with the system's usability some calculations have to be made.

Because some of the questions are positive statements and others negative ones, and the same scale is used to measure both, the first step is to normalize the values given to each question.

This said the answers to questions 1, 3, 5, 7 and 9 (these are the positive statements) will have the value that the user given minus one (transforming the scale from 1 to 5 into 0 to 4). Next the answers to questions 2, 4, 6, 8, 10 (these are the negative statements) will be calculated by 5 minus the value given by the user. The only thing left to do is to add all the calculated values and then multiply the total by 2.5, converting the final value into a scale from 0 to 100. [Bro]

5.2 Tests with Developers

One of the main users of this game engine will be the developers at FullSix. For this reason it is required that they have a chance to evaluate and give feedback about the engine. This feedback is very important not only improve the game engine but also to help in the integration of this tool in the company's workflow.

The following sections will describe the tests that were made as well as the obtained results and a small analysis of these results.

5.2.1 Test Description

The target users for these tests were the developers at FullSix, more specifically the multimedia developers that produce the RIA in Flash. These tests' goal is to get feedback on the usability of the Platform Games Engine as a development platform in the developer's point of view.

To do this, two scenarios were created, each composed by a group of tasks that the users were asked to do. The tasks were designed to challenge the user to use different parts of the game engine

and get a look and feel of all the development process using this platform. The objective of creating scenarios is to simulate, as much as possible, a real working environment were the user is trying to execute a project just like in real working conditions.

The created scenarios were:

Scenario 1 – Creating a new game
Scenario 2 – Editing an already created game

The conditions for the execution of these tests are the same for all the users. Initially the test facilitator (the person that is giving the user the task he has to do and that observes the user and takes notes about its comments) asks the user a group of questions about himself, next he explains the scenario to the user and gives him the first task. When the user finishes (or gives up) the task the user will give him the next task and so on until the end of the test. At the end of the test the user will have a questionnaire based on the System Usability Scale by John Brook.

The next sections will describe each of the scenarios and the tasks that compose them.

5.2.1.1 Scenario 1 – Creating a new game

In this scenario the user in invited to image the following situation:

> "You're creating a new Web platform game to put on your site. A designer just handed over the graphic elements of the game and you have to put them all together and create the game mechanics."

After introducing the user to this scenario the facilitator explains where he can find the tools to do each of the tasks that he will propose to him. The materials at the disposal of the user are:

A folder with graphical elements – This folder contains the Flash projects that the designer produced for the game's elements.

A folder with a complete game – This folder contains all the elements of a complete game. This includes costume code for AI elements and GUI elements, XML files for levels and game, and Flash projects for each of the elements that compose the game.

A folder with the Platform Games Engine – This folder contains all the code of the Platform Games Engine, the main Flash project that launches game engine and the XML Schema files that define the game and level's XML file dialects.

Tools for editing the projects files – These tools are simple texts editors to edit the XML files (Notepad and Visual Studio), Adobe Flash 8 to edit the Flash projects and compile the Platform Games Engine and SEPY editor to edit action script files.

This scenario is made of a group of five tasks. Each task's purpose is to test the usability of a different aspect or feature of the game engine. Each of the five tasks is described in the next sections.

Keep in mind that the facilitator cannot help the user in his task, his role is as a simple observer.

5.2.1.1.1 Task 1 – Creating the Game XML File

For this task the facilitator tells the user the following:

> "Create the main XML file for the game. The game has one character and two levels. The player will use the arrow keys to move around and to jump."

The goal of this test is to see if the user can easily create the Game XML file.

The user is expected to use the tools and elements at his disposal to execute this task. It is expected that he uses the XML file in the example has a starting point and that he alters it to fit the task's description. Another expected scenario is that the user uses the XML document definition to see how to build the XML file.

5.2.1.1.2 Task 2 – Create Level 1's XML File

For this task the facilitator tells the user the following:

> "Create the Level XML file for the first Level. This level is composed of 3 instances of the enemy that was designed by the Designer, and 2 instances of the item also designed by the Designer. The level will also be of the Scrolling type and will be setup so that it scrolls right automatically at a speed of 2 pixels for frame."

The goal of this task is to see if the user can easily create a level's XML file. It is also tested if the user can easily customize the level's mechanics.

The user is expected to use the tools and elements given to execute this task. One of the expected user paths is that he uses the example's level XML file and alters it to fit the task's description. Another expected path is that the user uses the XML document definition to see how to build the XML file.

5.2.1.1.3 Task 3 – Create Level 2's XML File

For this task the facilitator tells the user the following:

> "Create the Level XML file for the second Level. This level is composed of 2 instances of the item also designed by the designer of which one gives the player two lives; it is also composed of a spawn point for the enemy used in the previous level that spawns enemies every 1000 milliseconds. The level will also be of the Scrolling type and will be setup so that it only scrolls right as the player crosses the level."

The goal of this task is to observe the difference in users ease in creating a level XML file after the previous task, observing the user's learning curve.

It is expected that the user follows a similar of the one he did in the previous task.

5.2.1.1.4 Task 4 – Deploying the Game

For this task the facilitator tells the user the following:

> "Modify the Platform Game Engine platform so that it loads the game XML file created and consequently the game elements and mechanics implemented."

The goal of this task is to see if the user can easily alter the game engine's Flash project to deploy is newly created game.

It is expected that the user opens the Flash project and changes the source variable of the Game Engine's call to read from the newly created XML file.

5.2.1.1.5 Task 5 – Creating a Custom AI Class

For this task the facilitator tells the user the following:

> "Create a simple AIBot that implements the AIBotElement interface of the Game Engine Platform API and extends the AIBot class. Then modify the enemy graphical element so that it sets up this new class."

The goal of this task is to see if the user can easily create its own Artificial intelligence class and use it in the Game Engine.

It is expected that the user uses the costume AI class of the example to alter or see how he can create his own.

5.2.1.2 Scenario 2 – Altering an already created game

In this scenario the user in invited to image the following situation:

> "In this scenario you're working on an already created game that you want to modify to include some new elements and some new mechanics."

After introducing the user to this scenario the facilitator explains where he can find the tools to do each of the tasks that he will propose to him. The materials at the disposal of the user are the same as in the previous scenario (The folder with the new graphical elements, the folder with complete game example, the folder with the Platform Games Engine, and the same tools for editing the projects files) plus one new one:

A folder with the game – This folder contains the game that the user has to alter in this scenario.

This scenario is made of a group of nine tasks. Each task's purpose is to test the usability of a different aspect or feature of the game engine. Each of the nine tasks is described in the next sections.

Keep in mind that, just like in the previous scenario, the facilitator cannot help the user in his task, his role is as a simple observer.

5.2.1.2.1 Task 1 – Creating a new Player's Character

For this task the facilitator tells the user the following:

> "The first task that is proposed to you is to change the player's graphical element for the second level to the new one the designer created called "newPlayer.fla"."

The goal of this task is to observe if the user can easily add a new character to the game and associate it with a level.

This task only evolves changes in the Game XML file, so it is expected that the user goes to the Game XML file and adds a new Character element to the file. The user is expected to copy the already existing definition of a character in the file and change the library it refers to so that it imports the new character's graphical element.

5.2.1.2.2 Task 2 – Changing a Level's Enemy Graphical Representation

For this task the facilitator tells the user the following:

> "Now you will need to change the element that is loaded in the second level representing the enemies so that it loads the new version created by the designer (the "newEnemy.fla")."

The goal of this task is to see if the user can easily change an enemy of a level.

The user is expected to change the library attribute value for the EnemyWaypoint elements of the level. Another possible way to do this task is to replace the SWF file imported in the Level XML file to the new enemy element.

5.2.1.2.3 Task 3 – Changing a Level to Have Auto Scrolling

For this task the facilitator tells the user the following:

> "A change to the game mechanics is also needed. Now the game designer wants the first level to have Auto scrolling right with the speed of 2 pixels per frame."

The goal of this task is to observe how the user changes a level's mechanics, and how easy he considers the process to be.

The user is expected to change the Level's LevelScrolling element and add the scrollingSpeed attribute with the value "2". To do this the user is expected to consult the game example or the XML document definition.

5.2.1.2.4 Task 4 – Changing the Game Level's Order

For this task the facilitator tells the user the following:

> "The game designer also decided that he wants to change the order in which the levels are played. So now you need to change the order so that the second level is played first and the first level is player second."

The goal of this task is if the user considers changing a game's level order to be a simple task or not.

To perform this task the user is expected to go to the Game XML file and change the order in which the Level elements is declared. Alternatively the user can rename the Level XML files (renaming level 1 file to the name of the level 2 file and vice-versa).

5.2.1.2.5 Task 5 – Adding a new Item to a Level

For this task the facilitator tells the user the following:

> "You now have to add a new instance of the item that is used in level one. This new instance should be added at the end of the level right before the "end" element. This instance should give Zero points to the player but it should give him three lives."

The goal of this test is to observe if the user can add new items to a level in a simple way.

The user is expected to go to the level XML file and add a new ItemWaypoint element (by writing the new element or copying one of the existing elements). The user then is expected to open the elements Flash project (where the elements waypoints are defined) and add a new waypoint to the end of the level naming it accordantly with the name given in the XML file.

5.2.1.2.6 Task 6 – Adding a new Enemy to a Level

For this task the facilitator tells the user the following:

> "As you added a new item at the end of level one you will need to add an enemy instance to defend it. You should add an instance of the enemies that are used in the first level to the end of the level right next to the item."

The goal of this test is to observe if the user can add new enemies to a level in a simple way.

The user is expected to go to the level XML file and add a new EnemyWaypoint element (by writing the new element or copying one of the existing elements). The user then is expected to open the elements Flash project (where the elements waypoints are defined) and add a new waypoint to the end of the level, right next to the previously added item, naming it accordantly with the name given in the XML file.

5.2.1.2.7 Task 7 – Changing the Game's Keys

For this task the facilitator tells the user the following:

> "Another change that is proposed by the game designer is to change the game keys from the arrow keys to the "W, A, S, D" from."

The goal of this test is to observe if the user can easily change a game's keys.

To perform this task the user only needs to change the keycode property of the different Key elements defined in the Game XML file. The user is expected to change alter the key codes to the ones defined in the task using an online key code table (or other form of getting the key code values of the given keys).

5.2.1.2.8 Task 8 – Repositioning a Level's Elements

For this task the facilitator tells the user the following:

> "Some testes complained that in the middle of the second level the enemies are all in the same place and should be more distributed. You should change the positions of these enemies."

The goal of this test is to see if the user can easily change the position of the game element of a level.

To do this task the player has to alter the position of waypoints defined in the elements Flash project. The user is expected to open the flash project and reposition the elements that have the name like "enemyX" ("X" represents the number of the enemy).

5.2.1.2.9 Task 9 – Changing the Game's GUI

For this task the facilitator tells the user the following:

> "Finally the gamers complained that there was not a timer in the game, so the designer modified the GUI so that it has that timer element. You should now change the GUI manager class so that it updates that element."

The goal of this test is to observe is the user can easily change an already existing GUI.

The user will need to alter the GUI costume class that is created because the new GUI comes with a timer element and the management of this element must be implemented in the costume class. The user is expected to open the old GUI Flash project and to see where he can find the class

that manages the element. Then the user is expect to implement the same class but now for the new GUI element produced by the designer. Finally the user has to alter the "updateTimer" method so it updates the timer element of the new GUI.

5.2.2 Test Results

The previously defined tests were made with four of FullSix's multimedia developers. While these users where performing the suggested tasks some notes were taken about their comments, suggestions, and their frustration level. All of this data has been gathered and is presented in the following sections as well as the results of the System Usability Scale questionnaire.

5.2.2.1 User Frustration Level Results for Scenario 1

In the first scenario the user's frustration was evaluated as shown in Table 5.1.

Table 5.1:User Frustration Level Results for Usability Tests with Developers (Scenario 1)

	Frustration Level					
	task 1	task 2	task 3	task 4	task 5	Average
user 1	1	1	0	0	1	0.6
user 2	1	2	1	0	2	1.2
user 3	1	2	2	0	3	1.6
user 4	3	2	2	0	3	2
Average	1.5	1.75	1.25	0	2.25	1.35

From this table we can see that the user's average frustration level varies from 0.6 to 2 (in a 0 to 3 scale), and that the total average for this scenario is 1.35.We can view in more detail the average frustration level for each task in Figure 5.1.

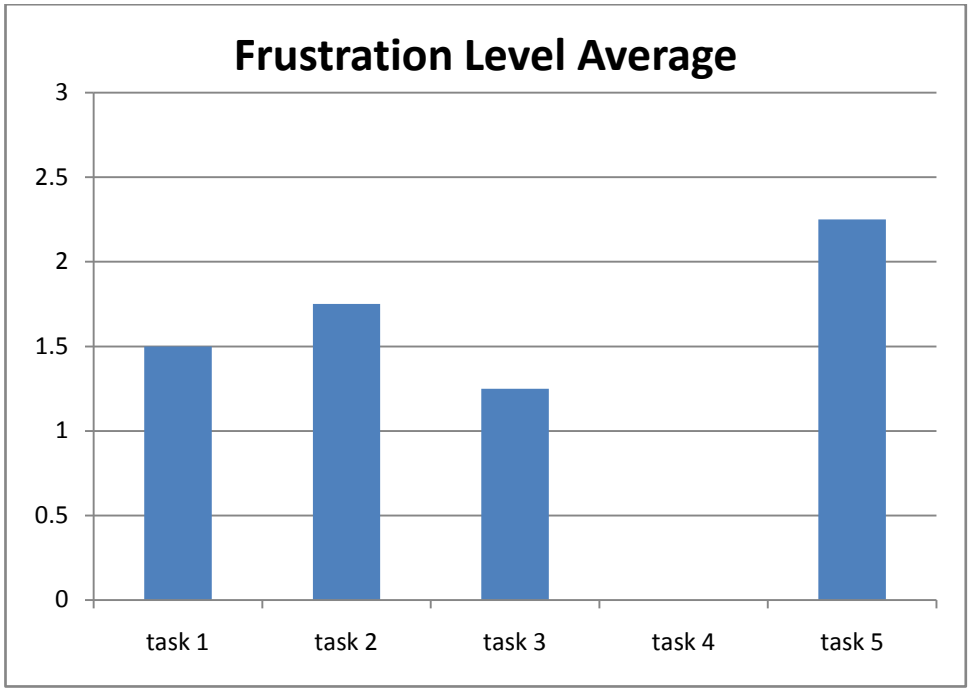


Figure 5.1: Frustration level average for the developers usability tests (Scenario 1)

Here we can see that the lowest average frustration level was in task 4 and the highest was in task 5.

