ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

DESARROLLO DE UN JUEGO EDUCATIVO EN XNA

Leyre Ramírez Ervíti

Benoît Bossavit, Alfredo Pina

Pamplona, 11/09/2015

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

INDICE DE CONTENIDOS

1 Introducción 3
1.1 Motivación 3
1.2 Objetivos 3
1.3 Organización de la memoria 4
1.4 Conceptos básicos de 3D 5
2 Estado del arte 8
2.1 Historia de los videojuegos 8
2.2 Videojuegos en la actualidad 19
2.3 Juegos educativos 21
2.3.1 Juegos educativos para niños discapacitados 25
2.4 Software para la creación de videojuegos 26
3 Herramientas utilizadas 29
3.1 Microsoft XNA Game Studio 29
3.1.1 ¿Qué es XNA? 29
3.1.2 Ventajas de usar XNA 30
3.1.3 Goblin XNA 31
3.1.4 Kinect SKD para Windows 32
3.2 Microsoft Visual Studio Express 33
3.2.1 C# 34
3.3 Maya 35
3.3.1 ¿Qué es Maya? 35
3.3.2 Ventajas de usar Maya 36
3.4 GIMP 37
3.5 Photoshop CS3 38
3.6 Sculptris 39
4 Análisis del proyecto 40
4.1 Descripción general 40
4.2 Descripción detallada 42
4.2.1 Safe Trip 42
4.2.2 ¡Sonríe! 51
5 Safe Trip 54
5.1 Diseño del proyecto 54
5.2 Implementación del proyecto 60
5.2.1 Modelado 60
5.2.2 Texturizado 67
5.2.3 Animación 71
5.2.1 Formato de los ficheros 76
6 Sonríe 77
6.1 Diseño del proyecto 77
6.1.1 Diseño Grafico 77

6.1.2 Modulado de la Arquitectura 78
6.1.2.1 Identificando los métodos 78
6.1.2.2 Relación entre métodos 79
6.2 Implementación del proyecto 80
6.2.1 Formato de los ficheros 80
6.2.2 Cámara y marcadores 81
6.2.3 Implementación del código 83
6.2.3.1 Estructura básica de un juego en XNA 83
6.2.3.2 Implementación del código 84
7 Gestión del proyecto 90
7.1 Ciclo de vida del proyecto 90
7.1.1 Sonríe 90
7.1.2 Safe Trip 91
7.2 Coordinación 92
7.3 Planificación del proyecto 93
7.3.1 Sonríe 93
7.3.2 Safe Trip 95
8 Conclusiones y líneas futuras 97
8.1 Conclusiones 97
8.2 Líneas futuras 98
9 Bibliografia 99

1 Introducción

1.1 Motivación

Cada vez es más frecuente en la actualidad el uso de juegos o videojuegos como métodos de aprendizaje. La combinación de creatividad, diversión y contenido educativo que tienen estas herramientas, hace mucho más sencillo y dinámico el proceso de asimilación de datos.

Los juegos educativos refuerzan la curiosidad en los niños y facilitan su futuro desempeño académico. Además, ayudan a desglosar tareas complejas, utilizando pequeños pasos que nos permiten una mejor comprensión de los problemas. Una característica importante de un videojuego educativo es que el conocimiento es adquirido de una forma implícita, es decir que los jugadores no se percatan que al estar jugando van adquiriendo una serie de conocimientos concretos, sino que se van apropiando de estos en el transcurso natural del videojuego.

1.2 Objetivos

Los principales objetivos para la realización de este proyecto son:

- Hacer una aportación a los métodos actuales de aprendizaje desarrollando una aplicación que sirva de ayuda a niños que por diversos motivos no pueden aprender a realizar tareas cotidianas de la manera habitual.

- Por otro lado, adentrarnos en el mundo de los videojuegos, aprender a diseñar e implementar un videojuego en su totalidad, y poder comprender y saber desarrollar cada una de sus etapas.

1.3 Organización de la memoria

En este apartado se muestra de una forma rápida y breve la lista de capítulos que se podrán encontrar en este documento, y de qué partes consta cada uno de ellos:

- **Capítulo 1 – Introducción:** En este apartado se reflejan las motivaciones que hicieron posible el proyecto, y los objetivos que se persiguen con éste. De igual forma se describe la estructura del presente documento.

- **Capítulo 2 – Estado del arte:** Este capítulo incluye lo relacionado al marco tecnológico donde se enmarca este proyecto, una descripción sobre la historia de los videojuegos, el estado actual de la industria de los videojuegos, algunas plataformas educativas de la actualidad y una introducción al software necesario para la creación de un videojuego.

- **Capítulo 3 – Herramientas utilizadas:** En este capítulo se introducen las herramientas utilizadas en este proyecto explicando en algún caso la importancia de las mismas en el desarrollo de los juegos.

- **Capítulo 4 – Análisis del proyecto:** El capítulo se compone de una descripción general de los dos proyectos realizados así como una descripción detallada de cada uno.

