BACHELOR THESIS

Design and Implementation of an Educational Game Control Module Through Microsoft Kinect

Autor: Antón García Dosil

Tutor: Telmo Zarraonandia Ayo

Leganés, Julio de 2013

Título: Design and Implementation of an Educational Game Control Module Through Microsoft Kinect

Autor: Antón García Dosil

Director: Telmo Zarraonandia Ayo

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Special Thanks

I would like to thank everyone that has been involved in my four years at University Carlos III. I leave Leganés with not only an accreditation but also with great friends met along the way.

Special thanks to Telmo for guiding me the whole way of this project, using even weekends for revision.

I would also like to thank my family for their support, without it would have been impossible to study away from my hometown.

Resumen

Este proyecto describe el diseño e implementación de un Módulo de Control para Juegos Educativos utilizando un sensor de movimiento. La solución pretende proporcionar una plataforma customizable para que educadores puedan diseñar juegos controlados con gestos.

Para desarrollar el proyecto las últimas tendencias en Interacción Humano-Máquina y detección de movimiento han sido analizadas a fondo.

Palabras clave: videojuegos, educación, customizable, aprendizaje con videojuegos, Kinect, Kinect for Windows, GREM, GREP, Interacción Humano-Máquina, HCI.

Abstract

This Project describes the design and implementation of an Educational Game Control Module using a motion detection device. The solution is focused on providing a customizable platform for educators to design games that may be controlled through gestures.

In order to develop the project the latest trends on Human-Computer interaction and motion detection have been analyzed thoroughly.

Keywords: videogames, education, customizable, videogame based learning, Kinect, Kinect for Windows, GREM, GREP, Human Computer Interaction, HCI

Content

Index of Figures

Index of Tables

1. Introduction

The objective of this chapter is to give a global view of the project, and to present the focus and goals that have motivated the study and implementation of the solution provided. A summarized overview on each section of this report will be given in order to make the lecture clear and easy to read.

1.1. Context

In this section a general view of the current context related to both videogames and Human Computer Interaction is given, providing a brief introduction to each one of these areas.

1.1.1. Videogames

The first videogames appeared after the development of programmable supercomputers in World War II and evolved at a slow pace until the 70s, when the game development industry suffered a considerable boom due to technological improvements. Since the 70s, the only limitations for videogame production were creativity and the technology available for development. Nowadays this sector of the entertainment industry comprises one of the most important of the whole area, providing a total income of $67 billion in year 2012, with a predicted growth to up to $82 billion by year 2017 [1]. This unstoppable growth since the 70s makes the videogame development field interesting from a professional perspective due to the long future ahead of videogame development. The impact of videogame development is evident in all fields of Computer Science, from aspects such as powerful new GPU based programming languages such as CUDA, to artificial intelligence algorithms or design of new hardware in order to achieve higher computational power and graphic rendering. All this investigation has been incremented by the huge competitiveness in platform developers (mostly Nintendo, Sony and Microsoft) as well as game developers (Ubisoft, Electronic Arts, Activision…).

Figure 1: Sega Duck Hunt (1969)

The interest of industry for videogames is evident but: is there any use for videogames in addition to the obvious entertainment aspect? No doubt – Education. The possibilities of using videogames for educational purposes has intrigued the

academic community and spawned many papers on the area (Squire, K. 2003 [2], Papastergiou, M. 2009 [3], Prensky, M. 2003 [4], etc.). As we may observe, formal academic studies on educational applications for videogames are all quite recent with respect to the start of videogame mass production (around the 70s). The interest of the videogame industry for educational games arose in the 80s suffering a decay in the 21st century due to the change in the market. In the 80s and 90s most of the videogame players were children or youth, but this fact changed in the 21st century (those children grew up) having a much larger market for adults. According to the ESA (Entertainment Software Association), nowadays the average gamer age is 37 years [5]. This market shift made industry focus on developing videogames for a wider age span, leaving educational games at one side. This loss of interest by industry does not imply that educational games are not required, as children have obviously not disappeared in the last decade. This is a clear signal that organizations not focused only on profit should fill this area and the academic community should work more now than ever on providing the best teaching solutions possible for their students. This project is aimed towards providing educators a tool to design their own videogames according to their personal teaching purpose, minimizing the technical skills and support required for it.

Figure 2: Educational game - Where in the World is Carmen Sandiego? (1985)

1.1.2. Human Computer Interaction

The history of Human-Computer Interaction or HCI is directly linked with the appearance of these technological devices, so we may conclude that the area commenced with the creation of the worlds' first computer, the ENIAC (Electronic Numerical Integrator And Computer) developed by the United States Army's Ballistic Research Laboratory.

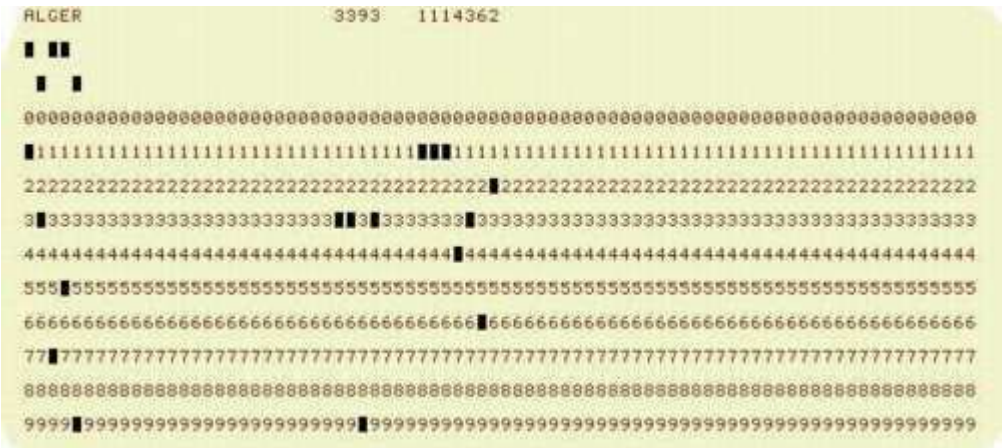

Figure 3: Punch Card

The ENIAC computer filled 167 square meters and the input/output interaction was done through punch cards (primitive form of HCI) such as the one seen in Figure 3. Does not seem very intuitive does it? Gracefully technology advanced and so did HCI.

In year 1948, the first device that could receive input from electronic, mechanical registers or tape readers was the IBM Selective Sequence Electronic Calculator (IBM SSEC), but it was not until 1951 that a primitive display in the form of blinking lights was developed for the UNIVAC 1 console.

In 1962 the teletype monitor appeared. Teletypes provided printed output of a computer session. These teletypes were improved in order to display the output of the computer onto a screen (glass teletype), making the technique faster and more versatile than paper. They remained the predominant way of interaction with computers until the mid 1970s.

The main boom of HCI occurred in 1976 with the development of the first cheap composite monitors by Don Lancaster, Lee Felsenstein and Steve Wozniak. This supposed the possibility of a cheap solution that would make computers accessible at homes. These composite monitors would continue to evolve, gaining the ability of representing color and resolution. In the 80s the QWERTY keyboard (still used nowadays) appeared, creating a revolution in terms of machine input. The evolution of technology reduced the size and increased the power of the devices available for non corporate users.

Figure 4: QWERTY Keyboard

Figure 5: J.C.R. Licklider

The advances of input and output devices created academic interest on the communication between human and computer. Joseph Carl Robnett Licklider, one of the fathers of HCI said in 1960 [6]:

"The hope is that, in not too many years, human brains and computing machines will be coupled together very tightly and that the resulting partnership will think as no human brain has ever thought and process data in a way not approached by the information-handling machines we know today. Produced goals that are pre-requisite to man-computer symbiosis".- J.C.R. Licklider

The importance of HCI is driven by the necessity of machines to get closer to human needs instead of system needs. Poor interfaces are rejected by the general public that will always seek an interface that is closer to the user's everyday life rather than the requirements of the system to capture the interactions. It is inevitable to notice the tendency of input devices to reduce size and complexity, and focus on usability by the user. The user prefers simplicity over complexity, and following this lead, this project will focus on providing the user the possibility of controlling a videogame using similar gestures that he/she uses on a daily basis

1.1.3. Previous Work – GREP

The project proposed will consist in an extension to the GREP (Game Rules scEnario Player) application developed by Daniel Barba Castaño [32] following the GRE model proposed by Zarraonandia et al (2011) [7] that fulfill the goals announced previously. Both GREP and the GRE model will be explained thoroughly in the report.

The main focus of the project is to provide a platform for education that allows videogame definition in a simple way, minimizing the technical knowledge required to do so. Making videogames development simple will allow non-technical people (e.g. school teachers) to define their own games using a set of predefined resources the platform provides and according to the educational goals each one of them may have.

1.2. Goals

In the context section we have seen the importance of the development of videogames focused on education and Human-Computer Interaction. The purpose of this project will be to join these two concepts, extending the GRE Player to allow educators to define controls for their games based on player's gestures. The objective is twofold. Firstly, to allow educators to rapidly set the gesture control that suits better the player profile. Secondly, to give educators the chance to use the player's gestures as another mean to enhance the learning in the subject covered in the game.

This objective will be approached by setting two main goals: to carry out a study on human-computer interaction for educational environments, and to develop a gesture recognition module for a game that will fulfill this study (cause-effect of the investigation).

1.2.1. Gesture Based Human Computer Interaction

As we will see in the study made on the subject and presented in the next chapter, teaching can be enhanced using simple gesture interfaces focusing on the message that the educator wants to transmit. The minimization of "secondary" learning of the interface allows the student to focus completely on the educational content and allow

a further didactic experience. The architecture of the GRE player will be extended in order to provide a simple way for the student to interact with the system through an intuitive interface based on gestures. In order to design this interface a study on human-computer interaction will be proposed with a set of predefined gestures the educator will have available for game design.

1.2.2. Design and implementation of Gesture Recognition and Game Control Modules

Derived of the necessity of developing the human interaction system, the project must be able to capture user gestures. These gestures must be able to trigger the actions of the games the GRE Player can interpret. Therefore two studies must be done to integrate this gesture recognition module: a study of current GRE Player implementation to decide the optimal way to integrate the new gesture recognition module, and a study on which motion detection device, tools and libraries can achieve the best performance for the desired project. The implementation of the module will be done according to these studies and usability of the solution will be given the utmost importance.

1.3. Development Phases

In this section the different development phases for the development of the project will be specified:

- **Planification**
- **Analysis**
- **Design**
- **Implementation & Testing**

Each one of these phases is explained in depth in the Life Cycle section, explaining each one of them thoroughly.

1.4. Tools for the Elaboration of this Document

For the creation of this project the following tools have been used:

- **Microsoft Word 2007**: Word processor developed by Microsoft used for the creation of the current document.
- **Microsoft Excel 2007**: Spreadsheet application developed by Microsoft to great tables and graphs. It has been used in this document to create Traceability Matrixes.

- **Dia 0.97**: Open source tool under GPL license inspired in Windows program "Visio". Dia has been used to create explanatory diagrams such as flow charts or UML charts.
- **SmartDraw 2014**: Graph creating tool used to create Gantt charts in a fancy and intuitive manner.

1.5. Report structure

The following document is organized in different chapters or sections. Each one of these chapters will rivet on different subjects which may be development phases or explanatory sections for further comprehension of the document and work done. A brief explanation on the chapters will be made below.

1. The first chapter (this one) will give a brief **Introduction** aimed to explain the project this work is based on as well as an overview on the current context in the videogame and human computer interaction areas. The goals of the project, the tools used and the structure of the report will also be included.
2. The second chapter will be the **State of the Art** of the document. Inside the state of the art the required HCI and motion device studies will be provided, including a comparison between all options possible and explaining the final decision made.
3. In the third chapter the **Project Management** done in the development of the solution will be explained. An explanation on the development phases of the project, the time schedule and the projects' budget will be included in this section.
4. The fourth chapter will focus on the **Analysis** provided for the system. In this section use cases, user requirements and system requirements will be specified.
5. The fifth chapter will comprehend the **Design** phase of the project. In this section the architectural design and a detailed design will be given according to the requirements specified in the analysis section. The final components of the system will be defined in this section.
6. In the sixth chapter details regarding the **Implementation** of the application will be provided. In addition to the information on the implementation, a sample game definition will be included.
7. The seventh chapter will specify the techniques followed for the **Validation and Verification** of the implemented system.
8. The eighth chapter will comprehend the **Conclusions** obtained developing the project from a personal point of view and a review of the result of the project with the goals specified in the introduction.
9. At the end of the document, the **References** used for the project will be stated, giving links towards them if available.

2. State of the art

In this point the different options available to develop the project and explain why the technologies have been chosen. First of all we shall enounce the different optical input devices available for the project and which has been chosen to be integrated in the existing solution. A brief description of the game engine Unity3D will also be provided.

2.1. Peripherals

The peripherals that have been considered for support the interaction with the GRE Player are Wiimote, Playstation Move and Kinect.

2.1.1. Wiimote

Wiimote (Figure 6: Wiimote and Nunchuk) is the primary controller for the Nintendo Wii and Nintendo Wii-U consoles since its presentation on the 14 of September of 2005. The body of the Wii Remote measures 148mm long, 36.2mm wide and 30.8mm thick. The controller communicates wirelessly with the console via a short-range Bluetooth radio. This communication allows 4 remotes functioning at the same time up to 10 meters away from the console using buttons, alas; to use the pointer functionality we must be within 5 meters.

Figure 6: Wiimote and Nunchuk

The number of attachments available is remarkable and the motion control is completed in most games with a Nunchuk.

Figure 7: Wiimote

Figure 8: Wii Nunchuk

The Wiimote has gesture recognition through-out its accelerometer and optical sensor technology. It achieves an acceptable precision at a good price. Rotational location is achieved with three accelerometers that are able to obtain the controllers position in space over the three axis.