5.2.2.2 User Frustration Level Results for Scenario 2

For the second scenario the user's frustration level was registered as shown in Table 5.2.

Table 5.2:User Frustration Level Results For Usability Tests With Developers (Scenario 2)

	Frustration Level									
	task 1	**task 2**	**task 3**	**task 4**	**task 5**	**task 6**	**task 7**	**task 8**	**task 9**	**Average**
user 1	0	0	1	0	0	0	2	0	0	0.33
user 2	0	0	1	0	0	0	2	0	0	0.33
user 3	0	0	2	0	1	0	2	0	1	0.67
user 4	0	1	3	0	1	0	3	0	1	1.00
Average	0	0.25	1.75	0	0.5	0	2.25	0	0.5	0.58

From this table we can see that the user's average frustration level varies from 0.33 to 1 (in a 0 to 3 scale), and that the total average for this scenario is 0.58.We can view in more detail the average frustration level for each task in Figure 5.2.

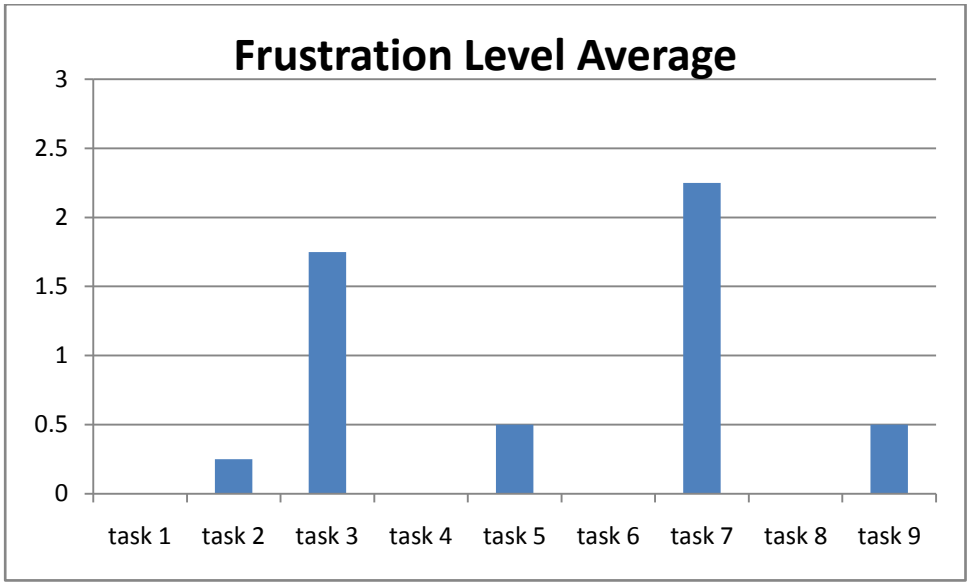


Figure 5.2: Frustration level average for the developers usability tests (Scenario 2)

Here we can see that tasks 1, 4, 6 and 8 got the lowest frustration score and that task 7 has the highest.

5.2.2.3 System Usability Scale Results

The results of the System Usability Scale's (SUS) questionnaire were gathered and processed as explained in section 5.1. The results of this questionnaire are shown in Table 5.3.

Table 5.3: SUS's Questionnaire results for the usability tests with developers

	user 1	user 2	user 3	user 4
Question 1	5	5	4	4
Question 2	2	1	2	3
Question 3	5	5	4	3
Question 4	1	1	1	3
Question 5	4	3	3	3
Question 6	1	2	2	2
Question 7	5	4	4	3
Question 8	2	3	3	2
Question 9	4	4	3	3
Question 10	1	1	2	2
SUS Score	90.00%	82.50%	70.00%	60.00%
SUS Average		75.63%		

The system got a SUS average of 75.63% from the developer's point of view. It is also important to note that the highest SUS score was 90% and the lowest was 60%.

5.2.2.4 Suggested Improvements

During these tests the users made some suggestions on how to improve the system. These suggestions are shown in Figure 5.3 along with the number of users that made the suggestions. It is important to note that four developers took this test.

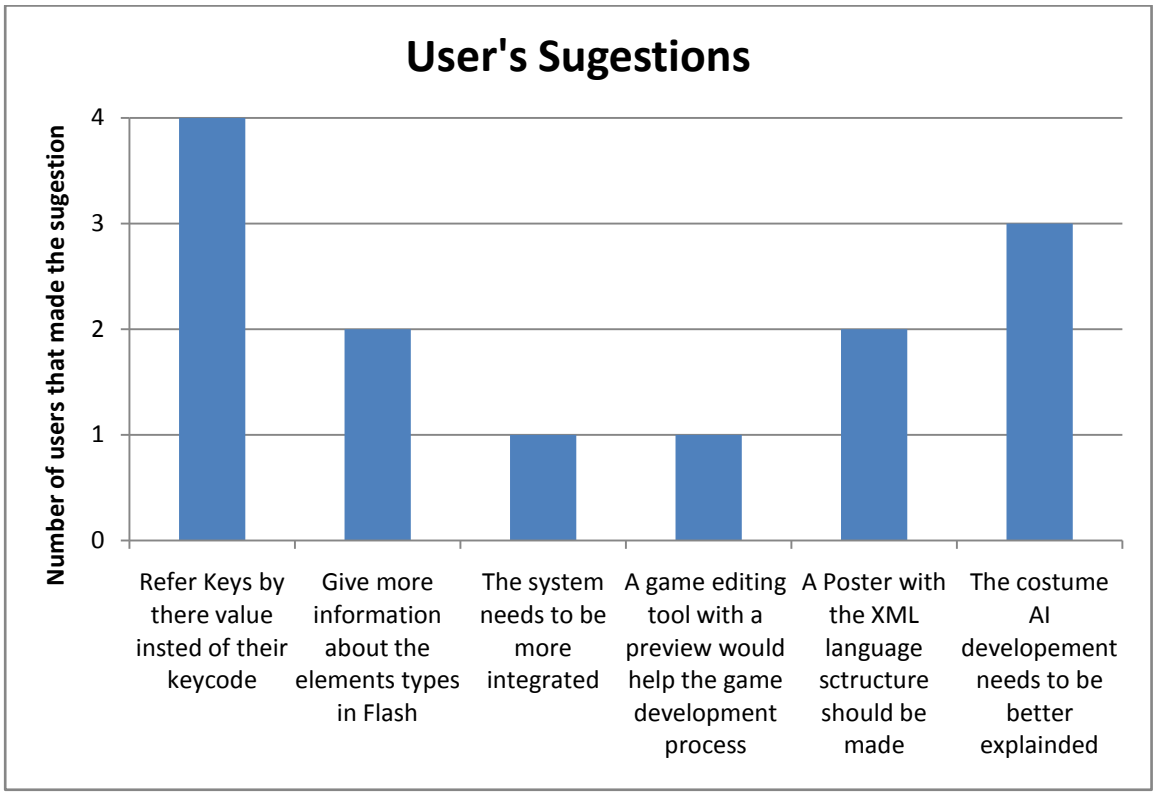


Figure 5.3: Chart for the developer's usability tests suggestions

5.3 Tests with Designers

Another of the users of this game engine will be the designers at FullSix. Although the game engine's purpose is to reduce the development time in developing platform game engines, the fact that developers also understand and use the game engine can be very helpful for developers and can significantly reduce the projects developing time as well. For this reason it is required that they have a chance to evaluate and give feedback about the engine. This feedback is very important not only improve the game engine but mainly because designers are non expert users in the development of games and can give valuable information about the platform's usability.

The following sections will describe the tests that were made as well as the obtained results and a small analysis of these results.

5.3.1 Test Description

The target users for these tests were the designers at FullSix, more specifically the ones that design elements for flash projects and games. These tests' goal is to get feedback on the usability of the Platform Games Engine as a development platform in the designer's point of view.

For this task one scenario was created, composed by a group of tasks that the users were asked to do. The tasks were designed to challenge the user to use different parts of the game engine and get a look and feel of using this platform. The objective of creating scenarios is to simulate, as much as possible, a real working environment were the user is trying to execute a project just like in real working conditions.

The created scenario was:

Scenario 1 – Altering an already created game

The reason that designs don't have a scenario where they create a new game is that it would involve a lot of development and these users aren't prepared for those aspects.

The conditions for the execution of these tests are the same for all the users. Initially the test facilitator (the person that is giving the user the task he has to do and that observes the user and takes notes about its comments) asks the user a group of questions about himself, next he explains the scenario to the user and gives him the first task. When the user finishes (or gives up) the task the user will give him the next task and so on until the end of the test. At the end of the test the user will have a questionnaire based on the System Usability Scale by John Brook.

The next sections will describe each of the scenarios and the tasks that compose them.

5.3.1.1 Scenario 1 – Altering an already created game

In this scenario the user in invited to image the following situation:

> "You are designing a new platform game that uses the Platform Game Engine. The game is already created but some graphical changes have to be made to the game before it is finished. You will have to do these changes."

After introducing the user to this scenario the facilitator explains where he can find the tools to do each of the tasks that he will propose to him. The materials at the disposal of the user are:

A folder with a complete game – This folder contains all the elements of a complete game as an example for the user. This includes costume code for AI elements and GUI elements, XML files for levels and game, and Flash projects for each of the elements that compose the game.

A folder with the Platform Games Engine – This folder contains all the code of the Platform Games Engine, the main Flash project that launches game engine and the XML Schema files that define the game and level's XML file dialects.

Tools for editing the projects files – These tools are simple texts editors to edit the XML files (Notepad and Visual Studio) and Adobe Flash 8 to edit the Flash projects and compile the Platform Games.

A folder with the game to alter – This folder contains the Flash projects and XML file of the game the user is asked to alter.

This scenario is made of a group of five tasks. Each task's purpose is to test the usability of a different aspect or feature of the game engine. Each of the five tasks is described in the next sections.

Keep in mind that the facilitator cannot help the user in his task, his role is as a simple observer.

5.3.1.1.1 Task 1 – Changing the player's look

For this task the facilitator tells the user the following:

> "The first change that you have to do is to alter is graphic element for the player's character. Testers did not like his clothing so you should change it to a more appropriate look."

The goal of this test is to see if the user can easily alter the graphical aspect of the player's character.

The game engine won't assist the user in the designer part of this task but it is expected that if he uses the platform the new element will be integrated in the game without the need for the developer's intervention.

The user is expected to use the Game XML file to see where he can find the player's character and open its Flash project. Then he is expected to alter the element as he wishes and publish the element, which will be automatically integrated in the game engine.

5.3.1.1.2 Task 2 – Creating a new enemy

For this task the facilitator tells the user the following:

> "Another change that was proposes is to create a second type of enemy based on the first and put it in the game's level one (replacing at random some of the enemies that already are included in the level)."

The goal of this test is to observe if the user can create a new enemy element to replace one that is already being used in a level.

The user is expected to use the old enemy element and alter it and publish its new version. To find the element the user is expected to go to the level XML file and find from which element is the game engine getting the enemy's graphic. Then open the Flash project for that element and perform the changes.

5.3.1.1.3 Task 3 – Changing an Item

For this task the facilitator tells the user the following:

> "Next you will have to add a new item to the first level to replace the one that is used, Testers complained that the item was too small and said that a larger version of the item should be used."

The goal of this task is to observe if the user can easily change the graphical aspect of an item that is being used in a game level.

The user is expected to use the old item element and alter it and publish its new version. To find the element the user is expected to go to the level XML file and find from which element is the game engine getting the item's graphic. Then open the Flash project for that element and perform the changes.

5.3.1.1.4 Task 4 – Adding more instances of Items

For this task the facilitator tells the user the following:

> "About the first level, testers also said that it needed more items. So you'll need to create more instances of the item used in this level, distributing these items on the level."

The goal of this task is to see if the user can easily add new instances of an already existing item in a level.

In this task the user is expected to open and alter the level's XML file and the Flash project for the level's waypoints. The user is expected to open the XML file and add new instances of the ItemWaypoint element (duplicating the existing ones and changing its names). Next the user is expected to open the element waypoint's Flash project and create new waypoint with the corresponding names.

5.3.1.1.5 Task 5 – Adding New Platforms to a Level

For this task the facilitator tells the user the following:

> "Finally the game designer asked you to create some platforms in the first level so that the player had more freedom on the way that he scrolls through the level."

The goal of this task is to see if the user can easily change the level's flow by adding new platforms to the level.

The user is expected to open the hit-map and background Flash projects and add some platforms to the game. Then the user is expected to add some elements that represent platform in the background and add some graphical element at the same place in the hit-map element. It is expected that when the user publishes these elements they will be integrated in the game engine without the need for the developer's intervention.

5.3.2 Test Results

The previously defined tests were made with three of FullSix's designers. While these users where performing the suggested tasks some notes were taken about their comments, suggestions, and their frustration level. All of this data has been gathered and is presented in the following sections as well as the results of the System Usability Scale questionnaire.

5.3.2.1 User Frustration Level Results

In the performed user tests the user's frustration was evaluated as shown in Table 5.4.

Table 5.4: Frustration Level Results for Usability Tests with Designers

	Frustration Level					
	task 1	task 2	task 3	task 4	task 5	Average
user 1	1	0	0	2	1	0.8
user 2	2	2	2	3	3	2.4
user 3	1	1	1	3	2	1.6
Average	1.33	1.00	1.00	2.67	2.00	1.6

From this table we can see that the user's average frustration level varies from 0.8 to 2.4 (in a 0 to 3 scale), and that the total average for this scenario is 1.6. We can view in more detail the average frustration level for each task in Figure 5.1.