- **Capítulo 5 – Safe Trip:** En este capítulo se desarrolla el diseño y la implementación del juego *Safe Trip* haciendo hincapié la creación de los objetos 3D.
- **Capítulo 6 – Sonríe:** En este capítulo se desarrolla el diseño y la implementación en su totalidad del juego *Sonríe.*

- **Capítulo 7 – Gestión del proyecto:** En este apartado se desarrolla el ciclo de vida de los proyectos y su planificación añadiendo información sobre la coordinación del equipo de desarrollo.

- **Capítulo 8 – Conclusiones y trabajo futuro:** En este capítulo se incluyen las conclusiones al proyecto, sus aportaciones, y las posibles ampliaciones al proyecto actual.

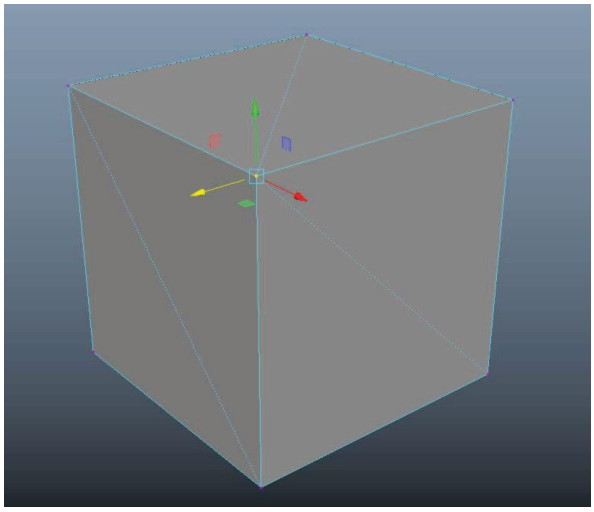
- **Capítulo 9 – Bibliografía:** Recursos bibliográficos y referencias utilizadas en la elaboración del proyecto.

1.4 Conceptos básicos de 3D

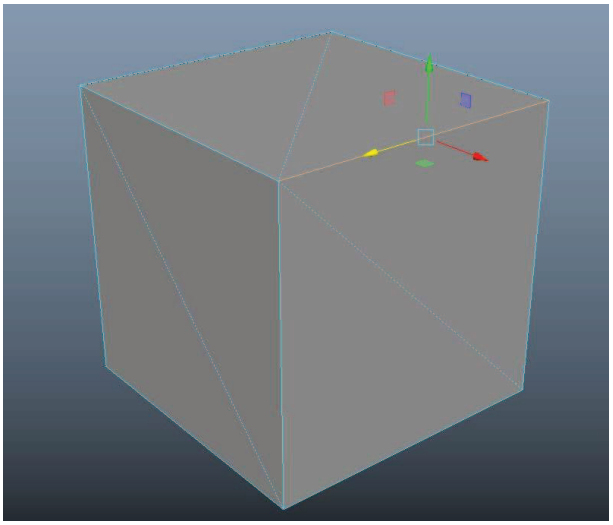
Dado que más adelante se explicara la creación de estructuras 3D, es importante que antes de llegar a ese apartado se tengan unas nociones básicas de 3D, para de esta forma poder entender todo sin problemas.

Los objetos 3d están construidos a partir de tres estructuras básicas: vértices, aristas y caras.

Un vértice es un único punto o una posición en el espacio 3D. Es la estructura de más bajo nivel en un objeto 3D.

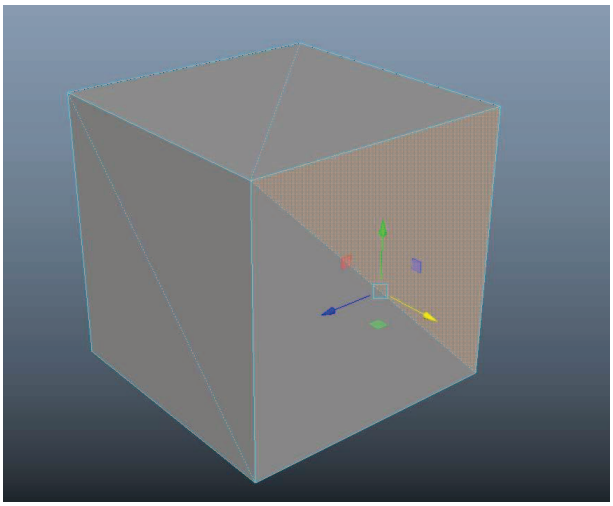


Una arista es una línea recta que conecta dos vértices.



La cara es la estructura de más alto nivel en una malla 3D. Estas son utilizadas para construir la superficie real de un objeto.

Una cara es el área entre tres o cuatro vértices, con una arista en cada lado. Cada cara de 4 lados siempre está formada por dos caras de tres lados.



Los objetos o modelos 3D están clasificados por polígonos, al igual que todo elemento que tenga una composición tridimensional, siendo un polígono una figura geométrica plana que está limitada por tres o más aristas y tiene tres o más vértices. Por lo general, se usan polígonos limitados por solo tres aristas para contabilizar el número de polígonos que conforman un objeto.