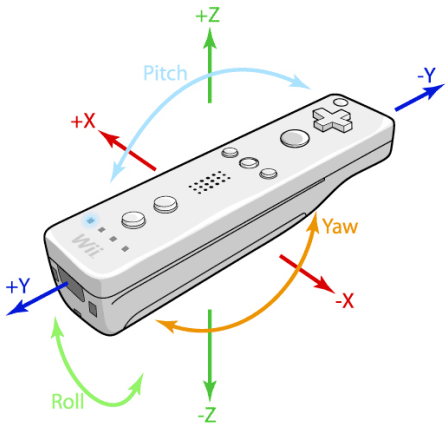

Figure 9: Wii Positioning

The Wii controller is present in the vast majority of the consoles' games, having a very good acceptance in classic Nintendo players and extending its market to non console playing users that accept the easy use of the Wiimote.

Figure 10: Man playing bowling on Wii Sports

Although the Wiimote was made only for its use for Wii and Wii-U, it has applications in the robotics field 1, medicine and augmented reality. A project called Wii Homebrew [9] has been developed to provide support for the Wiimote outside of the Nintendo environment. Notice that Wii Homebrew is not developed by Nintendo and may give some problems.

2.1.2. Playstation Move

Playstation Move is a control system for videogames on the Playstation 3 platform. Playstation Move requires to the move controller, Playstation Eye and a Navigation Controller to work.

Figure 11: Playstation Move

The move controller establishes its position with an accelerometer, a gyroscope and a magnetic sensor in three dimensions. The accelerometer measures the users' reaction time, the gyroscope is in charge of the controllers' angle and the magnetic sensor helps with certain strange movements such as spinning around. The controller additionally has a sphere that is lighten up, that serves as a reference for the Playstation Eye to calculate the position of the controller.

The positioning of the Playstation Move is similar to the one of the WiiMote, with the exception of the ball on its point that, with the help of the Playstation Eye, achieves a greater precision than the WiiMote.

Figure 12: Playstation Move Controller

Figure 13: Navigation Controller

Both the Move and Navigation controllers have the classical Playstation buttons and can be connected through Bluetooth 2.0 and USB 2.0. The navigation controller has an additional analog joystick to allow additional movements.

Figure 14: Playstation Eye

Playstation Eye is a RGB camera that takes around 120 frames per second with a 320x240 resolution or 60 frames with a 640x480 resolution. It also has a microphone able to minimize surrounding noise to focus better the users voice. The audio entrance has 4 channels of 16 bits each one.

Commercially the Playstation Move had less success than the Wiimote because it was released four years later, in 2010. Even though it went worse for the Playstation Move, it still came out with intelligent complements for its games and had a moderate success several games.

Figure 15: Man playing Killzone 3 with Playstation Move

As with the Wiimote, Playstation move has also been used for other purposes than videogames such as robotics [10] with the project Move.me [11] of Sony. The creation of Move.me is a great advantage with respect to the Wiimote, allowing third-party developers use an official tool to create applications for Playstation Move. Both Move.me and the greater precision of the Playstation Move make it an interesting option.

2.1.3. Kinect

Kinect is a device that allows us to forget completely about controllers to interact with videogames or the computer. We may simply use our own body, with no additional hardware to show the actions that must be done.

Figure 16: Microsoft Kinect with Sensors

Kinect uses an RGB camera and two 3D depth sensors that will give us a completely different experience than the previous seen options. The combination of these sensors allows us to see images captured in three dimensions, making it possible to know the distance each object is positioned.

The internal software of the Kinect device implements a skeleton recognition algorithm. It is able to separate a pattern of head, torso and limbs from the rest of the scene.

In addition, Kinect possesses four microphones that are able to perform voice recognition to give orders to our applications as well as locating where the sound came from. It has additional noise and eco removal algorithms to make the sound management more precise.

Figure 17: Man impersonating a Star Wars Jedi with Kinect

Kinect is very simple to program in comparison to the other options studied previously. There are a great number of video games that have had a good market approval as well as academic applications. Following the same strategy as Sony Playstation Move, Microsoft decided to provide developers with an official (and free) Kinect Software Development Kit [12]. In year 2012, a Japanese research group developed a Kinect application that was able to detect tongue movement [13]. This is only one of the examples of the efficiency of the Kinect movement detection and precision.

2.2. Peripheral Decision

	Kinect	Playstation Move	Wiimote
Controller	No	Yes	Yes
Camera	Yes	Yes	No
Resolution	High	High	Average
Skeleton Detection	Yes	No	No
Voice Recognition	Yes	Yes	No
Official SDK	Yes	Yes	No
Price	110 €	80 €	70 €

Table 1: Movement Detection Comparison

In Table 1 we can observe the comparison made for the studied tools to develop the movement detection controller for the project. In red we see the optimal solution according to each parameter.

At first sight we must discard the Wiimote as the only advantage respect to the others is its price (barely 10€ below Playstation Move and 40€ below Kinect). We proceed to make our choice considering only Kinect and Playstation Move.

Microsoft Kinect and Sony's Playstion Move provide a similar solution for movement detection with two main differences that will give Kinect the lead in performance for what we require for the project.

On the one hand, Kinect does not require a controller. This fact allows the user to have a direct interaction with the game enhancing the usability for inexperienced users. In addition, more movements are available due to the Skeleton Recognition feature, being able to capture movements for each joint of the body and not only the controllers movement as the Playsation Move.

On the another hand, this Skeleton Recognition software, higher market share and more robust SDK than the Playstation Move allows the implementation of the project in a simpler and faster way. The possibility of a wider range of moves allows a higher number of possibilities to develop intuitive movements for the user.

For these reasons and a mere difference in price of 40 €, Kinect has been chosen for the achievement of the required functionality.

2.3. Unity3D Game Engine

Unity3D (or simply Unity) is a cross-platform game engine developed by Unity Technologies that allows video game development for Windows, OS X, Linux, Xbox, Playstation 3, Wii, Wii U, iPad, iPhone and Android. In addition to the cited platforms, Unity3D may also be used to develop browser based games thanks to Unity's Web Plug-In.

One of the reasons Unity results so attractive for developers is because of its multiplatform nature and a variety of licensing plans. Unity offers a complete and free Indie license able to support small to large projects as well as a Pro Edition with additional features. Unity supports a simple way to create all the parts involved in a videogame (terrain, lighting, model physics, audio, different input possibilities…). Unity also provides support for Microsoft Kinect for all its licenses. The multiplatform nature of Unity and its support for different input devices such as Kinect, make it the most appropriate Game Engine to implement a practical use of the GREM model (explained in GREM: A Game Rules and Scenario Model). The latest version of Unity at the moment is 4.1.4. but the project this work is based on was developed for 3.5.0. Unity provides no backward compatibility from version 4.X. towards 3.X. projects so the latest version of the 3.X. series has been chosen. The version of Unity used is 3.5.7 (last release), with a free indie license.

Figure 18: Unity Logo

Development in Unity can be done programmatically in three different languages: UnityScript (based on Javascript), C# or Boo (based on Python). This allows the developer to choose the option that he/she feels more comfortable with. To develop the solution, both UnityScript and C# have been used (UnityScript for the extension of the existing code, and C# for the Kinect module).

Unity3D also has available an Asset Store. In the Asset Store the Unity development team offers different extra resources for your project at a determined price or some even for free. The Asset Store is available for both Pro and Indie licenses.

Figure 19: Popular horror game developed in Unity. Slender: The Arrival

One of the disadvantages of the Indie license provided by Unity is the lack of support and the proprietary nature of Unity. To have (reliable) support you must have the Pro license, and to have access to Unity's source code you need an additional license that must be negotiated with Unity's team, although they warn beforehand that this may be quite expensive [14].

2.4. Development with Unity3D and Kinect

In this section the different libraries available for Kinect application development in Unity shall be discussed. On the one hand we have the official Microsoft Kinect SDK (Kinect for Developers), and on the other hand we have an open source version (OpenNI) that can also be used. A comparison between both libraries can be found below.

2.4.1. Kinect for Windows (Microsoft Kinect SDK)

Kinect for Windows is the SDK provided by Microsoft for applications based on this device. It is mainly focused on an academic point of view and for independent developers.

Microsoft's SDK is used on Windows 7, although on the version 1.7, Windows 8 support has also been included. Applications created using this SDK are constrained to C++, C# and Visual Basic programming languages. For our desired solution this is not a problem due to Unity3D support of C#, but it may seem restrictive for other cases.

Figure 20: Kinect for Windows

The SDK includes all necessary drivers for the use of Microsoft Kinect on Windows and provides sample projects and broad documentation for application development. In addition, Kinect allows detection of one additional person (two people at the same time) that is in sight range of the device, capture and process audio, obtain distance to objects and more.

The version that may be applied for the project is v1.5 since the latest version (v1.7) is only supported after Unity3D v4. The main features lost due to this constraint is Kinect Fusion [15], used for 3D object scanning and model creation and the mentioned above Windows 8 compatibility [16].

Kinect for Windows is under a proprietary software license and its source code cannot be accessed by the public.

2.4.2. OpenNI

OpenNI provides an API for applications that use Natural Interaction (NI). This API is aimed towards covering different low level devices such as video and audio as well as higher level ones as visual detection through a computer camera. OpenNI is a multiplatform framework that provides the features necessary to write applications that use Natural Interaction. Although OpenNI is thought as multiplatform, issues regarding Kinect drivers would still arise as they are platform dependent. Even if the OpenNI framework would not fail per se, we would need to find compatible drivers for each platform, rendering the multiplatform concept unpractical for Kinect, as the official driver is proprietary software. OpenNI supports a large variety of 3D sensors as well as standard video and audio inputs.

Figure 21: OpenNI

OpenNI's API offer the functionality for whole body analysis, hand gesture and voice recognition. Coding in OpenNI is done with C++, but there is a large number of wrapper libraries (although some are incomplete) that extend its use to other programming languages (Java [17], Python [18], C# [19]…). This adds further complexity for the integration of OpenNI with Unity3D, as a functional wrapper library should be found for C#, UnityScript or Boo.

OpenNI is distributed under a GNU LGPL license, meaning that its source code is completely free and available to the public.

2.4.3. Library Decision

Table 2: Kinect for Developers and OpenNI comparison depicts the results of a comparison between the different features of the two libraries. The comparison has been carried out taking into account the previous consideration as well as the analysis perform by Hinchman in [20].

	Kinect for Windows SDK (Microsoft)	OpenNI
Calibration	No	Yes
Predictive joint tracking	Yes	No
Computation	High	Low
Resolution	1024x768	800x600
Integration with Unity	Easy	Complex
Integrated audio support	Yes	No
Noice cancelling	Yes	No
Speech Recognition	Yes	No
Multiplatform	No	Possible
Driver Installation	Easy	Hard
Programming Languages	C#, C++ and Visual Basic. Guaranteed.	C++, with additional third party wrappers for many other languages.
Documentation	Well documented with an official support forum.	No official support forum. E-mail list, Twitter and IRC.

Table 2: Kinect for Developers and OpenNI comparison

As shown in the table the SDK provided by Microsoft is a great step ahead of OpenNI, with only overperforms Kinect SDK in small power consumption and the possibility of developing a multiplatform solution. Another important advantage of the Microsoft SDK with respect to OpenNI is the great amount of additional features the former provides. As usability and user interaction are our main goals, we cannot give up the possibility of constraining our project with audio, driver or calibration limitations. As we may see explained in section Human-Computer Interaction. Gestures., speech is necessary for a broader type of semiotic human-computer gestures.

As we can see Kinect for Windows does not only have a more complete set of features, but it also provides a simpler solution for development as its installation (drivers, no need for language wrappers) is easier than the counterpart and the support and documentation are of greater quality than those offered by OpenNI.

2.5. GREM: A Game Rules and Scenario Model

The design of an Educational Game is a challenging task. EG designers not only have to deal with the inherent complexity of designing a game that engages the player but also have to interleave in its design activities that support the attainment of the learning goals proposed. This problem has created the need of models to facilitate the process of designing EGs (Kiili, 2005 [21]7; Amory and Seagram, 2003 [22]; Sweetser and Wyeth, 2005 [23]). In our case the model chosen to guide the design of the EGs has been the Game Rules and Scenario Model (GRE model) proposed in (Zarraonandia et al, 2011 [24]). The model focuses on facilitating the reuse of parts of an EG design and the production of variants of the game providing a set of configurable elements and a simple vocabulary for each feature.

The elements comprehended in the proposed model are arranged in two independent sub-models: the game rules model (left hand side Figure 22) and the scenario model (right hand side Figure 22). The first describes the rules and constraints of the game (how it should be played), and the last sub-model the virtual environment in which the game will be played and the interface provided to the user to interact with it. The idea is to be able to play a game in different scenarios and play many games on the same scenario. In order to be able to reuse the elements of the model they are organized in different layers so that the elements are hierarchically organized and outer layers are based on the inner layers. The two innermost levels of the game rules perspective describe the game mechanics and the goals the player must achieve during the game respectively. In addition, levels 3 and 4 allow designers to expand the game definition organizing different goals into a sequence of challenges that will create the storyline of the game, describe player social interaction, add

debriefing activities, define the feedback provided to the player or include rewarding and persistence mechanisms to the game definition.

With respect to the scenario model, the innermost level allows the definition of the entities represented in the game, the scenes, the characteristics and representations of game characters, and description other elements that are useful for situations that will occur in the defined scenario. The outer layers allow the definition of a set of services that will increase the possibilities of different games being played in the scenario, and to define an abstract layout in which the representation elements and the services are organized and presented to the user in each device as well as the type of interactions available for the player through the corresponding input/output devices.