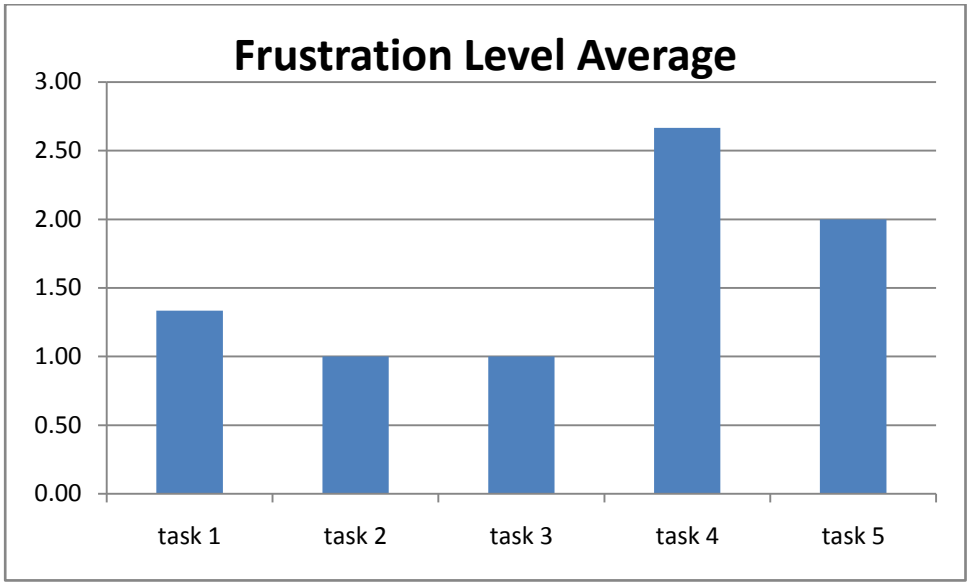


Figure 5.4: Frustration level average for the designers usability tests

Here we can see that tasks 2 and 3 got the lowest frustration score and that task 4 has the highest (almost 3 points).

5.3.2.2 System Usability Scale Results

The results of the System Usability Scale's (SUS) questionnaire were gathered and processed as explained in section 5.1. The results of this questionnaire are shown in.

Table 5.5: SUS's Questionnaire results for the usability tests with designers

	user 1	user 2	user 3
Question 1	1	4	2
Question 2	2	4	2
Question 3	4	2	3
Question 4	3	4	3
Question 5	3	2	2
Question 6	2	2	3
Question 7	3	2	2
Question 8	2	3	2
Question 9	4	3	4
Question 10	2	4	3
SUS Score	60.00%	40.00%	50.00%
SUS Average		50.00%	

The system got a SUS average of 50% from the designer's point of view. It is also important to note that the highest SUS score was 60% and the lowest was 40%.

5.3.2.3 Suggested Improvements

During these tests the users made some suggestions on how to improve the system. These suggestions are shown in Figure 5.5 along with the number of users that made the suggestions. It is important to note that three designers took this test.

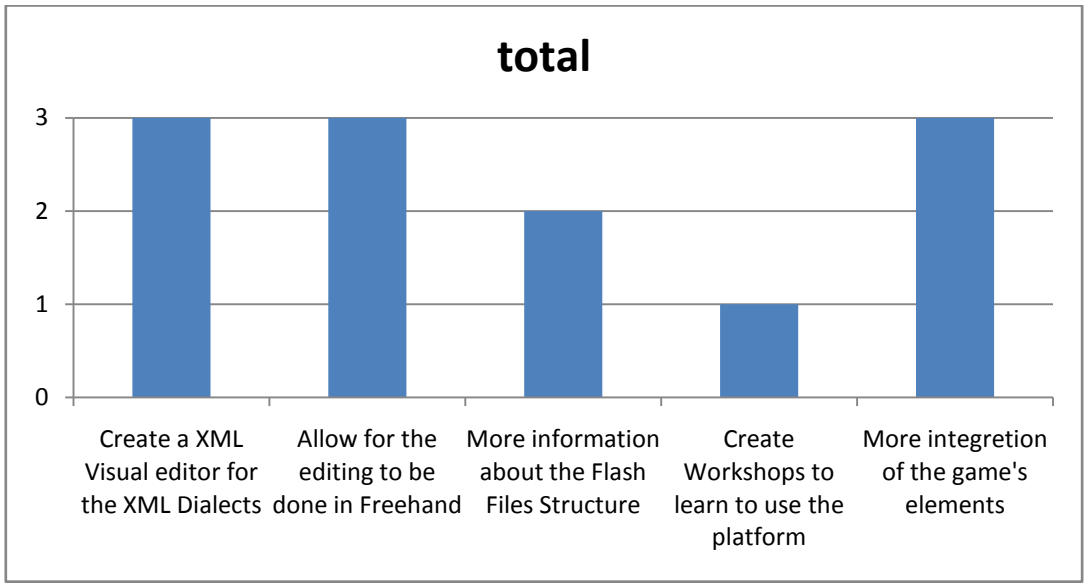


Figure 5.5: Chart for the designer's usability tests suggestions

5.4 Performance Tests

Performance is a key aspect of any game. Users don't want the game to lag while they are playing and for this game engine to be used to produce platform games in a large scale at FullSix performance must be guaranteed.

To measure the system's performance a heuristic is needed that the results can be evaluated and conclusions can be drawn. In this particular case, a game's performance can be measured by the systems frame rate.

The frame rate defines if the system is overloaded or not because if too much of the processing time is spent in calculations and logical operations (like physics or artificial intelligence) then there will be less frames being drawn every second. Although the frame rate does not maintain the same through the game (not even in optimal conditions) if its value drops too low then the player will experience little breaks in the games visual flow. To guarantee that the user's perception of the game's performance is good the frame rate cannot drop below 17 frames per second. Below the referred value the game elements will start to move to slowly and we can consider the system's performance to be unsatisfying. [Dic]

Based on these descriptions the game engine's performance will be tested based on the four tests described in the next section. The tests results and analysis will also be described in the next sections.

5.4.1 Test Description

In this game engine there are mainly two of the game elements that can cause performance problems, they are:

Number of Enemies – The number of enemy elements will call more processor need by the physics engine and artificial intelligence manager, and this may cause performance problems.

Number of Items – The number of item elements present in a level can cause some performance problems because the game engine has to test the intersections between the player and each of the items.

Both of these facts have been taken into consideration when developing the game engine. One of the measures that were taken was to test do collision detection only for the elements in the screen, because the viewport always follows the player it is impossible to have a collision with items outside the viewport.

Another precaution was relative to the artificial intelligence manager and physics manager when processing the list of enemies in the level. Like in the case of the items, the enemies that are outside the viewport should not move, so first the engine tests if the element is inside the screen or not and only processes the elements that are.

These optimisations led to the conclusion that there are two types of tests to be made, the number of elements inside the viewport that the engine can handle and the number of elements outside the viewport. This said then there are for distinct performance tests to be made, this is to test the number of items and enemies that the engine can handle inside and outside the viewport. Each of these tests is described in the next sections.

5.4.1.1 Test 1 – System's behaviour concerning the number of enemies on screen

The goal of this test is to study the way the system behaves with a large number of enemies on screen at the same time. This test is important to see if the system can support an acceptable number of enemies on the screen to allow the creation of a playable game. Based on the system's architecture this analysis will be based on the game's frame rate for a given number of enemies present on the screen at a given time. The optimal frame rate for the game is 32 fps, and the minimal frame rate is 17 fps.

For this test it will be used a level composed by an elevated platform, where the player will stand so as not to be hit by the enemies, a hit area that does not let the enemies leave the screen and an enemy spawn point that will create an enemy each second. The platform will be outputting the frame rate and the number of enemies that are on the screen at that moment.

To calculate the frame rate it will be used the following mathematical expression [Cla07]:

$$frameRate = \frac{1000}{CurrentTime - TimeLastFrame}$$

Note that the $CurrentTime$ and $TimeLastFrame$ variables are measured in milliseconds. Also note that the frame rate will be calculated for each frame.

5.4.1.2 Test 2 – System's behaviour concerning the number of enemies outside the screen

It is known that a platform game level can be very large in terms of length. This means that there will be a lot of enemies off the screen when the user is playing the game. This test's goal is to study the system's behaviour with large amounts of enemies spread throughout the level. The variables

that will be analyzed in this test will be the same ones that were analyzed in Test 1, they are the frame rate and the number of enemies, but this time the enemies that will count will be the ones out of the screen. As it is said in section 5.4.1.1, the optimal value for the frame rate is 32 fps, and the minimal value is 17 fps.

For this test a level will be setup so that has an enemy spawn point is placed outside the screen. Then some changes were made to the condition that the spawn point element has that prevents it from generating enemies if it is outside of the screen. Then the Game platform will output the value for the two variables in study in this test, the frame rate and the number of enemies outside the screen.

To calculate the frame rate the same mathematical expression that was used in section 5.4.1.1 (Test 1 – System's behaviour concerning the number of enemies on screen) will be used.

5.4.1.3 Test 3 – System's behaviour concerning the number of items on the screen

Although the architecture of the Game Engine gives less processing power to an item than an enemy (because the item isn't affected by gravity nor can be moved and collide with the level's hit-map) the engine test collisions between the player and the items. This fact creates the need to analyze the performance of the Game Engine based on the number of items on the screen to see if the number of items that can be on the screen, without degrading the user experience, is acceptable.

This test consists on a simple level that has an item. Then a script was that runs on this level and creates a duplicate of that item on a random position on the screen every second. The platform will output the two variables needed to perform the analysis for this test, these are the frame rate and the number of items on the screen at a given instant. The frame rate will be calculated using the same mathematical expression as described in section 5.4.1.1.

5.4.1.4 Test 4 – System's behaviour concerning the number of items outside the screen

This test is important for the same reasons as the test described in section 5.4.1.3. The only difference is that now the test is focused on the number of items outside the screen. The levels can have a great length, meaning that the items are distributed along that length and outside the boundaries of the screen. This means that there will be a lot of items outside the screen in each level and the Game Engine has to be able to perform within acceptable values.

This test will be executed in the same manner as the test described in section 5.4.1.3. A level will be created with only one item that is placed outside the screen somewhere in the level. Then a script will run on the level that will create a duplicate of this item and place it somewhere on the level (as long as it is outside the screen), the script will loop, repeating this on each second. The Game Engine will output the value of the current frame rate (this value is calculated according to the mathematical expression defined in section 5.4.1.1) and the number of existing items outside of the screen on each instant.

5.4.2 Test Results

The performance tests results for each of the preciously described tests are presented in the next sections.

5.4.2.1 Test 1 Results

The results for test 1, evaluating the system's frame rate with a great number of enemies inside the viewport, are presented in Figure 5.6.

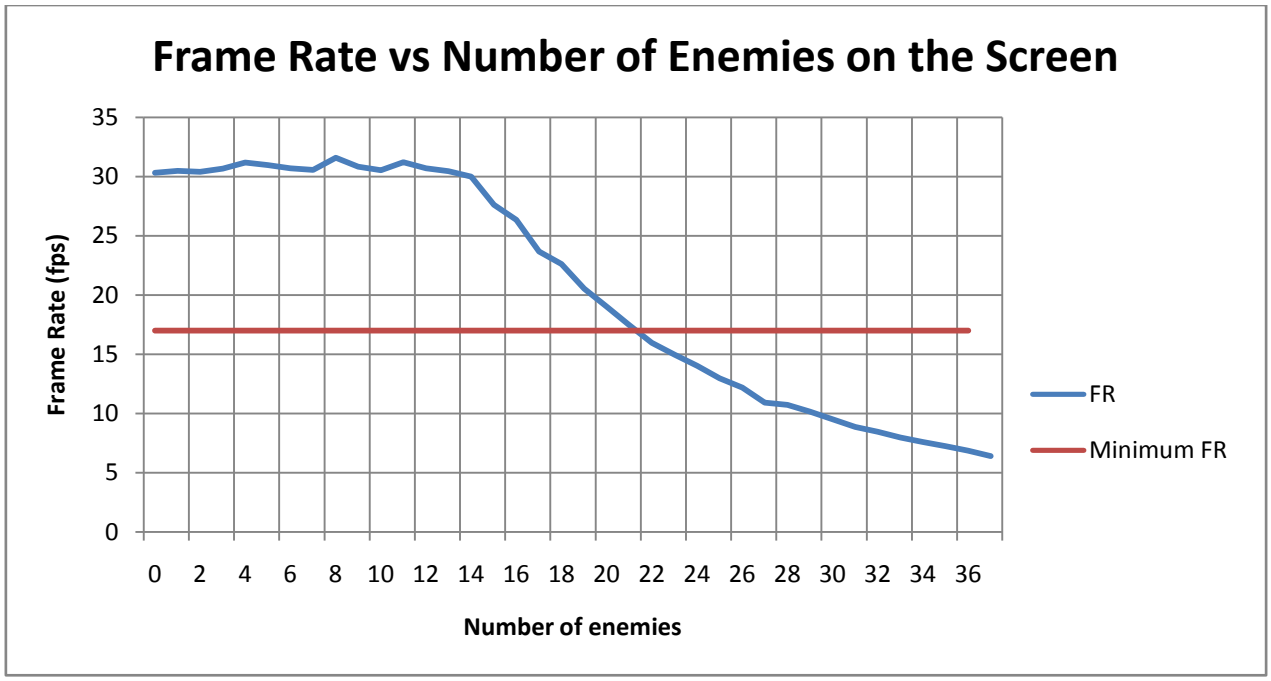


Figure 5.6: Chart of the Frame Rate with a different amount of enemies on the screen

As we can see the system maintains a stable frame rate level until approximately 14 enemies are added, then the frame rate starts to drop reaching the minimum acceptable level with 22 enemies on the screen.

5.4.2.2 Test 2 Results

The results for test 2, evaluating the system's frame rate with a great number of enemies outside the viewport, are presented in Figure 5.7.

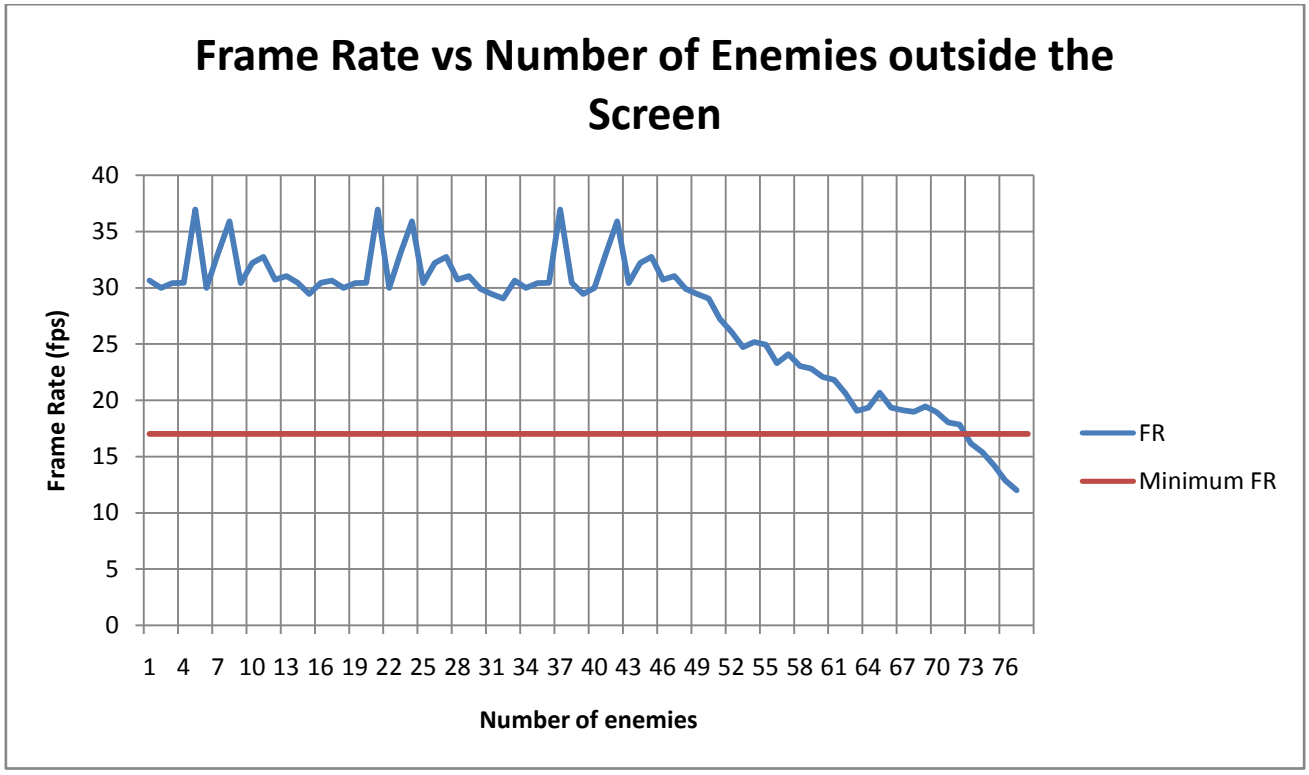


Figure 5.7: Chart of the Frame Rate with a different amount of enemies outside the screen

As we can see the system shows stable performance values until 49 enemies, at this point the frame rate value starts to drop reaching the minimum acceptable value at 73 enemies added.

5.4.2.3 Test 3 Results

The results for test 3, evaluating the system's frame rate with a great number of items in the viewport, are presented in Figure 5.8.

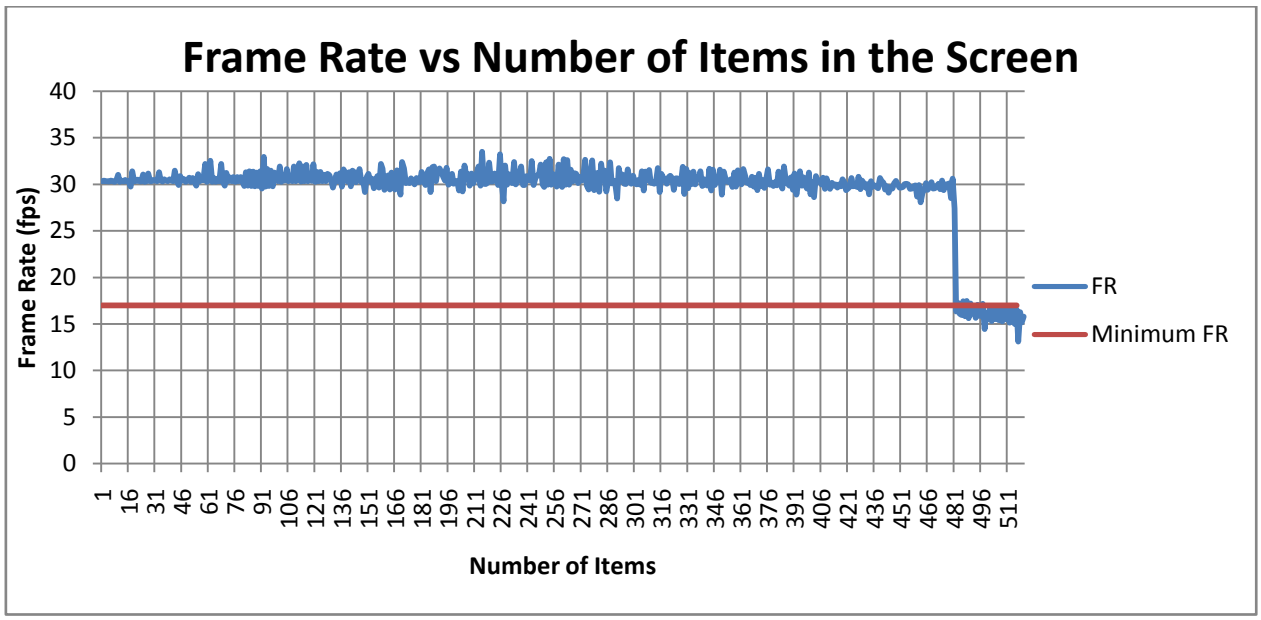


Figure 5.8: Chart of the Frame Rate with a different amount of items in the screen

This chart shows that the system maintains normal frame rate values until approximately 481 items are on the screen. At this point the frame rate drops very rapidly to values below the minimum acceptable level.

5.4.2.4 Test 4 Results

The results for test 4, evaluating the system's frame rate with a great number of items in the viewport, are presented in Figure 5.9.

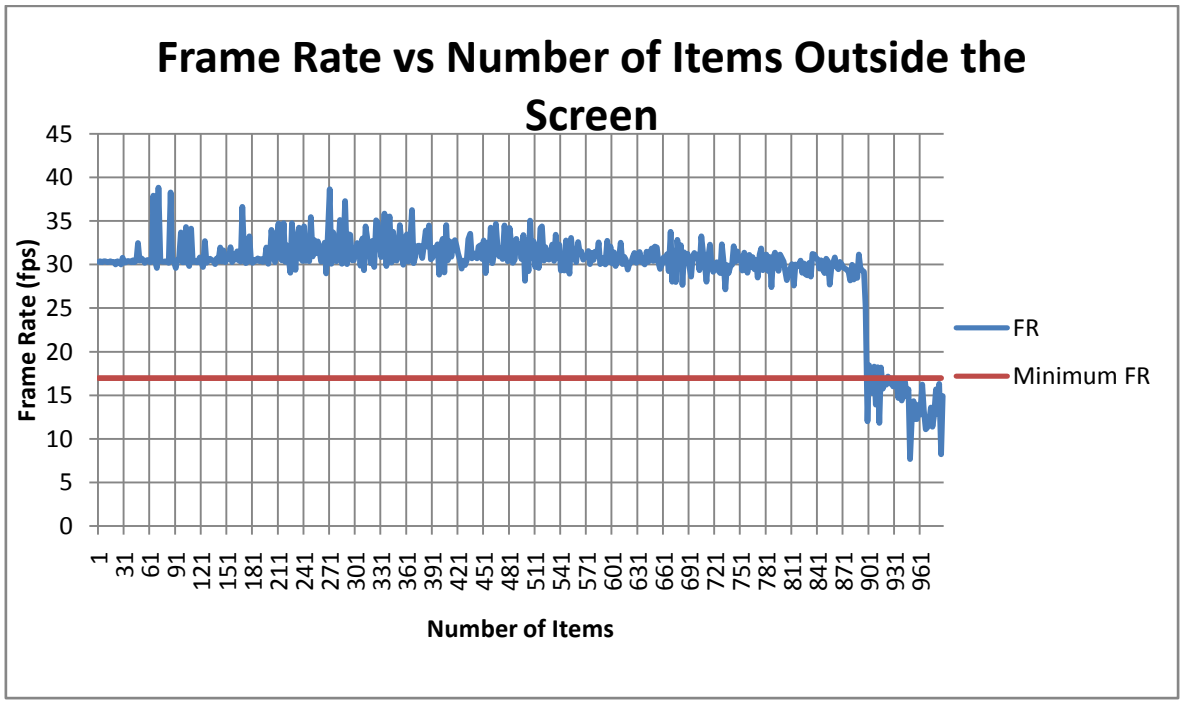


Figure 5.9: Chart of the Frame Rate with a different amount of items outside the screen

This chart shows that the system maintains normal frame rate values until approximately 901 items are on the screen. At this point the frame rate drops very rapidly to values below the minimum acceptable level.

5.5 Summary

When developing a tool like the one developed on this work, tests are a very important part of the project. The goal of this work is to create a tool that developers use for creating games at FullSix. One of the key factors for reaching this goal is the usability of the tool. The users have to feel comfortable using it and they have to be the ones that judge if the tool is really easy to use. This led to usability tests being made with developers (the target user for this tool) and designers (although designers are not the target user of this tool, they have a very important role in the development of the game). The results obtained were, overall, very positive. Developers gave very good feedback and were pleased with the tool, and with what it can do, however some improvements were suggested. Designers, on the other hand, had some problems using the tool. In spite of not having a negative feedback, they were not totally pleased with the usability of the tool, and suggested some improvements.

Performance tests also were made to evaluate how the tool handles great number of elements. Performance is a great issue in games, and a game engine should be able to guarantee the developer optimal performance values. This engine got very good performance results.

Chapter 6

Conclusions and Critical Analysis

6.1 The project's Usability

Analysing the results of both usability tests some conclusions can be drawn. In the next sections the different results are analysed separately. In the first section the users' frustration levels are analyzed and in detail, next the system's usability scale questionnaire results are analysed and finally the last section analyzes suggestions that the users made during the tests.

6.1.1 The users' Frustration Level Analysis

First analysing the tests made with developers, the frustration average was 1.35 (in a 0 to 3 scale where 0 represent total satisfaction and 3 total frustration) in the first scenario and 0.58 in the second scenario. The first thing to notice is the difference between both results, since that they used the same platform in both. A deeper analysis of this point is needed to justify this difference. What happened was that in the first scenario users had to build a game from scratch and this was their first contact with the platform. To do these tasks users had to learn how the platform works and as a result some frustration was felt executing the tasks. When this scenario ended users had have contact with all the major aspects of the platform, this is when the second scenario started, with all the knowledge they had acquired in the first scenario this one was a lot easier, users knew where to find the elements and which files to edit creating a feeling of trust in them instead of the feeling of frustration felt in the previous scenario, explaining the difference between both frustration levels.

To better understand the difficulties that the users felt during the tests an analysis of the frustration levels of each of task is needed. Starting with the first scenario, the results show that users felt the most difficulty in the last task that consisted in developing the artificial intelligence for the enemies of the game they built, although the tasks one, two and three also had some

elevated frustration levels (as shown in Figure 5.1). Creating a game is a very complex task, although the Platform Games Engine helps developers to build the game the users that were testing the platform did not had any time to tryout the platform and thus did not know how to solve the tasks. This fact was not taken into consideration when designing the tests and can be behind the fact that the frustration levels are higher in the first scenario than in the second. Another fact that supports this conclusion is that, apart of the frustration, all the users executed successfully all the tasks, with the exception of 3 tasks total. Analysing the results of the second scenario they show that the worst scores were obtained in tasks three and seven, and the remaining tasks got optimal or almost optimal results (as shown in Figure 5.2). Task three consisted in changing the level mechanics to make the level an auto scrolling level. Although this task involves a little change to be made in the level XML file, users had a lot of difficulties finding out which of the properties defined the fact that the level is auto scrolling explaining the high level of frustration. This fact is an important feedback and a correction that should be considered in future versions of the game engine. Task seven consisted in changing the game's keys. To do this the users only had to change the key code attributes of the keys defined in the XML file according to the new key configuration. The users found the XML file and the attribute with little effort, their frustration was due to the fact that they had to consult a key code table to see the values for the new keys. This task was the most criticised by the users because of the fact that the language did not support other formats than the key codes. This remark as a significant importance and should be corrected in future versions of the platform.

Analysing the results of the frustration levels of the designers' usability tests we can see that the average frustration level was 1.6 (in the same 0 to 3 scale that was previously explained). It is a high score, 0.1 higher than the middle value of the scale. The designers' high frustration is a warning factor that some usability changes must be made in the system so that it can be better accepted by these designers. For a better understanding a more detailed analysis of the frustration levels of each task is important (the results are shown in Figure 5.4). In this analysis we can see that the first three tasks have almost the same average frustration level, they score approximately 1.0. The last two tasks got very high frustration level scores, task four got a score of 2.67, and task five a score of 2.00. These tasks were the ones where users had to do changes in the XML files, and this was the main factor for their frustration, these designers are not used to do changes in text-based files, not even in XML, which explained the high frustration levels, and the number of them that did not complete the tasks.

6.1.2 System Usability Scale Questionnaire Analysis

At the end of both scenarios users answered a questionnaire intended to evaluate the users' satisfaction with the system's usability (this questionnaire is explained in section 5.1).

First analyzing the developer's results for this questionnaire we can see that the average score was 75.63%. Although this score is very good there is still room for improvement in terms of usability, and from the analysis made of the users' frustration levels some problems have already been identified.

Now analyzing the designers' results for this questionnaire we can see that the average score was 50.00%. This result shows that the users' lack of satisfaction with the system's usability is obvious, not only by this score but by the previous analysis of their frustration levels. The designers' usability problems should be one of the biggest issues addressed in the future development of the game engine.