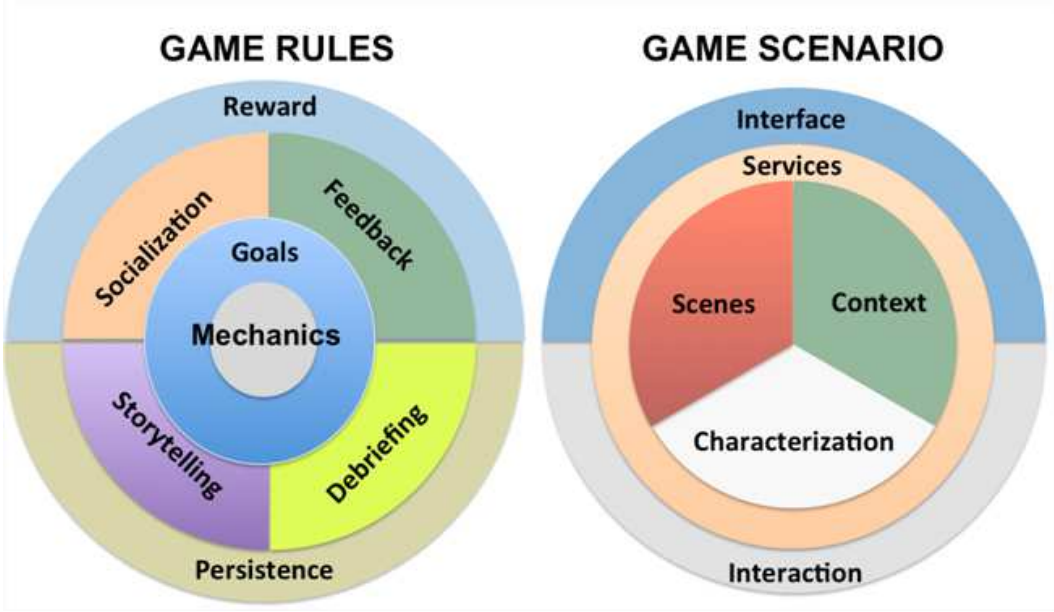

Figure 22: GREM model (Zarraonandia et al, 2011 [24])

2.6. Human-Computer Interaction. Gestures.

To consider developing a human-computer interaction application, we should first go back to human-human interactions to study which movements are most used on a daily basis (Billinghurst, 2011 [25]). Gestures may exist by themselves or they can involve an object. In this broad classification we may consider a gesture without an object a wave, a beckon or even sign language. Gestures involving objects can be pointing or handing an object to another person. According to Cadoz (1994), movements may be classified according to their function into three subclasses.

- **Semiotic**: Used to express meaningful information.
- **Ergotic**: Used to manipulate the physical world and create artifacts.
- **Epistemic**: Used to learn from the environment through tactile exploration.

In our case we will focus on semiotic gestures because we are interested in movements that may express information that the computer may analyze and interact with. Parting from the base of human-human interaction, we must attempt to choose friendly semiotic gestures that may be used on a human-computer based system. According to Rimé and Schiaratura [26], semiotic gestures may be broken up into four sub-categories.

- **Symbolic gestures**: Gestures that have a single meaning within society. "OK" gesture or thumbs up.
- **Deictic gestures**: Pointing or directing attention towards a specific action. Most common in HCI.
- **Iconic gestures**: Imitation of size and orientation of an object or a determined event. Showing how big a watermelon is with your hands or imitating a chicken are iconic gestures.
- **Pantomimic gestures**: These are gestures that consist in using imaginary objects or tools. Mimicking the use of a hammer or steering a wheel are examples of this type of gesture.

Figure 23: Symbolic Gesture (Thumbs up)

Figure 24: Deictic Gesture (I want you)

Figure 25: Iconic Gesture (Rock, paper, scissors)

Figure 26: Pantomimic Gesture (Air guitar)

This classification can additionally be ordered by the requirements of speech for their comprehension.

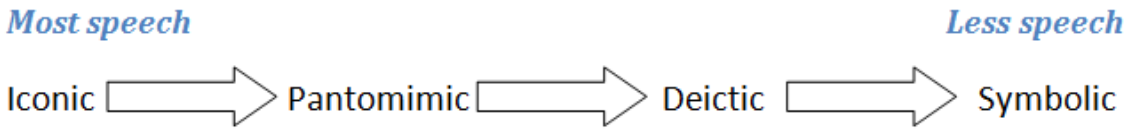

Figure 27: Gestures ordered by requirements of speech

Symbolic gestures require no additional speech to be understood, while Iconic gestures require it (thumbs up is understood easily but we cannot know if you are

imitating a watermelons size if you do not say so). Halfway between these edges we have Pantomimic and Deictic gestures that may require more or less speech depending on each gesture.

As Billinghurst says [25], one of the main reasons for using gestures for the interfaces is because of the concepts of chunking and phrasing. Each action has one or more cognitive steps (chunks) the user must learn. Unintuitive interfaces require simple conceptual actions to be broken up into compound tasks (Copy + Paste = Move), but with gesture based interfaces the input is chunked into meaningful units for the application (saying “move here” and an iconic gesture). The gesture based interface receives all its required parameters in a single cognitive chunk, while the compound task interface requires two.

This basically is telling us that an interface is as easy as the number of steps the user must take to achieve a goal, the less the steps required, the better and more intuitive the interface will be. To achieve the reduction of cognitive steps, gesture interaction can take advantage of semiotic gestures known by everyone (no need to learn neither keys nor steps to perform an action).

The easiest way of performing gesture HCI is by using Symbolic and Deictic gestures; although Iconic and Pantomimic gestures may also be used (speech recognition is required for Iconic gestures).

3. Project Management

Project management is the process to plan and control the evolution and the development of the system, the time taken in the process and the projects life cycle. In this section we will give an overview of the development life cycle for the project as well as the initial project planning (deadlines, progress diagram…).

3.1. People Involved

There are three main agents involved in this project and are directly involved with the development of the final solution.

- **Project Jury**: The project jury. The customer will be in charge of evaluating the work comprehended in this project. The jury is not in charge of any part of the life cycle; its responsibility resides in grading the completion of this in an according manner.
- **Mentor**: The mentor is in charge of supervising each phase of the life cycle of the project development. The mentor (or tutor) will also have the function of simulating the client, providing additional requirements for the project. For this project, the tutor is Telmo Zarraonandia Ayo [27].
- **Student**: The student is responsible for the completion of each phase of the system development.

3.2. Life Cycle

In the following capture, we can see the overall plan for the project. The process followed is a waterfall model. We can see that we have four different development phases each independent from each other. Once a phase has ended the cycle continues towards the next step, not returning to the previous one unless a critical or urgent fix is required. The capture is made with Dia 0.97.2, an UML drawing program [29].

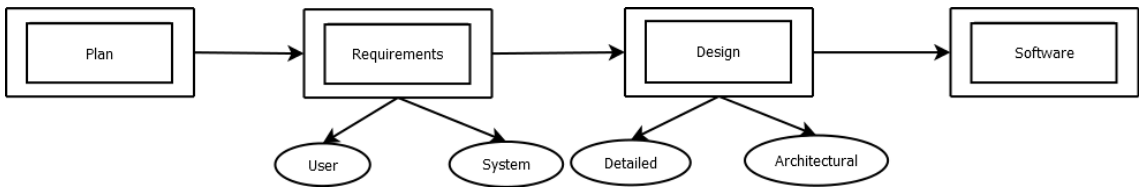

Figure 28: Project Life cycle

The four phases in the development life cycle are the following:

- **Plan**: This is the beginning step of the development cycle. We arrive at this point when we desire to create a new system. In this phase the tasks are focused towards documentation on the tasks to be done and a viability study to know if the project can be achieved. In our case the plan phase consists in expanding the knowledge in gesture based HCI, studying the previous existing project and a case study on existing peripherals and the mechanisms available to develop the solution programmatically.
- **Requirement definition (or Analysis)**: At this point we proceed to analyze use cases and user requirements taking into account the desired functionality inherited from the plan phase. The IEEE defines a requirement as a condition or a capability for a user to resolve a problem or reach a goal, or a condition a system must fulfill to satisfy a contract, a standard, a specification or any other imposed formal document. As we may observe in Figure 28: Project Life cycle, requirements can be split into two sub-groups: User requirements and System requirements.
    - **User Requirements**: Condition or capability for a user to resolve a problem or achieve a goal.
    - **System Requirements**: Condition or capability the system must comply with in order to satisfy a contract, a standard, a specification or any other imposed formal document.
- **Design**: Subsequent to the analysis phase comes the design phase. In this step the system components, the relationships between them and the used architecture are designed, comprehending all requirements defined in the analysis. Taking a glimpse again at Figure 28: Project Life cycle, we see that the design is composed by an architectural design phase and a detailed design phase. The architectural design is required to create the detailed design.
    - **Architectural Design**: The Architectural Design describes the basic system design for the software to be made during the project. It describes the physical model; this is a decomposition of the software into components. Each component is described in terms of its external interfaces and the dependencies on other components. All requirements with be incorporated into this architectural design, depicting at a high level the appropriate modules, data structures, databases and interfaces. This document serves as a basis for the detailed design, which will establish the design in increased detail.
    - **Detailed Design**: The purpose of the detailed design is to specify a high level view for the subsystems resulting in the architectural design. The detailed design is used to clear the formal description of the architectural design.

- **Software**: In this final phase, the application is developed following the previous design documents and complying with the defined requirements. This block can be split into three sub-blocks, in this order: Implementation, Verification and Maintenance. Implementation is the development of the application itself programmatically. Verification allows checking that the implemented solution is compliant with the specified requirements. Once verification is finished and the product is released, a maintenance sub phase could be considered, although it is not in the scope of this project. Additional documentation such as user manuals or diagrams may be included if considered necessary.

Splitting up the software phase for more clarity, the final process after planning would be the following:

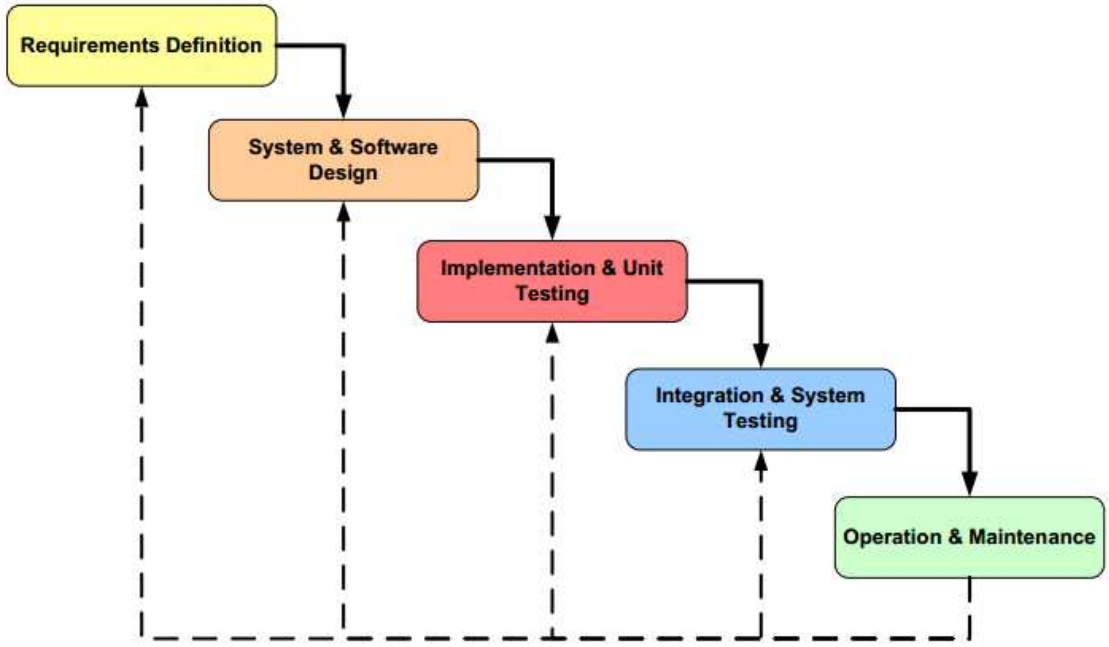

Figure 29: Waterfall model

In Figure 29: Waterfall model we can observe that the project process may seem intolerant to errors coming from the higher phases. In this model requirement changes from the client result difficult once the requirement definition has been approved and few systems have stable requirements. Although we have these problems, the dimension of the project and the relationship with the client (tutor) makes them less critical than in a real business environment. The distended deadlines stated for the project allow the definition of further improvements in the operation & maintenance phase.

3.3. Delivery Time and Initial Planning

This project was started the 30th of January and the final deadline for the project was set by the customer to the 25th of June; therefore the project has an approximate duration of 5 months.

For a correct planning, it is necessary to identify correctly each task and estimate the required time for each one of them. Normally each task is assigned to a different member of the team but in this case all of them are assigned to the student (obviously).

To illustrate the initial project plan we have chosen a Gantt chart, so it is possible to visualize the start and end of each of the development phases. In this Gantt chart you can see the effort for each task graphically in form of bars. The first line of the chart represents the whole duration of the project. The following Gantt chart was made with SmartDraw 2014 [28].

	Task	Start	End	Dur
	Project Planning	30/1/13	25/6/13	105
1	Plan	30/1/13	28/2/13	22
2	Analysis	1/3/13	15/3/13	11
3	Architecture & Design	15/3/13	1/4/13	12
4	Implementation	1/4/13	1/5/13	23
5	Testing	1/5/13	8/5/13	6
6	Implementation Milestone	15/5/13	15/5/13	
7	Additional documentation	15/5/13	25/6/13	30
8	Compulsory Hand In	25/6/13	25/6/13	

Figure 30: Gantt chart – Initial Planning

We can easily observe in Figure 30 the effort in days for each task and two existing milestones in red. The first one is a milestone that both the student and the tutor set to have the project implemented, and the second one is the milestone set by the customer. The deviation that has occurred with respect to this planning may be found in the conclusions section.

3.4. Budget

In this part the budget of the project is discussed. The final quality of the process depends directly on a correct and realist budget prevision.

3.4.1. Direct costs

In this section we provide an overview of the direct costs linked with project development. Direct cost is defined as the cost that the development of the product has implicitly, this is labor and material.

To complete this project one software engineer is required (the student). The software engineer will adopt the functions of a system analyst, a designer, a programmer and a tester according to the skills required for each one of the development phases.

According to Infojobs Trends [3029], the average wage for a software engineer is 25.000 € p.a. Calculating the amount necessary for 5 months, we obtain that the total cost of the software engineer is 10.416,67 €.

Additionally, a Microsoft Kinect device was added to the project for a total amount of 110 €.