Although the system is a tool that helps developers more than it does designers, it would improve even more the projects workflow if designers could easily use the platform with

developers, helping in the integration of the graphical content and even doing minor changes in the game when needed. Most of the problems detected by designers involve the fact that they are not used to work in any type of text editor, not even XML editors, being more used to graphical tools. One good help in this integration is to build a graphical tool for editing the game's XML files.

6.2 The Project's Performance

The project had good performance results in all the tests that were made, leading to the conclusion that system's architecture and implementation are solid. Analyzing in more detail each of the result we can see that although the test results are good, some improvements are still possible in some aspects.

The results of the first test, related to the number of enemies in the screen that the system can support, the minimum value for the frame rate was reached when 22 enemies were on the screen (as shown in Figure 5.6). In a closer analysis of the results we can see that the performance levels start to drop around the 14th enemy on the screen and continue dropping until the minimum value is reached. Given the type of game at hands, 22 enemies at the same time on the screen is much more than a user can handle and even 14 enemies could have a negative impact on the game's gameplay.

Analyzing the results of the second test, this one related to the number of enemies outside the screen, we can see that the minimum frame rate value is reached with 73 enemies in the level. In a more detailed analysis we can see that the game's performance levels started to drop around the 46th enemy added to the game. Although these values are very good, some improvements can be made so that the game engine can perform in optimal condition until approximately 100 enemies in the level, mainly because if a user plans a larger level with alternative paths, the performance of this version of the game engine can be seen as an obstacle.

Analyzing results for tests three and four, we can see that both got very good results. In the case of test three the performance value stayed within optimal values until approximately 480 items where in the screen, where the performance dropped rapidly below the minimum values. In the case of the second test, related to the number of items outside the screen supported by the system, the system worked within normal values until approximately 900 items were added, at which point the system performance has dropped rapidly below the minimum level. Both values are very good and no improving is needed.

6.3 Proof of Concept

The purpose of this section is to present a platform game that has different mechanics and a complex gameplay as a proof of concept for the designed game engine architecture and dialects. Because this tool was created so that FullSix could use it to develop games for their clients, the game that was created is intended for Iglo Olá, a Portuguese ice-cream company that is a client of FullSix.

Iglo Olá is a perfect candidate to build a platform game mainly because it already has a group of heroes and villains, and a great story to work on, associated to their products. Iglo Olá has three brands of ice-creams that target children. They have created a club called Clube Olá that has an online Web site with lots of activities and some flash games. The games that are at the users disposal are mostly small casual games like puzzle and action games, but they do not have any games that are considered a little less casual and that take advantage of the rich story that already exists.

So the goal is to design a platform game that can give the player a great user experience and, at the same time, test if the game engine can support all of the features necessary to develop the game. The target for this game will be the children that use the Clube Olá's Web site.

The following section will describe the game's idea and mechanics, the levels that will make up the game, the enemies and the characters. Finally, an overall description is presented with the key elements and features of the game and how the Platform Games Engine can help to develop the game.

6.3.1 The Game Description

The main plot for the game is as follows:

> "An evil villain called Nodja decided to invade the Olá Planet, home of our 3 heroes. He sent his minions to attack and destroy the towns where our heroes live. The player has to help our heroes in the task of stopping this invasion and save the Olá Planet from the evil Nodja. To achieve this objective, our heroes, after stopping the invasion, will travel to Nodja's home planet and beat him once and for all. To accomplish these objectives, our heroes will have to use the power of the Olá ice-creams that give them the strength they need to defeat Nodja and his troops and be victorious."

This game is composed by a series of four levels. Each of the three first levels is played with each one of the heroes and the final level will be played with all three heroes.

In the following sections each of the three levels will be explained. These sections will approach the key aspects of the level's game design and mechanics as well as some of the aspects of the levels history and how they connect with the defined plot for the game. A full description of this game can be found on appendix D: Olá's Platform Game – Full Description.

6.3.1.1 Level 1 – Nodja comes to Town

This level takes place in *Epá's* home town. The player will play this level as the hero *Epá*. The player's objective is to stop the evil *Nodja's* troops from taking over the town. The level fits in the horizontal and vertical scroller type games, and is divided into two parts, one where the level progresses vertically (where *Epá* has to go down his building) and another one that is horizontal (where *Epá* has to go through the city streets to get to the City Hall).

This level starts at *Epá's* room, and progresses out of his house, and down the stairs of his building to get out on the streets.

Next comes the part on the streets, here the level becomes horizontal, and *Epá* will have to face the enemy troops that are storming the city streets and try to get through to the City Hall, where the level ends.

In this level *Epá* encounters different types of enemies that he can kill by jumping on top of them or by shooting bubble gum. The main mechanics of this level is to scroll through the level jumping on top of the enemies and getting points.

6.3.1.2 Level 2 – Saving the Seas

This level takes place at sea, on *Perna de Pau's* pirate boat. The player will play this level as the hero *Perna de Pau.*

Perna de Pau finds himself on a ship filed with enemy pirates, and what he needs to do is to sink the ship. To do this the player will have to find a special item that gives *Perna de Pau* the strength he needs to sink the ship.

The level is like a maze, where the player has several alternative paths to explore so that he can find the item and complete the level.

6.3.1.3 Level 3 – The Big Rescue

This level takes place at *Super Maxi's* home town. The plot for this level is that all of the dogs in town were kidnapped. The player will have to go and rescue the dogs before time runs out and *Nodja's* ship arrives to take them. This is one of the key gameplay elements of this level.

The level has two parts: the first part is when the player has to run through town to get to the building where the dogs are held; the second is when the player has to jump from window to window to get to the top of the building and rescue the dogs.

The gameplay is like typical platform games here the player jumps on top of enemies to kill them.

6.3.1.4 Level 4 – Visiting Nodja

Finally, after vanquishing *Nodja's* troops the heroes decide to go to *Nodja's* planet to face him. This level has four parts.

The first part is a typical horizontal scrolling level with the difference that the player can chose with of the characters to use at any given time. This creates a very interesting form of gameplay.

The second part is when the player reaches *Nodja* and has to defeat him. This is the first Boss part of this level. The player will have to defeat *Nodja* in his "Big Foot" form.

After the fight *Nodja* runs away and the player follows him. This initiates the third part of the game, were the player has to defeat another Boss form of *Nodja*, the "Rock Monster" form.

When defeated *Nodja* starts to run to his ship to escape, this initiates the last part of this level. This Boss part is a race between the player and the "Bird Monster" form of *Nodja.*

This level has a very complex gameplay structure, filed with different game mechanics to give the user a great game experience.

6.3.2 Game Implementation Problems Overview

This game has a lot of different unique features and aspects that, at first, may seem to be incompatible with the Flash Platform Games Engine. These problems are explained in the next sections as well as their solutions.

The last section gives an overview on the game engine's fitness in implementing this game. In this section, a complete analysis of the detected problems is made as well as some comments and conclusions about the systems architecture and requirements.

6.3.2.1 Boss Type Enemies

The last level is composed by four parts. The first part is like any normal level, where the player scrolls trough the level killing the enemies and getting points. The big difference is when the player has to fight the three bosses. The problem is that the level has to change its mechanics to be compatible with the boss part of the level. The best way to do this is to treat this level as 4 separate levels so that each of the parts can have its own game mechanics. This will also allow for a better management of the system's resources and will make it easier for the developer to implement, not to mention if at a later time changes have to be made to any of the level's parts.

This solves the different level mechanics for each of the bosses, but one problem still remains. The game's dialect, as well as the game engine, does not have any specific type for boss enemies. There are two ways to solve this problem: one is to add a new type to the game dialect for this new type of enemy; the other is to use the EnemyWaypoint type and develop a custom artificial intelligence for each of the boss enemies. For the second choice some minor changes would have to be made so that an AIBot has more possible actions to be compatible with the defined boss mechanics.

6.3.2.2 Shooting Bubble Gum

The Epá's unique feature can be seen has an obstacle to the use of the Flash Platform Games Engine in the development process of this game. The main reason for this is the fact that this character can shoot bubble gum, which is not contemplated in the XML dialect or in the game engine.

The game engine's architecture was built based on the requirement that new features could easily be added at a later time. This character's bubble gum shooting can be seen as one of those features.

The changes needed so that the game engine can support this ability are simple. The first thing is to add this action to the Character's class in the game engine so that it can recognize the new character action, the shooting action. Next the game dialect also has to reflect this change, adding a new possible value for the "action" attribute of the character.

The second part that needs to be setup is the bubble gum bar for this character. This is the easiest part to implement, given that the amount of bubble gum is most like the amount of lives or energy of a character. The next thing to change is to add this attribute to the character's definition in the XML document (an attribute that can be optional and that defined the maximum amount of bubble gum that a character can have, when the attribute is not defined the game engine should ignore its existence).

Finally a new attribute needs to be added so that the game engine knows where to find the graphical representation of the bubble gum shot. This now attribute should be optional, and contain a path to an SWF file where this graphical representation could be found.

6.3.2.3 Level Countdown Timer

This game mechanic can, at first, be seen as a big change to the defined implementation of the game engine. However, looking closer to this problem, we can see that it will only imply little changes in the game engine. The main change will be made in the XML dialect for the levels of the game and it consists only in adding a new attribute to the Level element that tells the user the value of the countdown timer. In the game engine the change should be made to see if the new attribute is defined and, if so, the game engine should save countdown time and use it to initialize the level's timer. Afterwards the update timer function should decrease the timer instead of increasing it.

Although this change consists on the implementation of a new game mechanics the changes needed in the game engine to support it are minimal and easily made.

6.3.2.4 Character Changing during a Level

The last level of this game has a part were the user can change the character he is using in the middle of the game. This feature is probably the most complex feature to implement of all the game because the game engine does not consider the chance that the user could change characters.

Although this feature is complex, the changes needed to support it are relatively simple. The first thing to solve is how to tell the game engine that a level can have multiple characters. To do this no change has to be made to the dialect because the Level element already has an attribute that lets the user define a list of character; the only change needed is to interpret this information in the game engine. Now that the user can define various characters to a level, he needs to be able to know when the player wants to change character. For this, he needs a new action type of event that he can use in the Key element so that tells the game engine to change the character.

The changing a character part is already implemented as two different levels can have different characters defined and the ContentManager and CharacterManager classes already need to load and unload the different character of a game.

6.4 Game Engine Evaluation

Concerning the game that was created to test if the game engine's architecture could solve every type of game mechanic of a 2D platform game (as specified in the projects requirements in chapter 3 - Requirements and Architecture), the results are very good. Although some of the aspects were not possible to implement with the game engine in his current state, the changes that have to be made were very simple and can easily be done. This fact also proves one more requirement, that the platform should easily be expanded and corrected.

Another important point is the fact that apart from the changes to the game engine, most of the work needed to implement this game is creating all the elements that compose it, leaving very little amount of work in the development part of the game, the development work consists mainly in the creation of the game's XML files.

6.5 Overall Satisfaction

A great number of different tests have been made to this platform, from performance tests, to usability tests and even the architecture has been taken to his limit. The results are overall very good and the project meets all the requirements. Nevertheless some tests could have had better scores and so there is still room for improvements. The worst results were in the usability tests made to designers and therefore this aspect is one of the main things that could be improved in this system.

Chapter 7

Future Work

Although all the requirements of the project have been met, some improvements can still be made. One of the most important aspects detected is the usability problems detected in the test with designers. Designers had very high frustration levels and a very low score in the system usability scale (SUS) questionnaire. During the tests some improvements and suggestions were made and as shown in Figure 5.5: Chart for the designer's usability tests suggestions; these suggestions are a very good starting point for the improvement of the usability of the game engine.

Although developers had very good scores, both in the SUS questionnaire and the frustration levels, some suggestions were also made (as shown in Figure 5.3) and should also be taken into account in the next steps for improving this engine.

Another important next step in the development of any game engine is to build a game, and one is already created. As said in section 6.3.2, some new features have to be added to the game engine to be able to run the game fully, but with these changes done and the graphic elements designed the game is ready to play.

In some of the tests, users said that a good way to integrate this platform in the company's workflow would be to create a workshop so that developers and designers could have their first contact with this platform. This is a great idea, not only it guarantees an easier integration of the platform but also it gives the chance for users to learn all the aspects of the platform faster than they would on their own.

This platform solves some of the problems in Web game development, but it is focused on the 2D platform type of games, leaving out all other types of games. A great opportunity for future projects is to try and create a game engine for other types of games, like for example a game engine for Role Playing Games or Puzzle Games.

References

[Ada03] E. Adams and A. Rollings, *Andrew Rollings and Ernest Adams on Game Design*. Berkeley: New Riders Games, 2003.

[Bro] J. Brooke, "SUS - A quick and dirty usability scale".

[Cla07] K. T. Claypoo and M. Claypool, "On frame rate and player performance in first person shooter games," 2007.

[Cru07] A. Crugnola. , Mar. 2007. sephiroth.it. Available at http://www.sephiroth.it/file_detail.php?id=129#, last acceessed on Jun. 2008, 21

[Dic] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games".

[Fai06] T. Faison and E. W. Faison, *Event-based Programming: Taking Events to the Limit*. Apress, 2006.

[Fir07] R. Hauwert. , Oct. 2007. ActionScript Architect. Available at http://www.actionscriptarchitect.com/2007/10/03/first-person-3d-engine/, last acceessed on Jun. 2008, 22

[Ful08] FullSix. FullSix Corporation Web site. Available at http://www.fullsix.com/pt/, last acceessed on Jun. 27, 2008

[Hal02] B. Hall, et al., *Flash Enabled: Flash Design & Development for Devices*. New Riders, 2002.

[Har04] E. R. Harold and W. S. Means, *XML in a Nutshell: A Desktop Quick Reference*. O'Reilly, 2004.

[Lot08] T. Loton, *Introduction to Microsoft Popfly, No Programming Required*. Loton Tech, 2008.

[Mac06] L. A. MacVittie, *XAML in a Nutshell: A Desktop Quick Reference*. O'Reilly, 2006.

[Mic08] Microsoft. , 2008. Microsoft Expression. Available at http://www.microsoft.com/expression/, last acceessed on Jun. 25, 2008

[Mön06] V. Mönkkönen. , Sep. 2006. Gamasutra. Available at http://www.gamasutra.com/features/20060906/monkkonen_01.shtml, last acceessed on Jun. 25, 2008

[Moo03] C. Moock, *ActionScript for Flash MX: The Definitive Guide*. O'Reilly, 2003.

[Moo04] C. Moock, *Essential ActionScript 2.0*. O'Reilly, 2004.

[Moo07] C. Moock, *Essential ActionScript 3.0*. O'Reilly, 2007.

[Mor06] L. Moroney, *Foundations of WPF: An Introduction to Windows Presentation Foundation*. Apress, 2006.

[Nod05] T. Noda and S. Helwig, "Rich Internet Applications -," 2005.

[ORe05] T. O'Reilly, "What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software," 2005.

[ORo04] C. O'Rourke, "A Look at Rich Internet Applications," no. July/August, 2004.

[San07] W. Sanders and C. Cumaranatunge, *ActionScript 3.0 Design Patterns: Object Oriented Programming Techniques*. O'Reilly, 2007.

[Sap07] Sapo. , 2007. Sapo Prémios 2007. Available at http://premios.sapo.pt/winners/, last acceessed on Jun. 28, 2008

[Sol07] G. Solomon and L. Schrum, *Web 2.0: New Tools, New Schools*. ISTE, 2007.

[Spi08] P. Spitzer. Adobe Developer Connection. Available at http://www.adobe.com/devnet/flash/articles/flash_video_3d.html, last acceessed on Jun. 27, 2008

[Vli02] E. v. d. Vlist, *XML Schema: The W3C's Object-Oriented Descriptions for XML*. O'Reilly, 2002.

[Wen07] C. Wenz, *Essential Silverlight*. O'Reilly , 2007.

[Wik08] Wikipedia. , Jun. 2008. Wikipedia. Available at http://en.wikipedia.org/wiki/Sandbox_%28computer_security%29, last acceessed on Jun. 28, 2008

[Wik081] Wikipedia. , Jun. 2008. Wikipedia. Available at http://en.wikipedia.org/wiki/Likert_Scales, last acceessed on Jun. 28, 2008

[Wik082] Wikipedia. , Jun. 2008. Wikipedia. Available at http://en.wikipedia.org/wiki/Silverlight, last acceessed on Jun. 25, 2008

Appendix A

Game XML Dialect Schema Definition

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="Game">
         <xsd:complexType>
               <xsd:sequence>
                     <xsd:element ref="Config" minOccurs="1" maxOccurs="1"/>
                     <xsd:element ref="Levels" minOccurs="1" maxOccurs="1"/>
                     <xsd:element ref="Characters" minOccurs="1" maxOccurs="1"/>
               </xsd:sequence>
               <xsd:attribute name="name" type="xsd:string" use="required"/>
         </xsd:complexType>
   </xsd:element>
```

```
<xsd:element name="Config">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="Keys" minOccurs="1" maxOccurs="1"/>
                  <xsd:element ref="Startup" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Keys">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="Key" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Key">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="Event" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="keycode" type="xsd:string" use="required"/>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Event">
      <xsd:complexType>
            <xsd:attribute name="name" type="xsd:ID" use="required"/>
            <xsd:attribute name="type" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="onKeyDown"/>
                              <xsd:enumeration value="onKeyUp"/>
                              <xsd:enumeration value="isKeyDown"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
```

```
      </xsd:complexType>
</xsd:element>

<xsd:element name="Startup">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="Lives" minOccurs="1" maxOccurs="1">
                        <xsd:complexType>
                              <xsd:attribute name="amout" use="required">
           <xsd:simpleType>
             <xsd:restriction base="xsd:integer">
               <xsd:minInclusive value="0"/>
                                          </xsd:restriction>
                                    </xsd:simpleType>
                              </xsd:attribute>
                        </xsd:complexType>
                  </xsd:element>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Levels">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="Level" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Level">
      <xsd:complexType>
            <xsd:attribute name="definition" type="xsd:anyURI" use="required"/>
            <xsd:attribute name="character" type="xsd:IDREF" use="required"/>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Characters">
      <xsd:complexType>
```

```
            <xsd:sequence>
                  <xsd:element ref="Character" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Character">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="Actions" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
   <xsd:attribute name="name" type="xsd:ID" use="required"/>
   <xsd:attribute name="library" type="xsd:string" use="required"/>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Actions">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="Action" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="Action">
      <xsd:complexType>
            <xsd:attribute name="event" type="xsd:IDREF" use="required"/>
            <xsd:attribute name="frame" use="required">
     <xsd:simpleType>
       <xsd:restriction base="xsd:integer">
         <xsd:minInclusive value="1"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
   <xsd:attribute name="action" use="required">
     <xsd:simpleType>
       <xsd:restriction base="xsd:string">
         <xsd:enumeration value="moveLeft"/>
```

```
            <xsd:enumeration value="moveRight"/>
            <xsd:enumeration value="jump"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
         </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Appendix B

Level XML Dialect Schema Definition

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:element name="Level">
         <xsd:complexType>
               <xsd:sequence>
                     <xsd:element name="Description" minOccurs="0" maxOccurs="1" type="xsd:string"/>
```

```
                  <xsd:element ref="LoadingScreen" minOccurs="0" maxOccurs="1"/>
                  <xsd:choice minOccurs="1" maxOccurs="1">
                        <xsd:element ref="LevelScrolling"/>
                        <xsd:element ref="LevelTiling"/>
                  </xsd:choice>
                  <xsd:element ref="GUI" minOccurs="0" maxOccurs="1"/>
                  <xsd:element ref="WayPoints" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="title" type="xsd:string" use="required"/>
      </xsd:complexType>
</xsd:element>

<xsd:element name="LoadingScreen">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="Title" minOccurs="1" maxOccurs="1" type="xsd:string"/>
                  <xsd:element name="SubTitle" minOccurs="0" maxOccurs="1" type="xsd:string"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

<xsd:element name="LevelScrolling">
      <xsd:complexType>
            <xsd:attribute name="direction" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="horizontal"/>
                              <xsd:enumeration value="vertical"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="navigation" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="left"/>
                              <xsd:enumeration value="right"/>
                              <xsd:enumeration value="both"/>
                        </xsd:restriction>
```

```
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="length" use="required" type="xsd:integer"/>
                <xsd:attribute name="backGroundName" use="required" type="xsd:string"/>
                <xsd:attribute name="backGroundRefName" use="required" type="xsd:string"/>
                <xsd:attribute name="foreGroundName" use="optional" type="xsd:string"/>
                <xsd:attribute name="foreGroundRefName" use="optional" type="xsd:string"/>
                <xsd:attribute name="hitMapRefName" use="required" type="xsd:string"/>
                <xsd:attribute name="softHitMapRefName" use="optional" type="xsd:string"/>
                <xsd:attribute name="levelElementsRef" use="required" type="xsd:string"/>
                <xsd:attribute name="scrollingSpeed" use="optional" type="xsd:integer"/>
        </xsd:complexType>
</xsd:element>

<xsd:element name="LevelTiling">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Tile" minOccurs="1" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="direction" use="required">
      <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                                <xsd:enumeration value="horizontal"/>
                                <xsd:enumeration value="vertical"/>
                        </xsd:restriction>
      </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="navigation" use="required">
      <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                                <xsd:enumeration value="left"/>
                                <xsd:enumeration value="right"/>
                                <xsd:enumeration value="both"/>
                        </xsd:restriction>
      </xsd:simpleType>
                </xsd:attribute>
    <xsd:attribute name="levelElementsRef" use="required" type="xsd:string"/>
        </xsd:complexType>
```

```
 </xsd:element>

<xsd:element name="Tile">
  <xsd:complexType>
    <xsd:attribute name="backGroundName" use="required" type="xsd:string"/>
    <xsd:attribute name="backGroundRefName" use="required" type="xsd:string"/>
    <xsd:attribute name="foreGroundName" use="optional" type="xsd:string"/>
    <xsd:attribute name="foreGroundRefName" use="optional" type="xsd:string"/>
    <xsd:attribute name="hitMapRefName" use="required" type="xsd:string"/>
    <xsd:attribute name="softHitMapRefName" use="optional" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GUI">
  <xsd:complexType>
    <xsd:attribute name="name" use="required" type="xsd:string"/>
    <xsd:attribute name="library" use="required" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

 <xsd:element name="WayPoints">
       <xsd:complexType>
             <xsd:sequence>
                   <xsd:element ref="LevelStartingPoint" minOccurs="1" maxOccurs="1"/>
                   <xsd:choice minOccurs="1" maxOccurs="unbounded">
                         <xsd:element ref="ObjectWayPoint"/>
                         <xsd:element ref="EnemyWayPoint"/>
                         <xsd:element ref="ItemWayPoint"/>
                         <xsd:element ref="EnemySpawnPointWayPoint"/>
                   </xsd:choice>
                   <xsd:element ref="LevelEndPoint" minOccurs="1" maxOccurs="1"/>
             </xsd:sequence>
       </xsd:complexType>
 </xsd:element>

<xsd:element name="LevelStartingPoint">
  <xsd:complexType>
    <xsd:attribute name="name" use="required" type="xsd:ID"/>
```

```
    <xsd:attribute name="library" use="optional" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="LevelEndPoint">
  <xsd:complexType>
    <xsd:attribute name="name" use="required" type="xsd:ID"/>
    <xsd:attribute name="library" use="required" type="xsd:string"/>
    <xsd:attribute name="visibility" use="required" type="xsd:boolean"/>
  </xsd:complexType>
</xsd:element>
 <xsd:element name="ObjectWayPoint">
       <xsd:complexType>
             <xsd:attribute name="name" use="required" type="xsd:ID"/>
             <xsd:attribute name="library" use="required" type="xsd:string"/>
             <xsd:attribute name="destructible" use="optional" type="xsd:boolean"/>
             <xsd:attribute name="pickable" use="optional" type="xsd:boolean"/>
       </xsd:complexType>
 </xsd:element>
 <xsd:element name="EnemyWayPoint">
       <xsd:complexType>
             <xsd:attribute name="name" use="required" type="xsd:ID"/>
             <xsd:attribute name="library" use="required" type="xsd:string"/>
             <xsd:attribute name="score" use="required" type="xsd:integer"/>
       </xsd:complexType>
 </xsd:element>
 <xsd:element name="ItemWayPoint">
       <xsd:complexType>
             <xsd:attribute name="name" use="required" type="xsd:ID"/>
             <xsd:attribute name="library" use="required" type="xsd:string"/>
             <xsd:attribute name="score" use="required" type="xsd:integer"/>
             <xsd:attribute name="lives" use="optional" type="xsd:integer"/>
             <xsd:attribute name="event" use="optional" type="xsd:string"/>
       </xsd:complexType>
 </xsd:element>
 <xsd:element name="EnemySpawnPointWayPoint">
       <xsd:complexType>
             <xsd:attribute name="name" use="required" type="xsd:ID"/>
```

```
            <xsd:attribute name="library" use="optional" type="xsd:string"/>
            <xsd:attribute name="spawnInterval" use="required" type="xsd:integer"/>
            <xsd:attribute name="enemyRef" use="required" type="xsd:string"/>
            <xsd:attribute name="score" use="optional" type="xsd:integer"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Appendix C

The Elements SWF File Description

One of the key parts of the game engine elements is the Elements SWF file. This file defines the positions of each of the level's waypoints. The main reason for the positioning of the waypoints to be made using this file is that it is easier to alter and visualize the game if using a visual tool like Flash. In the following sections the file's architecture is explained.

C.1 The File's Architecture

There are no major restrictions in the files architecture. The only restriction is that all the waypoints need to be defined in the root element of the file so that the game engine can find it. Also when creating the Elements SWF file the developer has to use the same names for the waypoints has defined in the level's XML file.

Each waypoint consists of an instance of a MovieClip Flash element. This element should be empty, although the game engine empties it when loading the element relative to that waypoint the performance of the game engine could be altered because the file size of the Elements SWF file will be unnecessarily large.

Another point is that although the game engine does not use the layer structure of the file this structure can be very helpful when the developer needs to change any aspect of the level. One suggestion is to create layers for each type of waypoints so that they could be hidden individually at any time. One example of this structure can be found in Figure G.1.

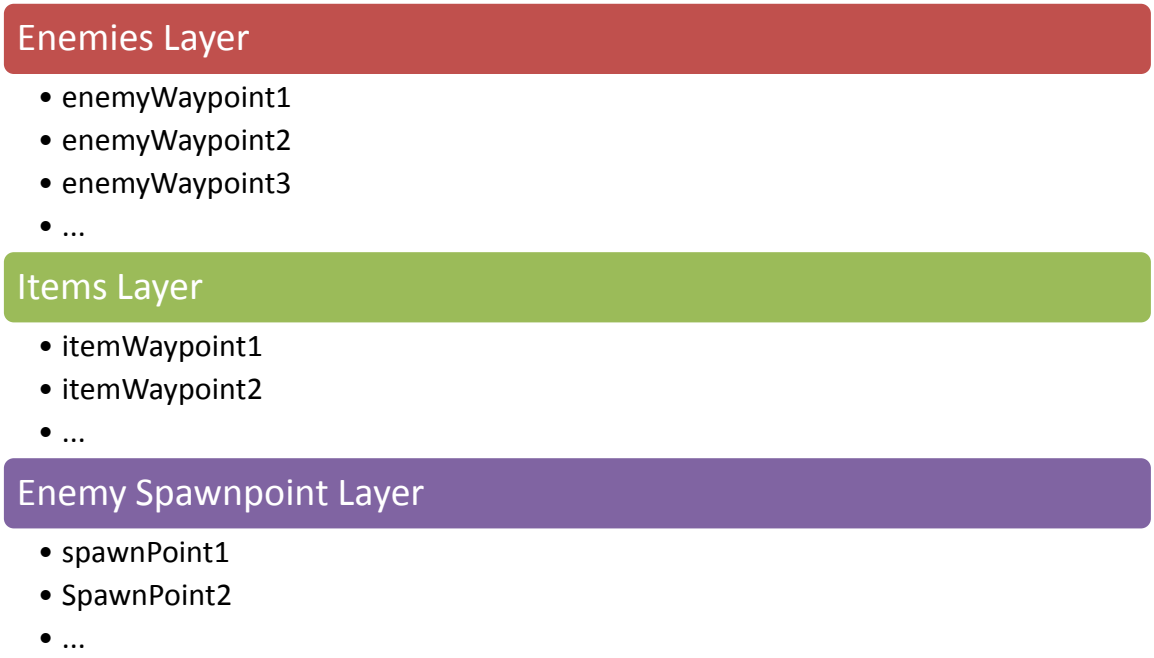


Figure G.0.1: Diagram of the Layer Structure of an example level Element SWF file.