Notice that no enterprise benefits are considered for this project as it is considered to be non-profit.

Asset	Cost
Software Engineer	10.416,67 €
Microsoft Kinect	110 €
Total Direct Costs	10.526,67 €

Table 3: Direct Costs

3.4.2. Indirect Cost

Indirect costs are those that are derived from the operation of an enterprise and are not linked to only one product (e.g. office, security, paper…). According to Rodrigo and Berges [31], indirect costs in most Spanish university investigation projects are calculated as a total percentage of the direct costs. In this work we can see that the most common percentage used to calculate indirect costs is between 10 and 15 %, so we have chosen 15 % for our indirect cost calculation.

Asset	Cost
15 % of Direct cost	1.579 €
Total Indirect Cost	1.579 €

Table 4: Indirect Costs

3.4.3. Total Cost

Taking into account both direct and indirect cost, we obtain the complete budget required for the completion of the product.

	Cost
Direct Cost	10.526,67 €
Indirect Cost	1.579 €
Total Cost Without Tax	12.105,67 €
Tax (21 %)	2.542,19 €
Total Cost (Tax included)	14.647,86 €

Table 5: Total Cost

In Table 5: Total Cost we observe that the budget (direct plus indirect costs) is of 12.105,67 €. Considering taxation of 21%, the final budget results in 14.647,86 €.

3.4.4. Payment Plan

This project is considered to be non-profit so neither labor, indirect costs nor tax should be considered in the payment plan. The only part required to be taken into account is the purchase of the Microsoft Kinect that will be done by the Informatics Department of Universidad Carlos III of Madrid at March 1st 2013 under the supervision of the Project Mentor.

4. Analysis

In the Analysis section we shall see an overview of the general functionalities the system will comprise. A formal definition will be given for each feature using use cases and requirement definitions. The intended functionality obtained in the planning phase will be explained in the **¡Error! No se encuentra el origen de la referencia.** part.

This process has been done consulting the Project Mentor (Telmo Zarraonandia) simulating the client. Once the analysis is finished, it shall be handed to the client for its verification and confirmation for the document to function as a contract between both parts.

4.1. General Capabilities

At the moment there are many solutions available for videogame developers to speed up the implementation of their projects, but all of them share the need of a high level of qualifications to be able to produce a videogame. This complexity creates an issue for educators exploring the possibility of teaching their students using videogames. The idea of the GREM model and this project is to lower to a trivial level the creation of videogames through a configurable platform and an interface appropriate to ease learning at any level, liberating the user of the obligation of learning keys and clicks and focusing more on the message sought by the videogame designer. As explained in Human-Computer Interaction. Gestures., it is simpler to learn with less cognitive steps; if we abolish the control cognitive step, we focus completely on the cognitive step intended by the educator creating the videogame.

The application is considered to be an extension to an existing project (Barba, D. 2012 []). The existing application provides a GREM architecture for a keyboard and mouse interface. The extension desired is focused to provide an intuitive interface based on the study in Human-Computer Interaction. Gestures. adding the Kinect functionality and enhancing the HCI experience in learning environments.

The tool used for the motion detection required for the sought HCI model is Microsoft Kinect. The motion detection extension will be fully compliant with the GREM model and will still allow the previous interface selection. The resulting application shall be a configurable game framework with the possibility of control through a mouse and keyboard interface or through a gesture interface with Kinect, allowing the user to choose between both options.

4.2. General Restrictions and Constraints

The restrictions for this project are similar to every available commercial application, additionally constrained by the GREM: A Game Rules and Scenario Model. The GREM model states that the system shall have a scenario and a rules sub-module and that they must be configurable to provide usability to the software components.

4.3. Operational Environment

The system must be able to interact with Microsoft Kinect through a Windows 7 operating system. The system will be able to capture player movements using the motion detection system provided by Kinect. The application will be in form of a portable executable file compatible in Windows 7 with a Kinect device installed.

4.4. Use Cases

Use cases allow us to describe the system from a user point of view, they shall specify the use of the system and how it interacts with the users.

First of all, we proceed to define all the actors that will interact with the system. According to the nature of the user that accesses the system, we define two actors:

-**Educator**: It is the user that desires create a new videogame aided by the GREM model. The educator defines the game rules and the game scenario through XML.

-**Player**: The player is the user that interacts with the videogame designed by the educator. The player's movements will be captured using the available interfaces, this is, keyboard and mouse or the gesture based interface.

Below we show a use case diagram for the player actor in the system that will give us a visual schema of which will be the actions this user type has available in the profiled system. The following figures have been made with Dia 0.97.2, an UML drawing program [29].

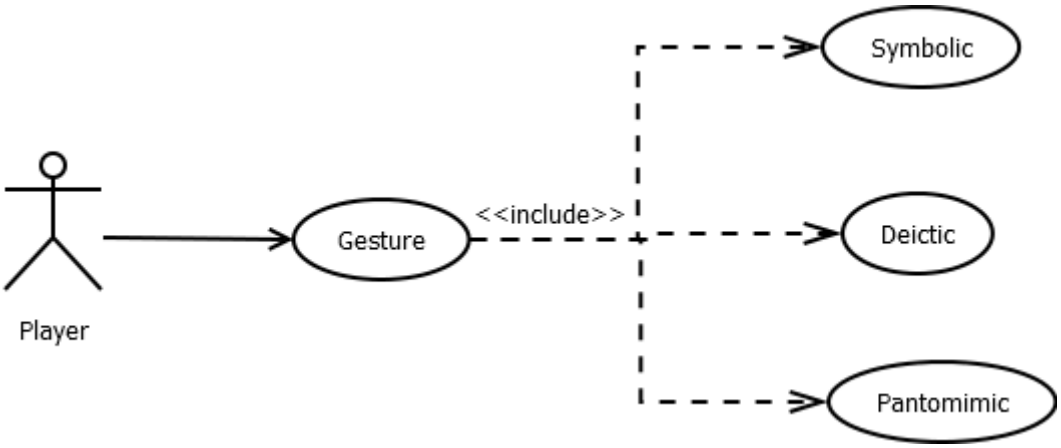

Figure 31: Player use case diagram

We proceed to describe formally each of the user requirements in order to provide an overview of the actor-system interactions taken place and what conditions must arise for them to appear. The user requirements section is aimed to help the customer understand the extension and functionality of the project.

4.5. User Requirements

In this section a specific user requirement definition will be made divided into logical subsections aimed for comprehension and further modification improvement. First of all a definition of functional and non-functional requirement will be made for clarification.

- **Functional Requirements**: The requirements that describe how the system works. They shall define software's the internal behavior, specifying how the use cases will be fulfilled.
- **Non-Functional Requirements**: A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system rather than specific behaviors.

4.5.1. Good Software Requirements: IEEE-830

The characteristics according to standard IEEE-830 [33] that a good software requirement definition must fulfill are the following:

A good software requirement should be:

- **Correct:** An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet. There is no tool or procedure that ensures correctness. The SRS should be compared with any applicable superior specification, such as a system requirements specification, with other project

documentation, and with other applicable standards, to ensure that it agrees. Alternatively the customer or user can determine if the SRS correctly reflects the actual needs. Traceability makes this procedure easier and less prone to error.

- **Unambiguous:** An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term. In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.
- **Complete**: An SRS is complete if, and only if, it includes the following elements:
    - All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.
    - Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.
    - Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.
- **Consistent:** Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.
- **Ranked for importance and/or stability**: An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement. Typically, all of the requirements that relate to a software product are not equally important. Some requirements may be essential, especially for life-critical applications, while others may be desirable. This ranking concerns stability and necessity.
- **Verifiable**: An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable. Non-verifiable requirements include statements such as “works well,” “good human interface,” and “shall usually happen.” These requirements cannot be verified because it is impossible to define the terms “good”, “well,” or “usually.” The statement that “the program shall never enter an infinite loop” is non-verifiable because the testing of this quality is theoretically impossible.
- **Modifiable**: An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and

consistently while retaining the structure and style. Modifiability generally requires an SRS to:

- Have a coherent and easy-to-use organization with a table of contents, an index, and explicit crossreferencing.
- Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS).
- Express each requirement separately, rather than intermixed with other requirements.

Redundancy itself is not an error, but it can easily lead to errors. Redundancy can occasionally help to make an SRS more readable, but a problem can arise when the redundant document is updated. For instance, a requirement may be altered in only one of the places where it appears. The SRS then becomes inconsistent. Whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable.

- **Traceable**: An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:
  - Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.
  - Forward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number.

  The forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

4.5.2. User Requirement Definition

Taking into account the standard IEEE 803 (explained in Good Software Requirements: IEEE-830) for software requirements, the attributes for the user requirements are the following:

- **Identifier:** Each user requirement shall include an identifier, to facilitate tracing through subsequent phases.
  - **Value**: URYXX (User Requirements plus number). We will group them in different categories, so Y will identify the category and XX the requirement number. This way, whenever a requirement gets modified, we will save some more numeration changes in the user requirement definitions and the system requirement definitions.

- **Necessity:** Essential user requirements shall be marked as such. Essential user requirements are non-negotiable; others may be less vitally important and subject to negotiation.
    - **Values**: essential / desirable / optional / possible future enhancement
- **Priority**: For incremental delivery, each user requirement shall include a measure of priority so that the developer can decide the production schedule.
    - **Values**: high / medium / low
- **Stability**: Some user requirements may be known to be stable over the expected life of the software; others may be more dependent on feedback from the system requirements, the architectural and design phases, or may be subject to change during the software life cycle. Such unstable requirements should be flagged.
    - **Values**: Yes/ No
- **Source**: The source of each user requirement shall be stated. This may be a reference to an external document (e.g. system requirement document) or the name of the user, or user group, that provided the user requirement.
    - **Values**: customer / developer
- **Clarity:** A user requirement is clear if it has one, and only one, interpretation. Clarity implies lack of ambiguity. If a term used in a particular context has multiple meanings, the term should be qualified or replaced with a more specific term.
    - **Values**: Yes / No
- **Description**: Describes the requirement.

4.5.2.1. Functional Requirements

Below the functional requirements of the Project will be defined using the representation model defined in the previous point.

4.5.2.1.1. Educator

ID	UR101		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	The kinect controller will detect a set of predefined gestures (semiotic, deictic, iconic and pantomimic		

Table 6: UR101

ID	UR102		
Priority	Medium	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		
Description	The educator will be able to assign game actions both to the mouse and keys controller as well as to the Kinect controller.		

Table 7: UR102

ID	UR103		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	The game shall be controlled as specified by the educator.		

Table 8: UR103

4.5.2.1.2. Player

ID	UR201		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	The system will be able to capture the player's gestures.		

Table 9: UR201

ID	UR202		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	A game action will be triggered when its associated gesture is performed.		

Table 10: UR202

ID	UR203		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	The player must control the character as intended by the educator.		

Table 11: UR203

4.5.2.2. Non-Functional Requirements

Below you can find defined the non-functional requirements following the description in User Requirement Definition.

4.5.2.2.1. Gestures

ID	UR301		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	The predefined gestures must be designed according to the HCI study provided in Human-Computer Interaction. Gestures. section.		

Table 12: UR301

ID	UR302		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	Kinect will be the motion detection device for gesture recognition.		

Table 13: UR302

ID	UR303		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		

Description	User gestures should be detected fast enough in order to play the game comfortably.

Table 14: UR303

ID	UR304		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		
Description	Kinect will use a library that allows audio detection.		

Table 15: UR304

ID	UR305		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	The predefined gestures will be at least 5.		

Table 16: UR305

4.5.2.2.2. Environment

ID	UR401		
Priority	Medium	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		
Description	The application will be distributed in a portable scheme, requiring no installation to be played.		

Table 17: UR401

ID	UR402		
Priority	Medium	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		

Description	The application will function on Windows 7 computer.

Table 18: UR402

ID	UR403		
Priority	Medium	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		
Description	The application will require Kinect drivers present on the computer (automatic plugging in Kinect)		

Table 19: UR403

ID	UR404		
Priority	Medium	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Desirable		
Description	The user will require the Kinect device to be able to use the gesture based interface.		

Table 20: UR404

ID	UR405		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential		
Description	After developing the gesture extension, the original functionality will remain intact. The educator and player may still use the mouse and keyboard functionality. The architecture will follow the GRE Player schema.		

Table 21: UR405

4.5.3. System requirements

The purpose of this section is to specify the System Requirements of the project in a clear and consistent manner. These requirements are a negotiated agreement between the customer and the student. Listed requirements, and only these, will be implemented in the project, according to their priorities. Any changes to these requirements require the full consent of both parties.

Every user requirement specified in User Requirement Definition can trigger a different number of system requirements.

The specific requirements discussed in this point will be divided into logical subsections. Taking into account the standard IEEE 803 (explained in Good Software Requirements: IEEE-830) for software requirements, the attributes for the user requirements are the following:

- **Identifier:** Each user requirement shall include an identifier, to facilitate tracing through subsequent phases.
    - **Value**: SRYXX (User Requirements plus number). We will group them in different categories, so Y will identify the category and XX the requirement number. This way, whenever a requirement gets modified, we will save some more numeration changes in the user requirement definitions and the system requirement definitions.
- **Necessity:** Essential user requirements shall be marked as such. Essential user requirements are non-negotiable; others may be less vitally important and subject to negotiation.
    - **Values**: essential / desirable / optional / possible future enhancement
- **Priority**: For incremental delivery, each user requirement shall include a measure of priority so that the developer can decide the production schedule.
    - **Values**: high / medium / low
- **Stability**: Some user requirements may be known to be stable over the expected life of the software; others may be more dependent on feedback from the system requirements, the architectural and design phases, or may be subject to change during the software life cycle. Such unstable requirements should be flagged.
    - **Values**: Yes/ No
- **Source**: The source of each user requirement shall be stated. This may be a reference to an external document (e.g. system requirement document) or the name of the user, or user group that provided the user requirement.
    - **Values**: customer / developer / inherited
- **Clarity:** A user requirement is clear if it has one, and only one, interpretation. Clarity implies lack of ambiguity. If a term used in a particular context has multiple meanings, the term should be qualified or replaced with a more specific term.
    - **Values**: Yes / No
- **Description**: Describes the requirement.