In this example the level is composed by a number of waypoints that are divided into three layers. Each layer contains one of the types of waypoints, creating an organized and structured definition of the level. With this file architecture it is easier to find a waypoint when needed.

Appendix D

Olá's Platform Game – Full Description

In the next sections a full description of the game is made. This description includes the description of the game's levels, mechanics, and characters. In a brief sentence the game idea is:

> "Three very different friends have to join their forces to defeat a common enemy that is attacking their world."

All of the game mechanics and characters are based on the existing history from Clube Olá.

D.1 The Game's Levels

An evil villain called *Nodja* decided to invade the Olá Planet, home of our 3 heroes. He sent his minions to attack and destroy the towns where our heroes live. The player has to help our heroes in the task of stopping this invasion and save the Olá Planet from the evil *Nodja*. To achieve this objective, our heroes, after stopping the invasion, will travel to *Nodja's* home planet and fight him once and for all. To do all this, our heroes will have to use the power of the Olá ice-creams that give them the strength they need to defeat *Nodja* and his troops and be victorious.

The game is divided into 4 chapters, each of them taking place in the home town of each hero and one, the last, in *Nodja's* world.

D.1.1 Level 1 - "*Nodja* comes to Town"

This level takes place in *Epá's* home town. The player will play this chapter as the hero *Epá*. The player's objective is to stop the evil *Nodja's* troops from taking over the town. The streets are filled

with evil troops that are spreading chaos. The only way to stop them is to reach the city hall and activate the milk sprinklers that will wash away all the troops and clean up the city.

This level fits in the horizontal and vertical scroller type games (like Super Mario), and is divided into two parts, one where the level progresses vertically (where *Epá* has to go down his building) and another one that is horizontal (where *Epá* has to go through the city streets to get to the City Hall).

This level starts at *Epá's* room, where some of *Nodja's* troops are attacking him. *Epá* has to go out of the house, and down the stairs of his building to get out on the streets and stop this attack. When *Epá* is going down his building, he can sometimes enter to the apartments of his neighbors and inside he can find more troops to eliminate, some ice-creams to bust his score and even, sometime, rescue his neighbors and get some extra points.

Next comes the part on the streets, here the level becomes horizontal, and *Epá* will have to face the enemy troops that are storming the city streets and try to get through to the City Hall. In this part *Epá* will be moving through the streets of the city, fighting *Nodja's* troops, jumping on top of balconies and trying to collect as much ice-creams as he can. He will also be able to enter some of the houses by pressing up next to the door. When inside a house *Epá* can find items and some troops to fight and get some extra points. *Epá* will also be able to go down some man-holes and into the sewers where he will find great items that are very well guarded by *Nodja's* troops. At the end of the level *Epá* will arrive at City Hall were the level ends.

From a graphical point of view, this level has two parts. The first part is indoors, inside *Epá's* building. Here the level will have the aspect of an apartment building, with a stairway in the middle and apartments on each side. The stairway should be big, so that there can be troops and *Epá* has room to fight them. The second part is outdoors, on the city streets. In this part the game is very rich in graphic details. The player should experience various types of backgrounds as he progresses through the city. First he is in the building complexes where *Epá* lives. Then he will exit this part of the city and be in the city park. After the park *Epá* will cross a long bridge leading to the center of the city. In the city center the scenery is of chaos, being this part the most affected by the attack. At the end of the level *Epá* will find the City Hall where he will have to enter to end the level and save the city.

D.1.2 Level 2 - "Saving the Seas"

This level takes place in *Perna de Pau's* home world, the sea. This hero lives in his pirate boat and patrols the coasts of Olá Planet. Player will help *Perna de Pau* to stop *Nodja's* troops from reaching the coasts, sinking his master boat. To do this, the player has to board the villain's boat and find a *Mega Perna de Pau* ice-cream, that will give our hero the strength he needs to sink the boat, after which he has to escape as fast has he can so that he does not sink with it.

This level takes place on a single boat and is mainly made of a large maze below the deck of the boat. *Perna de Pau* will start the level on the ship deck and will make his way through the maze. To navigate the maze, *Perna de Pau* will have to use rope stairs to go up and down the different levels of the maze. He will also have to use barrels that he can throw to open some doors that block his way. The maze will be filled with enemies and *Perna de Pau* will have to fight his way through them to reach the "Mega Perna de Pau" ice-cream and sink the boat. After this *Perna de Pau* will have to get off the boat as fast has he can. To do this he will have to get on the deck of the boat again where he will find a large catapult, he will have to jump on top of the catapult to end the level and get off the boat.

From a graphical point of view this level will look like a medieval boat, with wood floors and walls, and wooden tables and barrels. The challenge in the creation of this level is the balancing of the maze. It is very important that the player can have time to exit the boat as it sinks.

D.1.3 Level 3 - "The Big Rescue"

This level takes place in the dog hero *Super Maxi's* home town. All of the dogs in this town have been kidnapped and are on top of the city's main tower, where *Nodja's* minions are preparing to take them to Nodja's planet. Our hero has to be fast and rescue the dogs from *Nodja's* troops, before *Nodja's* ship arrives and takes them**.**

One of the challenges of this level is the fact that the screen will scroll automatically, forcing *Super Maxi* to advance in the level. The player will also find large jumps that he will have to do using a combination of sprint (to pick-up speed) and jump so that *Super Maxi* can jump farther.

This level is composed by two parts. The first one takes place in the woods next to town, where *Super Maxi* was asleep when *Nodja's* troops surprised him. In this first part *Super Maxi* has to run through the woods to get back to town. Through the way he will have some obstacles to overcome, like falling trees he will have to avoid, holes in the ground he will have to jump over, and big lakes he will have to jump from tree trunk to tree trunk to come across. The second part of this level starts when the *Super Maxi* reaches the building were his dog friends are held. Now comes the second part of this level. In this part the game turns vertical, and the player has to jump from window to window, as the screen moves up automatically. In this part the challenge is, not to fall, as *Super Maxi* jumps from balcony to balcony, getting higher and higher in the building. There are also some objects that are being thrown down from the roof by *Nodja's* troops. Sometime they throw ice-creams by mistake, so *Super Maxi* has to avoid the objects but get the ice-creams to get points. They also have troops that bungee jump from the roof and try to hit *Super Maxi.*

D.1.4 Level 4 - "Visiting Nodja"

After saving their home towns, our heroes decide to join forces and travel to *Nodja's* world and defeat the villain once and for all. So this chapter takes place in *Nodja's* world, and the player will have to play with each of the heroes to overcome all the challenges of this chapter, taking advantage of each hero's unique features. After overcoming all the challenges our heroes reach *Nodja's* castle, where they have to defeat the evil *Nodja* in a last big, heroic battle.

This level is divided into two parts, the first one is composed by a series of challenges where the player will be introduced to the hero changing mechanic, and solve a series of challenges. The player will have a key assigned to change the current player, when the player presses that key, the current hero will flip to the background and the next hero will come to the foreground using some kind of 2D animation. The interface should also show the order in which the change will occur.

The first part of this level will take place outside *Nodja's* Castle. This part will be divided into two parts, the first one takes place where our heroes have landed, in a wasteland. In this wasteland the player will encounter some canyons that he will have to jump across (using "Super Maxi") and some large boulders blocking the passage that he will have to destroy (using *Perna de Pau's* strength to throw rocks at it). Next the player will enter the forest that surrounds the castle were they will face some new challenges. This forest is a dark and evil forest that the player will have to pass. In this forest the trees in the background are alive and are constantly throwing objects at our heroes, these objects are falling from above and will damage the hero if they hit him. This will force the player to keep moving. In this forest there will be some object that the player can pick-up (using *Perna de Pau*) and some Bubble Gum items to pick-up (by *Epá*), but there aren't any

specific obstacles to overcome, the challenge is to decide the best hero to play with on each instant taking advantage of the items available and the skills of each of them. At the end of the forest the player will arrive at *Nodja's* Castle, where he will have to pick-up a "Mega Perna de Pau" ice-cream to destroy the castle's gate and enter.

Inside the castle our heroes will be welcomed by *Nodja* that transforms into a big monster with big feet and tries to steep on them. To defeat him the player will have to use *Epá's* bubble gum to make the monster fall and then jump on him to damage him. When defeated *Nodja* will run to a next area and transform into a big rock monster. To defeat this monster the player will need *Perna de Pau's* strength, using the rocks that are on the ground to throw at the monster, after some hits the monster's belly will crack and a "Mega Perna de Pau" ice-cream will fall and *Perna de Pau* can use it to damage *Nodja*. The player will have to repeat these steps some times to defeat the monster. When *Nodja's* defeated he will start to run, transforming into a big bird. The player will have to use "Super Maxi" to sprint trough the level and get to the end before *Nodja*. Throughout the level there will be obstacles to overcome (like objects to jump over, canyons to jump across and some falling rocks that *Nodja* will throw to damage "Super Maxi"). The player will also encounter some enemies in this race that if he jumps on top of them they will transform into little bombs that will hit *Nodja* and slow him down.

At the end of the level, the player will beat *Nodja* in his race to escape and reach the hangar, but he will find that *Nodja's* ship is not there. *Nodja will make* its way to is space ship that has hidden outside the hangar and escape, but our heroes will be happy because they defeated the evil *Nodja*.

D.2 The Game's Characters

In the following sections all of the characters of the game are described in detail. The goal of this detailed description is to guide developers and designer of the game when creating each character.

D.2.1 The Hero Characters

In the next sections a complete descriptions of the player characters is made.

D.2.1.1 Epá

Name:	Epá
Type:	Hero
Personality	He's a very cool kid, but also keeps everything to himself.
Behavior:	He blushes every time he gets a compliment, but also feels very proud of himself.
Unique Features:	He has the ability to shoot bubblegum balls at his enemies to defeat them.
Inventory:	He has an inventory bar that represents the amount of bubblegum he has which he can use to shoot at enemies.

D.2.1.2 Perna de Pau

Name:	Perna de Pau
Type:	Hero
Personality	He's a very happy and optimistic pirate. He also is a little brute. He always craves for adventure.
Behavior:	He's so strong that when he compliments anyone all flee because when he

	taps someone in the back he can really hurt them.
Unique Features:	He's able to pick-up heavy objects and throws them at his enemies.
Inventory:	He does not have any

D.2.1.3 Super Maxi

Name:	Super Maxi
Type:	Hero
Personality	He’s a loyal but hard headed dog.
Behavior:	He's all fun and games.
Unique Features:	He’s able to run really fast and jump long distances.
Inventory:	He does not have any.

D.2.2 The Enemy Characters

In the next section a complete description of the enemy characters is made. These enemies can appear in different levels of the game.

D.2.2.1 Nodja Troop – Normal Troop

Name:	Grunt
Type:	normal enemy
Appears:	Grunt can appear in any level.
Personality	This troop is evil, he has that killing eye and that army grunt expression.
Description	He’s a short stature troop, he does not carry any weapon and he wears his combat hat always.
Behavior:	He only moves left and right. He is not very bright so he moves until he hits a wall and then he comes back and repeats this behavior.
Unique Features:	This troop does not have any Unique Feature.
Weaknesses:	This troop is vulnerable to any type of attack.
Inventory:	He does not have any.

D.2.2.2 Nodja Troop – Scuba Troop

Name:	Scuba Trooper
Type:	normal enemy
Appears:	Scuba Trooper can appear on any level when the player is passing over water (in the bridge section of level 1, and on the woods of levels 3 and 4).
Personality	Scuba Trooper is a very sneaky unit. He links to appear out of the water when no one is expecting and shoot his harpoon at his enemies.
Description	He wears his underwater goggles and his breathing tube and he carries around is harpoon weapon to shoot at his enemies.
Behavior:	He jumps out of the water and in front of his enemies when they get near and shoots them with his harpoon.
Unique Features:	He has the unique feature of jumping out of the water and surprising his enemies.
Weaknesses:	This troop is vulnerable to any type of attack.
Inventory:	He does not have any.

D.2.2.3 Nodja Troop – Mercenary Troop

Name:	Merc
Type:	normal enemy
Appears:	Merc can appear in any level.

Personality	Merc is a mercenary that works for *Nodja*. He takes great pleasure in creating chaos and devastation.
Description	Merc is very strong, as can be seen by his muscled body. He walks around carrying a combat helmet and his belt of bullets for his M16 rifle.
Behavior:	Merc just moves around the level and when an enemy gets near him he starts to shoot his gun. Because of his size, he's a very slow moving character.
Unique Features:	Merc's unique feature is his strength that makes him immune to damage from bubble gum balls and objects that are thrown at him. He also is the only one that uses a M16 rifle.
Weaknesses:	Because of his strength this unit takes no damage from being hit with objects and bubble gum, but he's a ranged unit so if a hero gets close to him and jumps on top of him he will take damage.
Inventory:	He does not have any.

D.2.2.4 Nodja Troop – Sword Pirate

Name:	Sword Pirate
Type:	normal enemy
Appears:	Sword Pirate appears on level 2.
Personality	Sword Pirate is a real pirate, as bad as pirate comes; at least that is what he wants people to see in him. Although he does not need it, he uses an eye patch to give him that pirate look.
Description	Sword Pirate wears a red and white tissue on his head and an eye patch on his left eye. He also walks around with his pirate sword.
Behavior:	Sword Pirate walks around below the deck of the boat. Because of his eye patch he cannot see very well so he swings around his sword sometimes.
Unique Features:	Sword Pirate is a unique swordsman; the only problem is that eye patch he wears...
Weaknesses:	Because Sword Pirate swings his sword at random, it is dangerous to engage him in close combat, so throwing objects at him is the best way to damage him, but he can also be damaged by jumping on top of him.
Inventory:	He does not have any.

D.2.2.5 Nodja Troop – Gun Pirate

Name:	Pirate Captain
Type:	normal enemy
Appears:	Sword Pirate appears on level 2.
Personality	He's a very calm captain, always walking around with a superiority look on his face.
Description	Pirate Captain is a very particular pirate; he wears a captain's hat and walks around with his pirate gun.
Behavior:	As a captain he walks around paroling the decks and sometimes he likes to shoot his gun just for the fun of it.
Unique Features:	Pirate Captain is the only pirate that carries a gun.
Weaknesses:	Pirate Captain takes damage from being hit by objects and by jumping on top of him.
Inventory:	He does not have any.

D.2.2.6 Nodja Troop – Robin

Name:	Robin
Type:	normal enemy

Appears:	Robin appears in the woods parts of the levels (namely in the level 1, 3 and 4).
Personality	He's a very quiet guy. He does not like people in his woods so he shoots arrows at anyone that crosses his woods.
Description	Robin uses a green cap (like Robin Hood) and a bow to shoot arrows at his enemies.
Behavior:	Robin patrols the woods attacking every one that enters. When he sees a target he stops and takes aim, then he shoots an arrow at his target.
Unique Features:	Robin is very good with his bow.
Weaknesses:	As a ranged unit Robin is vulnerable to close combat attacks, so the best way to damage him is to get close to him and jump on top of him.
Inventory:	He does not have any.

D.2.2.7 Nodja Troop – Bungee Jumper

Name:	Bungee
Type:	normal enemy
Appears:	Bungee appears in level 3 in the part were *Super Maxi* is climbing the building.
Personality	He's a radical troop that loves to fall from buildings with is bungee rope.
Description	Bungee wears a kamikaze bandana on his head and has a bungee rope tied to his feet.
Behavior:	He drops from the top of the building and tries to punch *Super Maxi* on his way down.
Unique Features:	Bungee's unique feature is the fact that he can bungee jump from buildings.
Weaknesses:	He's immune to all kinds of attacks, so the best thing is to avoid it.
Inventory:	He does not have any.

D.2.2.8 Nodja Troop – Parachute Troop

Name:	Parachuter
Type:	normal enemy
Appears:	Parachuter appears on the wasteland part of the final level.
Personality	This troop loves to fly, and he likes to surprise his enemies by dropping from the sky.
Description	He wears an airman's hat on his head and uses a small gun to shoot his enemies.
Behavior:	He drops from the sky with his parachute then attacks his enemies with his gun.
Unique Features:	This troop has the ability to drop from the sky with his parachute.
Weaknesses:	This troop is vulnerable to any type of attack.
Inventory:	He does not have any.

D.2.2.9 Nodja Troop – Pumpkin-head Troop

Name:	Pumpkin-Head
Type:	normal enemy
Appears:	Pumpkin-Head appears on the forest of the final level.
Personality	This troop likes to scare his enemies. He thinks that every day is Halloween.
Description	Pumpkin-Head uses a Halloween pumpkin on his head to scare his enemies.
Behavior:	This troop only walks around with a pumpkin on his head, when he gets hit the pumpkin breaks and he starts to run really fast.
Unique Features:	The unique feature of this troop is the fact that he wears a pumpkin on his head.
Weaknesses:	This unit is vulnerable to any type of attack, but the first hit he gets only

	breaks his pumpkin head, and makes him run faster, he needs another hit to be killed.
Inventory:	He does not have any.

D.2.2.10 Nodja Troop – Rocket Pack Troop

Name:	Rocket Man
Type:	normal enemy
Appears:	Rocket Man appears only on the final part of the final level, when the player has to race *Nodja* to his hangar, to stop him from escaping.
Personality	Rocket Man is totally crazy, he loves to fly as long has he can, in the end of his flight, do a crash landing.
Description	He wears the rockets on his back and a bullet helmet on his head.
Behavior:	Rocket Man runs through the level in the opposite direction of the hero, trying to hit him. When the hero can successfully jump on top of him his back pack will activate and he will start to fly and if he hits *Big Bird Nodja* he will slow him down.
Unique Features:	This troop's unique feature is the fact that the hero's only way to slow down *Big Bird Nodja* is to use these troops to hit him.
Weaknesses:	The only way to damage Rocket Man is to hit him on the head.
Inventory:	He does not have any.

D.2.3 The Boss Characters

In the next sections a complete description of the game's boss characters is made. These bosses appear in the last part of the fourth level.

D.2.3.1 Nodja – Big Foot Monster

Name:	Big Foot Nodja
Type:	Boss
Appears:	Big Foot Nodja appears in part 1 of the final Boss.
Personality:	He loves to steep on people.
Behavior:	This is one of Nodja's transformations. In this transformation, Nodja becomes a very big monster with large feet that looks a lot like a prehistoric man. This monster is not very bright, and his behavior reflects that. This monster moves right and left, trying to crush everything and everyone in is way.
Unique Features:	This monster's unique feature is the fact that he's huge and has very large feet that crush everything that he steps on.
Weaknesses:	Beneath all of that strength and size there's always a soft spot, in this monster's case is his belly. If anyone jumps on his belly he will take damage, this is why he always sleeps standing and never lies down on the ground, except when he falls… When he gets bubble gum on his feet he sometimes gets stuck on the ground and falls, making his belly exposed to attacks.
Inventory:	He does not have any.

D.2.3.2 Nodja – Rock Monster

Name:	Hard Rock Nodja
Type:	Boss
Appears:	Hard Rock Nodja appears in part 2 of the final Boss.
Personality:	He's a giant brute that likes to crush and destroy.
Behavior:	He likes to throw big rocks at people and jump very hard on the ground making it shake and everyone dizzy .He's very big and very heavy so he does not move a lot, but he's also very strong so he can jump from a place to another and when he lands he causes the ground to shake.
Unique Features:	Hard Rock Nodja's unique features are his unlimited supply of big boulders he has on his pocket and his ability to make the ground shake when he jumps and make everyone that is on the ground dizzied from the shock.
Weaknesses:	Beneath all that rock body there are hidden some *Mega Perna de Pau* ice-creams. If he gets hit by rocks these ice-creams may fall and if *Perna de Pau* gets his hands on them he will have the strength to break that rock body.
Inventory:	He does not have any.

D.2.3.3 Nodja – Bird Monster

Name:	Big Bird Nodja
Type:	Boss
Appears:	Big Bird Nodja appears in part 3 of the final Boss.
Personality:	This bird is a transformation that *Nodja* uses to escape when he's defeated. He's a frighten bird that wants to escape at all cost.
Behavior:	This bird flies at high speed through the level. As he flies he sometimes throws objects to damage our hero that is trying to get him.
Unique Features:	This bird can fly very fast and has the ability to throw objects down as he flies.
Weaknesses:	This bird is always flying so there is no direct way to hit him, the only way to stop him is to reach Nodja's hangar first so that he cannot escape. Although there is no way to damage this bird there are ways to slow him down, using the little troops that have rocket packs on their backs, if a hero jumps on top of these troops it will activate the rocket packs and they will go up and hit the bird, slowing him down.
Inventory:	He does not have any.

D.3 The Game's Items

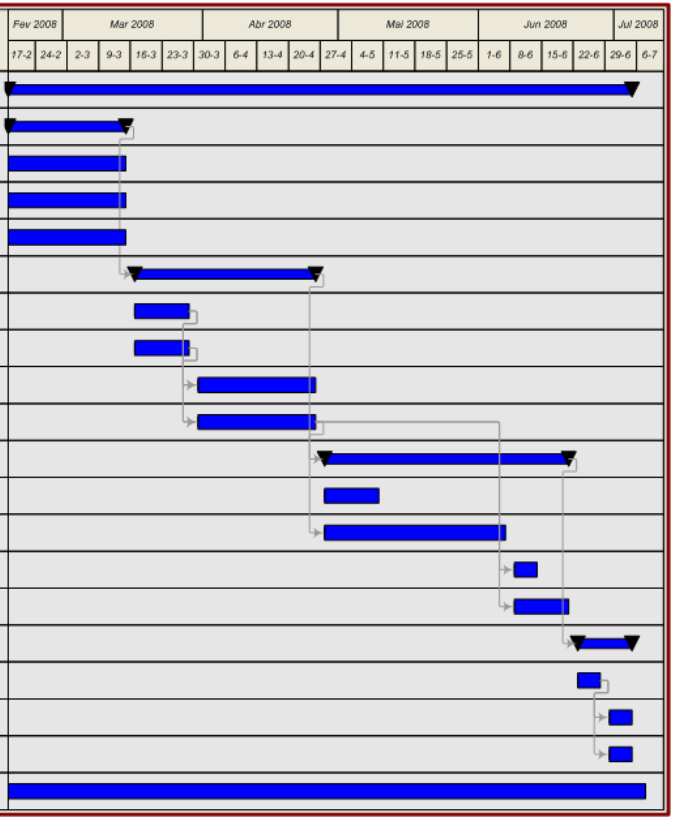
Along the game the player will have the chance to pickup different items that give him points or power-ups. These items are:

- **Bubblegum Balls**: This item refills the bubblegum bar of the hero *Epá*.
- **Epá Ice-cream**: This item refills the life bar of the hero *Epá*.
- **Perna de Pau Ice-cream**: This item refills the life bar of the hero *Perna de Pau*.
- **Super Maxi Ice-cream**: This item refills the life bar of the hero "Super Maxi".
- **Mega Perna de Pau Ice-cream**: This ice-cream makes *Perna de Pau* even stronger. This item can be found in the level 2, somewhere in the maze. It is also dropped by Hard Rock Nodja when he's hit.

Appendix E

Project's Planning – Gant Diagram

ID	Task Name	Start	Finish	Duration
1	**Project's Development**	**18-02-2008**	**04-07-2008**	**100d**
2	**Research**	**18-02-2008**	**14-03-2008**	**20d**
3	Literature Research about Web-Development Platforms	18-02-2008	14-03-2008	20d
4	Literature Research about Game Design and Gameplay	18-02-2008	14-03-2008	20d
5	Research on Game Development Tools and Frameworks	18-02-2008	14-03-2008	20d
6	**Project's Requirement and Architecture Analysis**	**17-03-2008**	**25-04-2008**	**30d**
7	Definition of Project's Requirements	17-03-2008	28-03-2008	10d
8	Definition of Project's Architecture	17-03-2008	28-03-2008	10d
9	Definition of a Complex Platform Game to serve as Proof of concept for the Game Engine	31-03-2008	25-04-2008	20d
10	Prototype implementation	31-03-2008	25-04-2008	20d
11	**Project Implementation and Testing**	**28-04-2008**	**20-06-2008**	**40d**
12	Test definition	28-04-2008	09-05-2008	10d
13	Product implementation	28-04-2008	06-06-2008	30d
14	Usability Testing	09-06-2008	13-06-2008	5d
15	Performance Testing	09-06-2008	20-06-2008	10d
16	**Project completion**	**23-06-2008**	**04-07-2008**	**10d**
17	Implementation of Test Level of designed Game as Proof of Concept	23-06-2008	27-06-2008	5d
18	Final Testing	30-06-2008	04-07-2008	5d
19	Overall Maintenaince and Corrections	30-06-2008	04-07-2008	5d
20	Project's Report	18-02-2008	07-07-2008	101d



Appendix F

XML Game File and Level File Example

```
<?xml version="1.0"?>
<Game name="Jogo de Teste">
  <Config>
    <Keys>
      <Key keycode="37">
        <Event name="moveLeft" type="isKeyDown"/>
      </Key>
      <Key keycode="39">
        <Event name="moveRight" type="isKeyDown"/>
      </Key>
      <Key keycode="38">
        <Event name="jump" type="onKeyDown"/>
      </Key>
    </Keys>
    <Startup>
      <Lives amout="3"/>
    </Startup>
  </Config>
  <Levels>
```

```
    <Level definition="level1.xml" character="hero1"/>
  </Levels>
  <Characters>
    <Character name="hero1" library="Libs\player.swf">
      <Actions>
        <Action event="moveLeft" frame="3" action="moveLeft"/>
        <Action event="moveRight" frame="4" action="moveRight"/>
        <Action event="jump" frame="5" action="jump"/>
      </Actions>
    </Character>
  </Characters>
</Game>
```

```
<?xml version="1.0"?>
<Level title="Level 1">
  <Description>
    This is the first level of the prototype.
  </Description>
  <LevelScrolling levelElementsRef="Libs\elements1.swf" direction="horizontal" lenght="3000" navigation="right"
scrollingSpeed="2"  backGroundName="background" backGroundRefName="Libs\background1.swf" foreGroundName="foreground"
foreGroundRefName="Libs\foreground1.swf" hitMapRefName="Libs\hitmap1.swf"/>
  <GUI name="gui" library="Libs\gui.swf"/>
  <WayPoints>
    <LevelStartingPoint name="start"/>

    <EnemyWayPoint name="enemy1" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy2" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy3" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy4" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy5" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy6" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy7" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy8" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy9" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy10" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy11" library="Libs\enemy1.swf" score="100"/>
    <EnemyWayPoint name="enemy12" library="Libs\enemy1.swf" score="100"/>
      <EnemyWayPoint name="enemy13" library="Libs\enemy1.swf" score="100"/>
    <ItemWayPoint name="item1" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item2" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item3" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item4" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item5" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item6" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item7" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item8" library="Libs\item1.swf" score="100"/>
    <ItemWayPoint name="item9" library="Libs\item1.swf" score="100"/>
    <EnemySpawnPointWayPoint name="spawn1" enemyRef="Libs\enemy1.swf" library="Libs\portal.swf"
spawnInterval="5000"/>
    <EnemySpawnPointWayPoint name="spawn2" enemyRef="Libs\enemy1.swf" library="Libs\portal.swf"
spawnInterval="5000"/>
```

```
  <LevelEndPoint name="end" library="Libs\endPoint.swf" visibility="true"/>
</WayPoints>
 </Level>
```

Index

W

X