4.5.3.1. Educator

ID	SR101		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	Actions available for the educator will be those defined in the Gestures section.		

Table 22: SR101

ID	SR102		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	Semiotic gestures available for the educator will be those defined in the Gestures section.		

Table 23: SR102

ID	SR103		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	Deictic gestures available for the educator will be those defined in the Gestures section.		

Table 24: SR103

ID	SR104		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	Iconic gestures available for the educator will be those defined in the Gestures section.		

Table 25: SR104

ID	SR105		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	Pantomimic gestures available for the educator will be those defined in the Gestures section.		

Table 26: SR105

ID	SR106		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	The new game interactions will be defined in XML format.		

Table 27: SR106

ID	SR107		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR101
Description	New interactions should be compliant with the specification of the GRE model		

Table 28: SR107

ID	SR108		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR102
Description	The system shall interpret correctly the actions defined with the Kinect controller with the specified syntax in SR103.		

Table 29: SR108

ID	SR109

Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR103
Description	When an action is triggered after a gesture, it shall match the action definition done by the educator.		

Table 30: SR109

ID	SR110		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR103
Description	If an action is not defined, a default value may be assigned to it.		

Table 31: SR110

4.5.3.2. Player

ID	SR201		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR201
Description	A player will be able to perform a predefined gesture and have it captured by the system.		

Table 32: SR201

ID	SR202		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR201
Description	The position of all the joints of the player will be available to the controller. The joints captured by the controller will be the following 20 skeletal parts: - Hip Center		

	- Left Hip - Right Hip - Spine - Head - Shoulder Center - Left Shoulder - Right Shoulder - Left Elbow - Right Elbow - Left Wrist - Right Wrist - Left Hand - Right Hand - Left Knee - Right Knee - Left Ankle - Right Ankle - Left Foot - Right Foot

Table 33: SR202

ID	SR203		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR201
Description	Additional gestures may be defined using the joints captured by the controller.		

Table 34: SR203

ID	SR204

Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR201
Description	Angles between bones will be accessible using the joints detected.		

Table 35: SR204

ID	SR205		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR201
Description	The predefined gestures that will be captured by the controller will be those defined in the Gestures section		

Table 36: SR205

ID	SR206		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR202
Description	Provided a gesture has been captured by the system, an action will be triggered if it has been designed to do so by the educator.		

Table 37: SR206

ID	SR207		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR202
Description	The actions the user may commit are defined in the Gestures section		

Table 38: SR207

ID	SR208		
Priority	High	Source	Customer

Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR202
Description	The character will remain idle until all joints of the player are detected.		

Table 39: SR208

ID	SR209		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR203
Description	The actions and gestures available for the player must be defined in XML format to be valid. If an action has been defined		

Table 40: SR209

4.5.3.3. Gestures

ID	SR301		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR301
Description	The system shall allow definition of semiotic gestures.		

Table 41: SR301

ID	SR302		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR301
Description	The system shall allow definition of deictic gestures.		

Table 42: SR302

ID	SR303		
Priority	High	Source	Customer

Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR301
Description	The system shall allow definition of iconic gestures.		

Table 43: SR303

ID	SR304		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR301
Description	The system shall allow definition of pantomimic gestures.		

Table 44: SR304

ID	SR305		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR301
Description	Gestures shall be defined regarding usability.		

Table 45: SR305

ID	SR306		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR302, UR303, UR304
Description	The solution will be developed for Kinect.		

Table 46: SR306

ID	SR307		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes

Necessity	Essential	Related to UR	UR303
Description	The controller must detect movements in a fast manner once the body is detected (less than 0.5 seconds per body part).		

Table 47: SR307

ID	SR308		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR303
Description	When the body is not detected, no actions nor body parts will be detected.		

Table 48: SR308

ID	SR309		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR304
Description	The library used will be Kinect for Windows, as specified in Development with Unity3D and Kinect.		

Table 49: SR309

ID	SR310		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR305
Description	The solution will provide at least 5 predefined gestures. The predefined gestures are defined in the Gestures section.		

Table 50: SR310

ID	SR311		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR305

Description	The predefined gestures will have at least one example of each of the following categories defined in Human-Computer Interaction. Gestures.: Semiotic, Deictic and Pantomimic.

Table 51: SR311

4.5.3.4. Environment

ID	SR401		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR401
Description	The solution will be compiled as a Windows application using Unity 3.5. This compilation provides a portable solution.		

Table 52: SR401

ID	SR402		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR402
Description	The application will function on Windows 7 computer.		

Table 53: SR402

ID	SR403		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR403
Description	The application will require Kinect drivers present on the computer (automatic plugging in Kinect)		

Table 54: SR403

ID	SR404		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes

Necessity	Essential	Related to UR	UR404
Description	The user will require a Kinect device installed to use the gesture based interface.		

Table 55: SR404

ID	SR405		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR405
Description	The mouse and keys controller will function in the new solution.		

Table 56: SR405

ID	SR406		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR405
Description	The XML parser will function in the new solution.		

Table 57: SR406

ID	SR407		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR405
Description	The scenario and rule definition syntax will remain unchanged.		

Table 58: SR407

ID	SR408		
Priority	High	Source	Customer
Stability	Yes	Clarify	Yes
Necessity	Essential	Related to UR	UR405

Description	The extension shall be made following the existing architecture of the GRE Player

Table 59: SR408

4.5.4. System Requirement Traceability Matrix

SR/UR	UR101	UR102	UR103	UR201	UR202	UR203	UR301	UR302	UR303	UR304	UR305	UR401	UR402	UR403	UR404	UR405
SR101	■															
SR102	■															
SR103	■															
SR104	■															
SR105	■															
SR106	■															
SR107	■															
SR108	■	■														■
SR109		■														■
SR110		■														
SR111			■													
SR112			■													
SR201				■												
SR202				■												
SR203				■												
SR204				■												
SR205				■												
SR206				■												
SR207					■											
SR208					■											
SR209					■											
SR210						■										
SR301							■									
SR302							■									
SR303							■									
SR304							■									
SR305							■									
SR306								■	■	■						
SR307									■							
SR308									■							
SR309										■						
SR310											■					
SR311											■					
SR401												■				
SR402													■			
SR403														■		
SR404															■	
SR405																■
SR406																■
SR407																■
SR408																■

Table 60: Traceability Matrix

To illustrate the granularity of the system requirements definition a traceability matrix has been included.

5. Design

In this step the system components, the relationships between them and the used architecture are designed, comprehending all requirements defined in the analysis.

5.1. Architectural Design

In this section we will proceed to specify the architectural design for the system. The architectural design describes the physical model; this is a decomposition of the software into components. Each component is described in terms of its external interfaces and the dependencies on other components. All requirements with be incorporated into this architectural design, depicting at a high level the appropriate modules, data structures, databases and interfaces. This section serves as a basis for the detailed design, which will establish the design in increased detail.

5.1.1. Architecture Pattern – GREP Engine

The proposed solution has been made following the GREP (Game Rules scenario Player) architecture. The application interprets EGs described in XML files following the GRE model (GREM: A Game Rules and Scenario Model) defining both scenario and rules. The application is implemented using Unity3D engine version 3.5 and is currently limited to single player EGs. Figure 32: GREP Architecture has been made with Dia 0.97 [29].

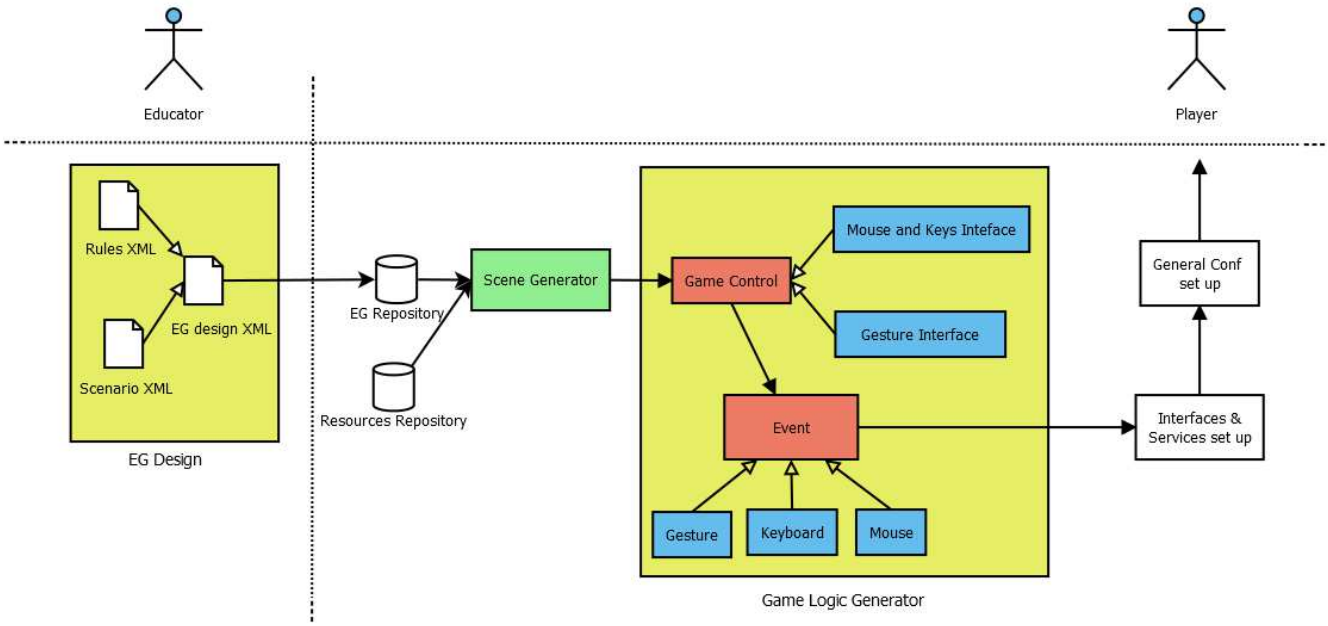

Figure 32: GREP Architecture

As shown in Figure 32: GREP Architecture all available textures, models, scripts are included in the Resources Repository in order to make them available for game

definition. Once a series of resources have been uploaded, the educator can begin to develop EG XML files for the EG Repository. The EG XML files will comprehend the scenario and rules definition as specified by the GRE model. If the scenario requires a determined resource, this asset must be present in the Resources Repository.

When the user selects a game to play, the GREP system interprets the EG XML file in order to define all the assets corresponding to the game scene. Each one of these assets will have a specified size, position and texture specified in the XML file.

After the game scene is created, the game logic will be extracted from the XML file. This game logic includes actions that are then associated to controls and model animations. Once the game logic is defined, each one of the events listed in the XML file will have a determined listener in order to capture them.

This project is focused on modifying the Game Logic Generator section in Figure 32: GREP Architecture to provide support for a gesture based interface. If the gesture interface is selected, a motion detection controller will be used and a gesture listener will be provided.

5.1.2. Component Description

All the components in the project are described in this chapter. The system will be composed of 2 components: Gestures Control and Listener as follows:

ID and Title	**AD01 Gesture Control component**		
Identifier	AD01	Type	Class
Children	None	Dependencies	None
Input Data	Nothing	Output Data	Gestures (if any)
Goal	Detect gestures provided the users joint positions		
Interface			
Functionality	If the user has made a predefined gesture (defined in Gestures), the component shall detect it.		
References	SR102, SR103, SR104, SR105, SR108, SR201, SR202, SR203, SR204, SR205, SR208, SR301, SR302, SR303, SR304, SR305, SR306, SR307, SR308, SR309, SR310, SR311, SR403, SR404		

Table 61: AD01 Gesture Component

ID and Title	**AD02 Listener component**		
Identifier	AD02	Type	Class
Children	None	Dependencies	AD01
Input Data	Gesture	Output Data	Event

Goal	Make an event occur after a gesture is detected.
Interface	
Functionality	If the user has made a predefined gesture (defined in Gestures), the component shall check if an event is associated. If an event is associated it shall be triggered.
References	SR101, SR106, SR107, SR108, SR109, SR110, SR206, SR207, SR208, SR209, SR308, SR405, SR406, SR407, SR408

Table 62: AD02 Listener Component

5.1.3. Architectural Design Traceability Matrix

SR/ADC	AD01	AD02
SR101		
SR102		
SR103		
SR104		
SR105		
SR106		
SR107		
SR108		
SR109		
SR110		
SR201		
SR202		
SR203		
SR204		
SR205		
SR206		
SR207		
SR208		
SR209		
SR301		
SR302		
SR303		
SR304		
SR305		
SR306		
SR307		
SR308		
SR309		
SR310		
SR311		
SR401		
SR402		
SR403		
SR404		
SR405		
SR406		
SR407		
SR408		

Table 63: AD Traceability Matrix

Table 63: AD Traceability Matrix has been included in order to show the granularity of the system. Please observe that neither SR401 nor SR402 have been included into any component due to their implicit nature in the Unity platform.

5.1.4. Data model

Below in Figure 33: Datamodel the data model of the system has been profiled. We may observe that the main unit of information for the educator will be the actions. These actions are used to link the "actions" in-game with the physical interaction the player performs. Notice that the definition of these actions could include additional information (additional requirements in Figure 33: Datamodel) such as a certain amount of "mana" or health that the character needs to have in order to trigger an action. The interactions associated to the actions control can be of different nature according to the interaction the user performs with the system. The interaction associated may be a Gesture performed by the user, a mouse click or stroke, or a keyboard event (a key is pressed).

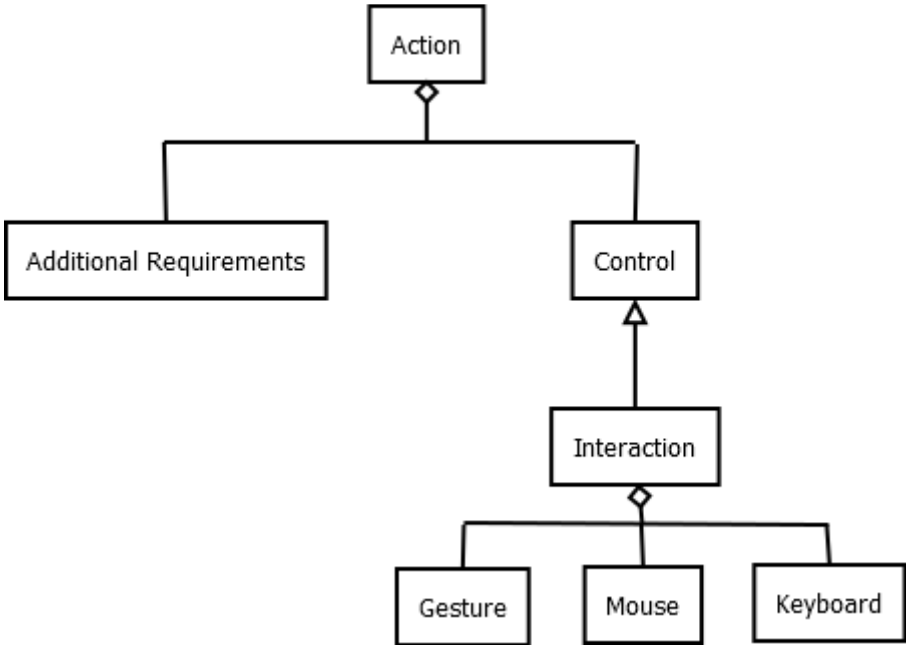

Figure 33: Datamodel

The data model is taken into account when defining action syntax for the XML files.

5.1.5. Activity Flow Diagram

As we may see in the Activity Flow Diagram in Figure 34: Flow chart, the paths followed by the system are quite simple. After the controller is instantiated, it shall proceed in detecting the players' body. If the body is detected the bones are captured. These bones are analyzed by the system that checks if the bones position corresponds to a gesture. Once a gesture is captured the system will proceed to check if it is associated with a game action. If so, the action is performed.

If the body is not detected the system shall remain idle. If a gesture is not captured, it is not associated with an action or an action is performed, the system shall return to the initial state and attempt to detect the players' body.

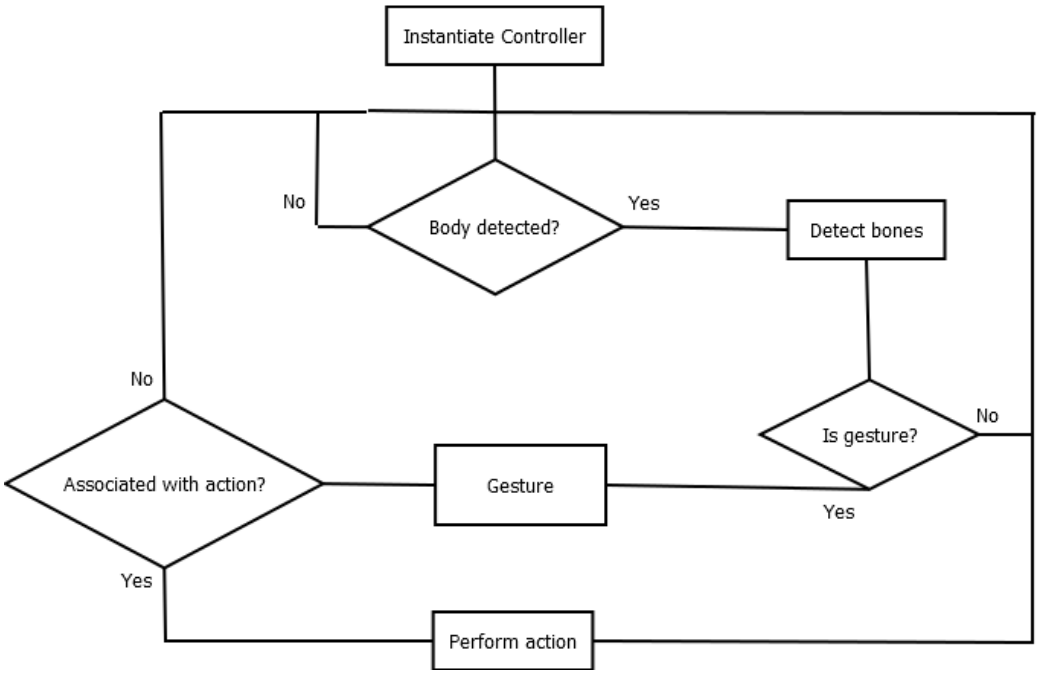

Figure 34: Flow chart

5.1.6. Gestures

According to the requirements stated in User Requirement Definition, it has also been necessary to design a set of predefined gestures that will be made available to the game user to define the actions and interactions of his/her game. Table 64: Gesture Classification presents the set of gestures that will be implemented:

Symbolic Gestures	Deictic Gestures	Pantomimic Gestures
Left Arm forming 90 degrees angle	**Join Hands**	**Character Walking**
Right Arm forming 90 degrees angle	**Raise Left Arm**	**Join Hands**
Right Leg Raised	**Raise Right Arm**	**Crouch**
Left Leg Raised		**Jump (both legs raised)**
Both arms Raised		

Table 64: Gesture Classification

As shown in the table, we have designed a set of 11 movements according to the study done in the section Human-Computer Interaction. Gestures. We have 5 symbolic gestures, 3 deictic gestures and 3 pantomimic gestures. Notice that the gesture based on joining hands can be both a deictic gesture (point with both hands at something) and a pantomimic gesture (fire a water hose). In the study we explained that movements are classified according to the required speech required to understand

them. The less the speech required, the easier the gesture can be developed programmatically (this explains the greater number of symbolic gestures). Iconic gestures would require user speech analysis and has been considered out of scope for this project, although it can be achieved in future works on the same application.

The actions available will be those of the previous solution. These actions are:

- Walk forward
- Move Left
- Move Right
- Move back
- Jump
- Run
- Shoot
- Interact with object

Notice that these actions may be increased in future versions of the solution.

5.2. Detailed Design

The purpose of this section is to refine the high level view of the system into subsystems and interactions between them. The detailed design clarifies the formal description of the system.

5.2.1. Component Description

ID and Title	**DD01 Skeleton Wrapper component**		
Identifier	DD01	Type	Class
Children		Dependencies	DD02
Input Data	Skeleton position from the Kinect Sensor	Output Data	
Goal	Poll the users' skeletal position in order to obtain body joints.		
Interface			
Functionality	Obtain body joints using KinectSensor.pollSkeleton		
References	SR201, SR202		
Resources			
Process	Update skeleton position, tracking all joints. Process new skeleton information. **pollSkeleton ()**		

Table 65: DD01 Skeleton Wrapper

ID and Title	**DD02 Kinect Sensor component**		
Identifier	DD02	Type	Class
Children		Dependencies	
Input Data		Output Data	
Goal	Initialize Kinect device to use in the system.		
Interface			
Functionality	Sensor height and look position. Initialize NUI for skeleton tracking.		
References	SR307, SR309, SR404		
Resources	Kinect for Windows.		

Process	Initialize NUI features **awake ()** Get next frame from Kinect and update skeleton **pollSkeleton()**

Table 66: DD02 Kinect Sensor

ID and Title	**DD03 Kinect Point Controller component**		
Identifier	DD03	Type	Class
Children		Dependencies	DD05
Input Data	User Skeleton	Output Data	Gesture identification
Goal	Identify gestures provided a determined skeletal position.		
Interface			
Functionality	Process the skeletal position to identify gestures.		
References	SR203, SR204, SR205, SR301, SR302, SR303, SR304, SR305, SR308, SR310, SR311		
Resources			
Process	Poll skeletal position and identify gestures. **update ()**		

Table 67: DD03 Kinect Point Controller

ID and Title	**DD04 Depth Wrapper component**		
Identifier	DD04	Type	Class
Children		Dependencies	DD02
Input Data		Output Data	
Goal	Wrapper for depth frames		
Interface			
Functionality	Provides: -a frames of depth image (no player information), -an array representing which players are detected, -a segmentation image for each player, -bounds for the segmentation of each player.		
References	SR308, SR309, SR404		
Resources			

Process	Poll image depth and update information specified in Functionality **pollDepth ()**

Table 68: DD04 Depth Wrapper

ID and Title	**DD05 Kinect_Prefab component**		
Identifier	DD05	Type	Prefab
Children		Dependencies	DD01, DD02, DD03, DD04
Input Data		Output Data	
Goal	Object to encapsulate all motion detection.		
Interface			
Functionality	All components must be in the scene. The Kinect_Prefab component will allow DD01, DD02, DD03 and DD04 to be present in the Unity scene.		
References	SR309, SR401, SR404		
Resources	Unity Editor		
Process			

Table 69: DD05 Kinect_Prefab

ID and Title	**DD06 Action component**		
Identifier	DD06	Type	Method
Children		Dependencies	DD03
Input Data	Detected gestures	Output Data	Event
Goal	Check if a gesture is made and spawn an action if defined in the game rules.		
Interface			
Functionality	Receive gestures from DD03 and execute actions as specified.		
References			
Resources	SR101, SR102, SR103, SR104, SR105, SR106, SR107, SR108, SR109, SR110, SR206, SR207, SR208, SR209, SR405, SR406, SR407, SR408		
Process	Check for gestures and spawn movement actions. **UpdateSmoothedMovementDirection ()** Check for gestures and spawn general actions (fire, jump…). Extend existing KeyPressed method. **KeyPressed(t_action)**		

Table 70: DD06 Action Component

5.2.2. Traceability Matrix

In order to illustrate the granularity of the design, the following traceability matrix is appended, showing the relationship between detailed design components and architectural design components.

AD/DD	DD01	DD02	DD03	DD04	DD05	DD06
AD01						
AD02						

Table 71: Detailed Design Traceability Matrix

5.2.3. Component Relation Diagram

Below in Figure 35: Component Relation Diagram an explanation of the relations between all components defined in the Detailed Design section. As you can see, all the body detection is done in the Gesture control subsystem. The Kinect_Prefab englobes all components required to detect the users skeleton and depth. The skeleton wrapper and the depth wrapper require the KinectSensor to function correctly as it

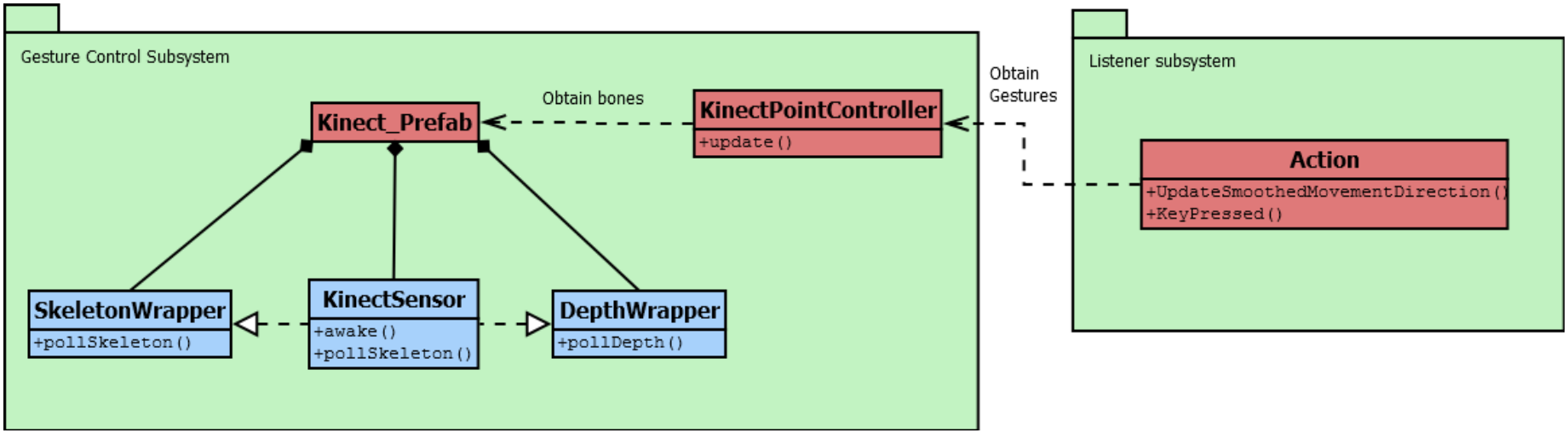

Figure 35: Component Relation Diagram

initializes the Kinect device to be available to the system. The KinectPointController obtains the skeleton from the Kinect_Prefab and analyzes the bone positions and detects user gestures. These gestures are checked by the Listener subsystem (Action component) in order to create new game actions.

6. Implementation

In this section the most relevant implementation issues of the system will be summarized. Previous points specifying the applications' requirements and architecture shall be taken into account to provide the intended functionality.

The implementation is achieved around two objects that will provide all the functionality stated in User Requirement Definition.

6.1. Kinect_Prefab

This object is instantiated after parsing the XML file. It contains all the scripts required to detect the users' joint positions. This object is used to instantiate NUI, allowing the use of Kinect in the application. A depth wrapper is included to detect depth frames and players. Additionally a skeleton wrapper is included to process and poll the users' skeletal position.

6.2. Point_Avatar

This object is created directly after the Kinect_Prefab. The point avatar communicates with the Kinect_Prefab object each frame to know the skeletons position. The point avatar uses the joints provided by the Kinect_Prefab to create an avatar of the user, plotting each joint as a sphere. The purpose of creating the avatar is for testing purposes as we may see what the device is capturing at each instant.

As said above, the point avatar polls the users skeletal position each frame through the Skeletal Wrapper of the Kinect_Prefab. Gesture detection will be made taking advantage of this polling. Obtaining the positions in 3D space of each one of the users' joints allows us to create a vector representation of each one of the users' body parts and detect movements with trivial vectorial calculus. In order to illustrate the implementation a simple movement detection mechanism will be dissected: right arm in a ninety degree angle position.

In order to know if the right arm of the user forms a ninety degree angle, we must obtain the arm and forearm vectors. Once we obtain the vectors, we calculate the dot product of the vectors obtaining the projection of the forearm over the arm.

$$scalarproduct = \overrightarrow{forearm} \circ \overrightarrow{arm}$$

Once we have obtained the scalar product of both vectors, it is trivial to detect a ninety degree position (the projection is near zero).

For correct gesture detection we must give a certain threshold for the system detection as it is improbable that the user will have his arm with an exact ninety degree angle. In order to enhance the usability of movement detection, scalar product values around zero will be detected also as a ninety degree angle gesture. To summarize the ninety degree example:

$$if(scalarproduct < 0.055 \;\&\&\; scalarproduct > -0.055)$$

$$rightarm90 = true$$

Different vector operations are used for gesture capturing, such as vector subtraction and addition.

6.3. Sample Game Definition – Where is Wilson?

In order to show how the GRE model works a sample game has been developed. The objective of the game will be to help the man at the beginning of a maze to find his long lost ball and intimate friend called Wilson. There is fire in the maze and you must run quickly past it or extinguish it with the players' hose. There is an additional secondary mission aimed towards recollecting stars throughout the maze. Finding Wilson, extinguishing a fire and finding a star will give the player points. The XML file defining the game will be explained piece by piece for better comprehension.

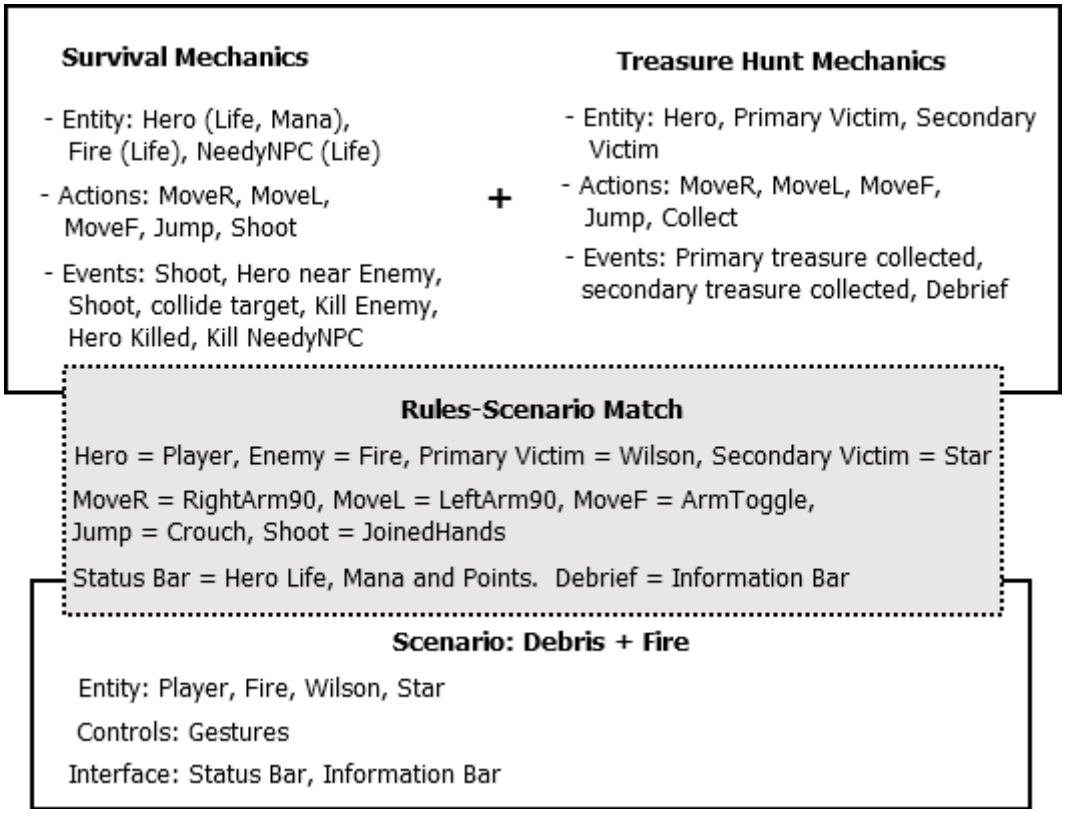

Figure 36: Sample Game

In Figure 37 a general description of the game is given. We can see that we have several entity types classified in different sections: A hero (that haves health and mana), Fire (with health) the NeedyNPC that we will help (with life) and Primary and secondary Victims that will be our objective. These entities are mapped with predefined assets or controls creating the Rules-Scenario matching. This matching is the base of the XML definition of entities, actions and events.

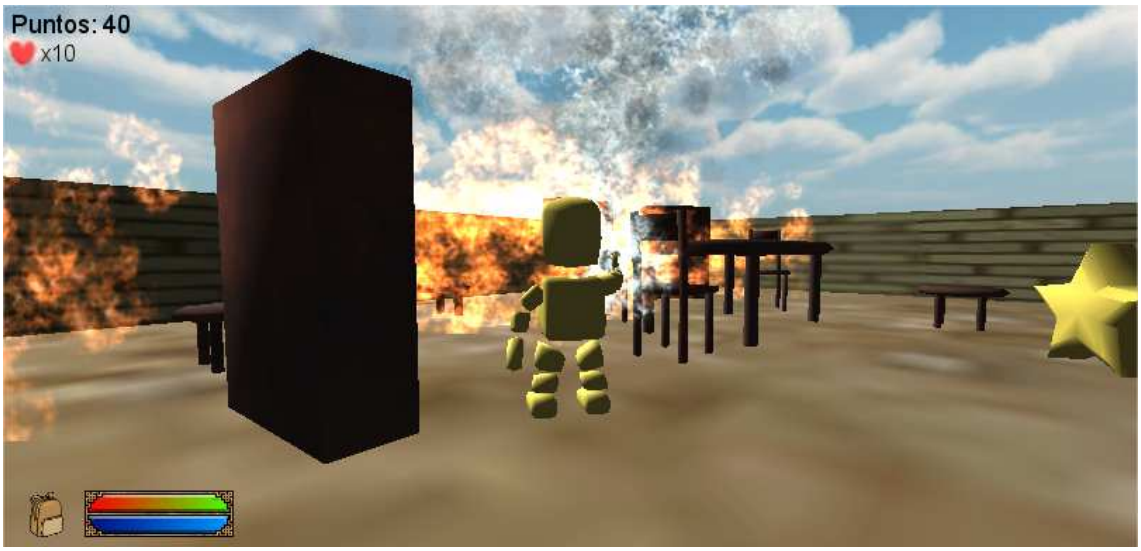

Figure 37: Extinguishing Fire. Hose action.

First of all we will observe the definition of the Hero Entity in the XML file.

```
<main-character name="Player" position="23;0;80" rotation="90">
    <attribute type="health" value="5000"/>
    <attribute type="mana"  value="8000"/>
    <attribute type="hpregen"  value="1"/>
    <actions>
        <action name="kinectMoveLeft" id="actMoveL">
            <control button="kinect" alt="leftArm90" />
        </action>
        <action name="kinectMoveRight" id="actMoveR">
            <control button="kinect" alt="rightArm90" />
        </action>
        <action name="kinectMoveFront" id="actMoveF">
            <control button="kinect" alt="armToggle" />
        </action>
        <action name="jump"  id="actJump" >
          <control button="kinect" alt="crouch"/>
         </action>
        <action name="Hose" id="actHose" >
            <control button="kinect" alt="joinHands" />
            <requires attribute="mana" value="1"/>
        </action>
    </actions>
</main-character>
```

Figure 38: Player XML definition

In Figure 38: Player XML definition we can see that the Hero is defined as a Player object (predefined in Resources as specified in GREP) specifying the position, scale and rotation. The additional attributes, health and mana are included in the entity definition and so are the controls. We can observe that the control definitions match those specified in Figure 36: Sample Game.

The rest of the entities are specified in a similar manner as the player (even simpler, as we do not require controls). Below in Figure 39: Definition of Fire, Wilson and NPC we can see the definition of three additional entity examples, fire (Fuego object), Wilson (pelotafutbol object) and the NeedyNPC (PersonaAyuda object). Each one of these entities have different attributes and are defined according to their requirements.

```
<entity id="Fuego" name="Fuego" pos="99.110;0.505;93.41">
    <attribute type="health" value="3500"/>
    <attribute type="hpregen" value="2"/>
</entity>
<entity id="pelotafutbol" name="pelotafutbol" pos="99.60;0.0045094;105.12">
    <attribute type="canbepicked" maxDistance="2.5" invImage="pelota" invName="Wilson" pickpoints="100"
    <attribute type="label" value="Wilson"/>
</entity>
<entity id="PersonaAyuda" name="PersonaAyuda" pos="38.034;0.0;80.21">
<attribute type="health" value="500"/>
</entity>
```

Figure 39: Definition of Fire, Wilson and NPC

After giving some examples of entity definitions, we proceed to define the events that may occur between these entities. Event definitions will include the entities that perform the interaction and the consequences after the event happens. To give an example of an event, the initial debriefing (Figure 40: Debriefing) given to the player will be explained.

Figure 40: Debriefing

The XML notation for this debriefing is the following:

```
<event id="HelpMe" type="near" distance="14.0">
    <entities>
        <entity-name name="PersonaAyuda"/>
        <entity-name name="Player"/>
    </entities>
    <consequences>
        <consequence type="feedback" feedback-type="sysmessage" value="Help me find my ball Wilson!!
        As reward you can keep the stars you find in my maze!!" />
    </consequences>
</event>
```

Figure 41: Help petition event

The XML notation in Figure 41: Help petition event makes the system create a message each time the player is near the NPC (PersonaAyuda). In the consequence tags you can see the text spawned in the bottom left corner of Figure 40: Debriefing. This event is triggered by proximity between the entities. Another event of this type used is the one in charge of fire inflicting damage to the player.

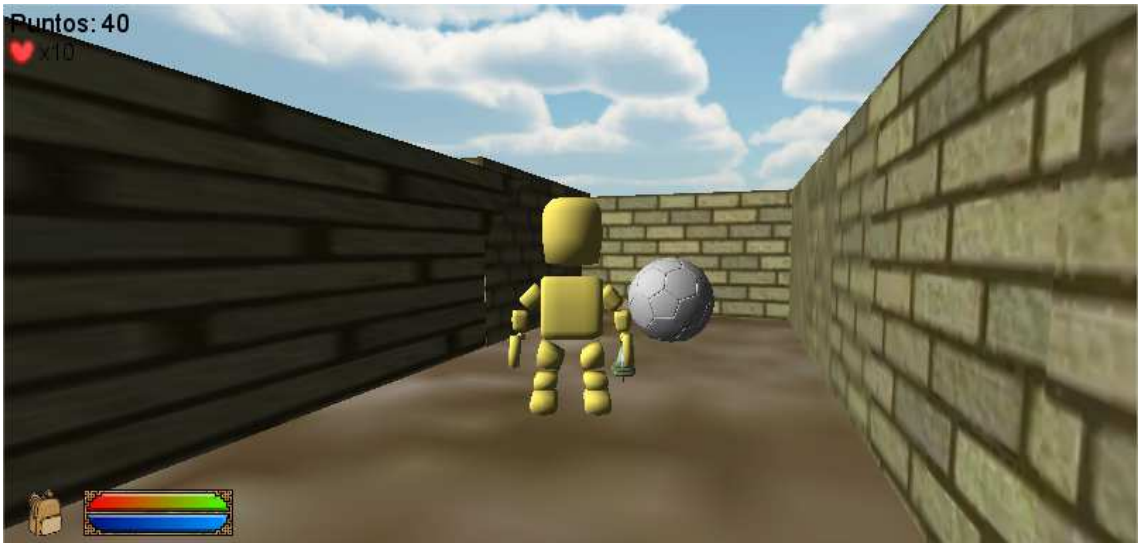

Figure 42: Finding Wilson

Another event type is the one that occurs by collision between two entities. In this game there are three examples, collision between water and fire, collision between the player and stars and the collision between the player and Wilson. A sample definition of this event will be provided below: Fire Water.

```
<event id="FireWater" type="collision">
    <entities>
        <entity-name name="Agua"/>
        <entity-name name="Fuego"/>
    </entities>
    <consequences>
        <consequence type="damage" entity-name="Fuego" attribute="health" value="10000" />
    </consequences>
</event>
```

Figure 43: FireWater XML Definition

The fire water event is an interaction between water and fire (Agua object and Fuego object). The collision of these two entities will suppose a loss of 10000 health attribute points for the fire entity (you can see the fires' health defined in Figure 39: Definition of Fire, Wilson and NPC).

In order for this health loss not to be worthless, an additional attribute event must be defined. An attribute event may be defined for an entity in order to trigger an action after some attribute has been modified. In this sample games there are several attribute events in charge of fire extinguishing, player death and NPC death. The NPC death will be explained for the comprehension of these event types.

```
<event id="PersonDie"  type="attributeValue">
    <entities>
        <entity-name name="PersonaAyuda"/>
    </entities>
    <attributes>
        <attr id="health" value="0" operator="equal"/>
    </attributes>
    <consequences>
        <consequence type="kill" entity-name="PersonaAyuda" />
        <consequence type="feedback" feedback-type="sysmessage" value="You killed an Tom Hanks!!" />
    </consequences>
</event>
```

Figure 44: Kill NPC XML definition

This event will trigger the actions in the consequences tags once the entities health reaches zero. The NPC is killed and feedback is given to the user as seen in Figure 45: Dead NPC.

Figure 45: Dead NPC

7. Validation and Verification

In this chapter the different tests to verify the correct functioning of the application will be presented. On the one hand we have the tests related to movement detection and on the other hand we have the tests related to the gesture-action association. The tests have been developed following a functional point of view, using black box test cases for design components. The tested components can be found in the section Component Description. As the design components comprehend all user requirements, verifying the design components will verify all user requirements as well. Additionally the functionality of the previous version of the application must be checked.

Black box testing is a testing methodology that examines the functionality of the application without taking into account the internal flow of the same. The user requirements of the system to be achieved can be found in the section User Requirement Definition.

7.1. Motion Detection

This section comprises the test cases corresponding to body detection and gesture capturing. Each component related to motion detection will be tested separately to verify its functionality. In order to test the components the following test cases have been defined:

Test Case ID	TB01
Test Item	Skeleton Wrapper::pollSkeleton()
Input	Skeleton position from Kinect Sensor
Output Specification	1. Check if position of joints is correct.

Table 72: TB01

Test Case ID	TB02
Test Item	KinectSensor::awake() KinectSensor::pollSkeleton()
Input	
Output Specification	1. Check if NUI has been correctly initialized by polling new skeleton frames.

Table 73: TB02

Test Case ID	TB03
Test Item	KinectPointController::update()
Input	Skeletal position from Skeleton Wrapper Component
Output Specification	1. Verify that all received joint positions are correct (plot them in game scene). 2. Verify that gestures are detected (can observe this in the Unity Editor during run time).

Table 74: TB03

Test Case ID	TB04
Test Item	DepthWrapper::pollDepth()
Input	
Output Specification	1. Verify that detected players are correct. 2. Verify depth frames of the depth image. 3. Check bounds for the segmentation of each player.

Table 75: TB04

Test Case ID	TB05
Test Item	Kinect_Prefab
Input	
Output Specification	1. Check that Kinect_Prefab object is in the scene and has all its components initialized.

Table 76: TB05

7.2. Gesture-Action Association

In this part the creation of actions is tested. For an action to be spawned, the associated gesture must be made. There is only one component in charge of actions so only this component shall be tested in this section. In order to test the component the following test cases have been defined:

Test Case ID	TB06

Test Item	Action::UpdateSmoothedMovementDirection() Action::KeyPressed()
Input	Detected gestures from the KinectPointController component.
Output Specification	1. Check that gestures received are correct. 2. Check that the intended action is spawned. 3. Verify that Kinect actions are correctly parsed on game selection.

Table 77: TB06

7.3. Compatibility

In this section the test cases to check that the previous version of the application is still functional:

Test Case ID	TB07
Test Item	Action::UpdateSmoothedMovementDirection() Action::KeyPressed()
Input	Keys pressed, mouse clicked.
Output Specification	1. Check that events received are correct. 2. Check that the intended action is spawned. 3. Verify that actions are correctly parsed on game selection.

Table 78: TB07

8. Conclusions

Once finished the development of the application, we will focus on whether the goals specified at the introduction of this document have been fulfilled. Additionally, the project management will be reviewed to discuss schedule changes and overall compliance with the planning. Below we will review point by point the goals of the project taking into account the accomplishments in each section.

8.1. Gesture Based Human-Computer Interaction

One of the main purposes of this project before this extension was to provide a simple tool for educators to teach using videogames. The focus of making the development in a simple way should also be accompanied by a simple way to interact with the system so the players focus more on learning the message provided by the educator rather than game controls.

This goal has been widely achieved providing a gesture based HCI study and developing the solution according to this study.

8.2. Gesture Recognition

Taking into account the previous point, we need to develop a solution that is able to recognize user gestures to control the application. The project is able to successfully capture eleven gestures. Each one of these gestures can be assigned to any action present in the game.

To develop the gesture recognition, a study has been made thoroughly in order to decide what device would be the chosen to achieve the best gesture based HCI solution. An extensive comprehension on videogame visual input devices was delivered due to this study on Nintendo Wiimote, Playstation Move and Microsoft Kinect.

The final selection of Microsoft Kinect provided knowledge on development of motion detection applications for Windows. The implementation given with Kinect for Windows SDK from Microsoft allowed to appreciate the real potential of the Kinect device (camera functioning, skeleton detection and depth detection). Although it has not been used in this project, the options it allows for speech recognition with different language packs available.

8.3. Project Management

This project has been profiled to meet a deadline for each one of the development phases. The initial planning for this project can be found in the section Delivery Time and Initial Planning. In this section the deviation of the project with respect to this initial planning will be discussed. The following Gantt chart was made with SmartDraw 2014 [28].

Figure 46: Final Planning

In Figure 46: Final Planning we can see the actual planning that was followed during the realization of the project. If we compare it with the initial planning Gantt chart in Delivery Time and Initial Planning, we observe that the planning phase was overextended eleven days, the analysis and implementation were overextended one day, the testing phase was overextended two days, and the architecture and design was done in the stipulated time.

We can also observe that the implementation milestone that both tutor and student set was missed, having a total delay of eighteen days. When designing the planning for the project this possibility was taken into account so the consequences (due to the time until the official hand in milestone) were not important for the development of the project.

The delays are minor in most cases, but it is interesting to discuss the cause of the major delay, caused in the planning phase. The loss of eleven scheduled days was caused by issues with the purchase of the Kinect device. The lack of a cable required to connect the device with a computer (instead of an Xbox), caused an additional delay of eleven days due to the shipping of this piece.

The rest of the delays were caused by issues that could not have been predicted in any way (student exams, course practical hand-ins…), so from my point of view the planning has been designed and followed correctly as we had already provided a

month between the implementation milestone and the hand in milestone in case any problems occurred.

The planning could have been improved if the implementation milestone had been met, although it was quite ambitious. I believe that the main success of this planning was the contemplation of a certain time after the implementation to add new features, additional documentation, or absorb delays from the other phases. From a point of view regarding objectives and goals, the planning served to finish the project before the stipulated deadline, hence it rendered itself useful.

8.4. Future Work

Given the capabilities of the GRE model and the existing studies related to human-computer interaction there are many possibilities of extensions related to the existing project. The current implementation may be improved regarding the Gesture interface and usability for educators and players. Several additional areas will be stated summarizing only some of the improvements available to enhance the solution.

- **Additional Gestures**: Given the architecture following the GRE model, the definition of new gestures and association with game actions is rendered trivial. A collaborative approach to gesture definition could be adopted, creating a social community able to share new gestures. In order to do so, a graphic interface for gesture definition could be provided to make gesture definition easy for educators.
- **Implementation of Iconic Gestures**: Due to the scope of this project, iconic gestures (as seen in the section Human-Computer Interaction. Gestures.) have not been implemented. In order to define iconic gestures speech recognition must be implemented in the solution. The project has been developed regarding this possibility and gives freedom for further development in this area with the simplest library possible in order to do so (Kinect for Windows).
- **Graphic interface for game definition**: At the moment educators must define games directly in an XML file. With the purpose of making game definition simpler for non specialized educators a graphic interface could be made in order to create the XML file, providing rules and scenario definition in a simple and intuitive manner. The association of actions with the gesture based interface would be easier to do with this graphic interface, allowing the educator to see a preview of the gesture to be captured.
- **System controls through the gesture interface**: Due to the architecture of the solution, system controls are not defined in the XML file. This could be extended for the use with the gesture based interface, being able to close or pause the game through gestures.

- **Multiplayer**: It would be interesting to implement a multiplayer solution to allow a better educational experience, being able students to play with other classmates. The current project allows the development of an extension providing the multiplayer improvement using the gesture based interface using only one Kinect device.

8.5. Personal Conclusions

In this section the individual abilities gained by the author will be reviewed. The main knowledge provided with the development of this project is focused mainly on technical and organizational aspects of the design and implementation of the solution using Kinect.

Personally, the project gave the formation to the author on Human-Computer Interaction in an academic manner that was not found before the implementation. Research on the subject was crucial for the design of the interface for the existent project.

Another point considered important was the additional information absorbed by studying the possible solutions to provide a Gesture based interface and the motion detection tools investigated in order to do so. The technical abilities gained developing for Kinect using the Kinect for Windows SDK on the Unity Game Engine, unknown before the implementation of this project, gave experience that could have been difficultly acquired out of the environment of this project.

The full potential of the GRE model was seen in a practical manner, comprehending the existent solution in less than a month by an inexperienced programmer as the author. The implementation of the extension was straightforward once the game architecture was understood.

Regarding the project planning, the problem of having fixed deadlines was acknowledged and the utility of the planning phase was made evident. Delays spawned and rescheduling was required to be done. I believe this experience is very useful due to the necessity of including a correct 'padding' time after each phase in case any unexpected issues occur in order to avoid deadline rescheduling. The creation of a small Kinect demo in the planning phase to study the capabilities of the device was very useful for the implementation, as the basic technical implementation skills were achieved before the actual development phase.

Finally, the experience writing this document is specially appreciated, gaining research capability and formal document creation. Special attention has been made in the document content following citation standards [34] and others such as IEEE-830 [33].

References

[1] Gaudiosi, J. (2012, July 18) New Reports Forecast Global Video Game Industry Will Reach $82 Billion By 2017. Message posted to http://forbes.com (http://www.forbes.com/sites/johngaudiosi/2012/07/18/new-reports-forecasts-global-video-game-industry-will-reach-82-billion-by-2017/)

[2] Squire, K. (2003) Video Games in Education. Games & Simulation. (http://f3program.org/sites/all/files/eg/20082009/mmorpg/Research%20-%20Video_Games_in_Education-MIT_Study.pdf)

[3] Papastergiou, M. (2009, November) Exploring the potential of computer and video games for health and physical education: A literature review. In Computers & Education, Volume 53, Issue 3, (pp 603-622)

[4] Prensky, M. (2003, October 1) Digital game-based learning. In Computers in Entertainment (CIE) – Theoretical and Practical Computer Applications in Entertainment, Volume 1 Issue 1 (pp 21-21)

[5] Warman, M. (2011, June 8) Average Video gamer is 37. Message posted to http://www.telegraph.co.uk (http://www.telegraph.co.uk/technology/video-games/8564342/Average-video-gamer-is-37.html)

[6] History of Human Computer Interaction. Extracted from course slides of Saul Greenberg from the University of Calgary. (http://pages.cpsc.ucalgary.ca/~saul/hci_topics/pdf_files/history.pdf)

[7] Telmo Zarraonandia, Paloma Díaz, Ignacio Aedo, Mario Rafael Ruíz (2011) : "Seeking Reusability of Computer Games Designs for Informal Learning" in Athens, GA ,Proceedings of ICALT ,Ed: IEEE ,455-459.

[8] DJ Sures demonstrates how Wiimote can be added to EZ-Builder and configured to control a servo. (http://www.youtube.com/watch?v=jS2SGFvF5iw)

[9] Homebrew is a framework that allows application development for Wii and the Wiimote (http://wiibrew.org/wiki/Making_homebrew)

[10] Denmead, K. (2011, May 24). Controlling Robots With the Playstation Move (Maker Faire 2011). Message posted to http://www.wired.com. (http://www.wired.com/geekdad/2011/05/controlling-robots-with-the-playstation-move-maker-faire-2011/)

[11] Move.me is a website that is essential for Playstation Move developers. (https://us.playstation.com/ps3/playstation-move/move-me/)

[12] The Kinect SDK is a development tool launched by Microsoft to allow developers program the Kinect device. (http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx)

[13] Lee, K. (2012 April 4). The Kinect Project Uses Your Tongue as the Controller message posted to http://techhive.com. (http://www.techhive.com/article/253199/this_kinect_project_uses_your_tongue_as_the_controller.html)

[14] Unity3D faq page. Unity licensing. (consulted 6/6/2013). (http://unity3d.com/unity/faq)

[15] Kinect Fusion provides 3D object scanning and model creation using a Kinect for Windows sensor. (http://msdn.microsoft.com/en-us/library/dn188670.aspx)

[16] Kinect for Windows Release Notes. (http://msdn.microsoft.com/en-us/library/jj663803.aspx)

[17] Java Wrapper for OpenNI. (https://github.com/almerindo/OpenNI2/tree/master/Wrappers)

[18] Python Wrapper for OpenNI. (https://github.com/jmendeth/PyOpenNI)

[19] C# Wrapper for OpenNI. (https://github.com/OpenNI/OpenNI/tree/unstable)

[20] Hinchman, W. (2011, June) Kinect for Windows SDK beta vs. OpenNI. (http://labs.vectorform.com/2011/06/windows-kinect-sdk-vs-openni-2/)

[21] Kiili, K (2005). Participatory multimedia learning: Engaging learners. Australasian Journal of Educational Technology. 21(3), 303-322.

[22] Amory, A., & Seagram, R. (2003). Educational Game Models: Conceptualization and Evaluation. Journal of Higher Education. 17(2), 206 – 217.

[23] Sweetser, P., & Wyeth, P. (2005, July). GameFlow: A Model for Evaluating Player Enjoyment in Games. ACM Computers in Entertainment, Vol. 3, No. 3, July 2005, Article 3A.

[24] Zarraonandia, T., Díaz, P., Aedo, I., & Ruiz, M.R. (2011) A Model for Designing Reusable and Adaptable Educational Games.

[25] Billinghurst, M. (2011, August) Gesture Based Interaction. In Buxton, B., Haptic Input (chapter 14). Obtained from Buxton, B. personal website. (http://www.billbuxton.com/input14.Gesture.pdf)

[26] Rime, B., & Schiaratura, L. (1991) Gesture and Speech. In Feldman, S., & Rime, B. Fundamentals of Nonverbal Behaviour (pp 239-281).

[27] Professional Website of Telmo Zarraonandia Ayo. (http://www.inf.uc3m.es/es/component/comprofiler/userprofile/tzarraon)

[28] SmartDraw is a program to create fancy graphs. (http://www.smartdraw.com/)

[29] Dia is a tool for creating charts. (http://projects.gnome.org/dia/)

[30] InfoJobs Trends is a website to find average wages in Spain. (http://salarios.infojobs.net/index.cfm)

[31] Rodrigo Martínez, I., & Villarroya Berges, C. (2001) Costes Indirectos de la Investigación Bajo Contrato en la Universidad Española. (http://www.uji.es/bin/ocit/art/overhead.pdf)

[32] Barba, D. (September, 2012) Desarrollo de un juego educativo de lógica configurable.

[33] (1998) IEEE 802 is a standard for requirement definition. Retrieved from the University of Alaska Anchorage. (http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf)

[34] APA citation style refers to the rules and conventions established by the American Psychological Association for documenting sources used in a research paper. Retrieved from the Cornell University online Library. (http://www.library.cornell.edu/resrch/citmanage/apa)