De esta forma, tenemos tres tipos de objetos 3D:

- **Objetos Low-Poly:** Se trata de modelos cuya composición de polígonos es baja. Estos modelos tienen muy poco detalle, pero resultan una carga pequeña para el motor. Son usados sobre todo para videojuegos que se ejecutaran en ordenadores poco potentes, o que tengan una carga de procesamiento, ya sea porque hay una gran cantidad de objetos o por cualquier otro motivo. También se usan el objetos que solo vayan a ser vistos desde una gran distancia.

- **Objetos Mid-Poly:** Se trata de modelos cuya composición de polígonos es media. Estos objetos logran dar un mejor detalle que los "Low-Poly" pero su velocidad de procesamiento es más lenta. Estos objetos son más usados para modelos que requieren un poco más de detalle.

- **Objetos High-Poly:** Se trata de modelos cuya composición de polígonos es alta. Estos modelos tienen una calidad muy alta y muchos detalles, pero su procesamiento es más complejo y tiende a ralentizar el ordenador. Estos modelos son usados para modelos que necesitan mucho detalle. Se usan sobre todo en las películas de animación.

En función de las características que tenga la superficie del objeto que se desea crear, se pueden distinguir dos tipos de modelado:

- **Modelado inorgánico:** Llamamos modelado inorgánico a la creación de objetos 3d cuyas superficies sean mayormente planas y ordenadas, normalmente estos objetos imitan a objetos reales construidos por el hombre. Ejemplos de modelado inorgánico serian edificios, vehículos…

- **Modelado orgánico:** Llamamos modelado orgánico a la creación de objetos 3d cuyas superficies sean irregulares, curvas y caóticas. Estos objetos suelen imitar a seres vivos. Ejemplos de modelado orgánico serian plantas, animales, personas…
  El modelado orgánico es más complejo que el modelado inorgánico.

2 Estado del arte

2.1 Historia de los videojuegos

Si bien es cierto que comparando con otras artes como el cine o la música, la historia de los videojuegos es relativamente muy corta, su evolución y expansión han sido enormes en las últimas tres o cuatro décadas.

Tanto es así que a pesar de la última crisis española, se trata de un sector que ha continuado incrementando sus ventas, superando en ingresos a la industria musical y cinematográfica juntas. Se trata, por tanto, de una industria que puede aportar grandes beneficios económicos por lo que cualquier país desarrollado debería tenerla en cuenta.

La evolución de esta industria, ligada siempre al avance de la tecnología, es más que notable. En sus poco más de 50 años de vida se ha pasado de pequeños desarrollos caseros durante algunos días o semanas con no demasiados recursos, a grandes multinacionales cuyo desarrollo de videojuegos puede alargarse durante años y sus presupuestos pueden llegar a ascender a decenas de millones de Euros.

A día de hoy aún no queda claro exactamente cuál fue el primer videojuego de la historia. Si buscamos en internet encontraremos muchas referencias que afirman que fue el famoso *Pong* de Atari en 1972. Sin embargo, encontramos otros juegos años atrás como un juego de lanzamiento de misiles (1947), el tres en raya electrónico (1952), *Tennis for two* (1958) o *Spacewar* (1961). Pero muchos coinciden que estos últimos juegos no pueden considerarse videojuegos como tal (aunque existen discrepancias al respecto) bien por la falta de movimiento en la pantalla, bien por usar circuitería en lugar de un ordenador o bien por no tener una gran acogida.

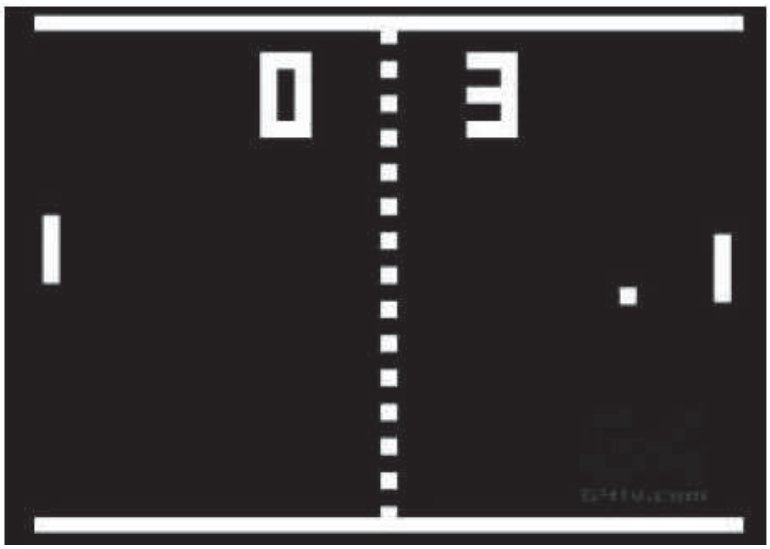


Figura 1: *Pong* de Atari (1972)

Tras la aparición del *Pong*, los videojuegos evolucionan a lo largo de los años de manera notable y rápida. Es así como nos encontramos con el que es, posiblemente, el primer juego de plataformas: el famoso *Donkey Kong* (1981). Tuvo un éxito rotundo y en él pudimos ver por primera vez al personaje "Mario", conocido inicialmente como "Jumpman", y que aún hoy en día sigue protagonizando diferentes juegos.

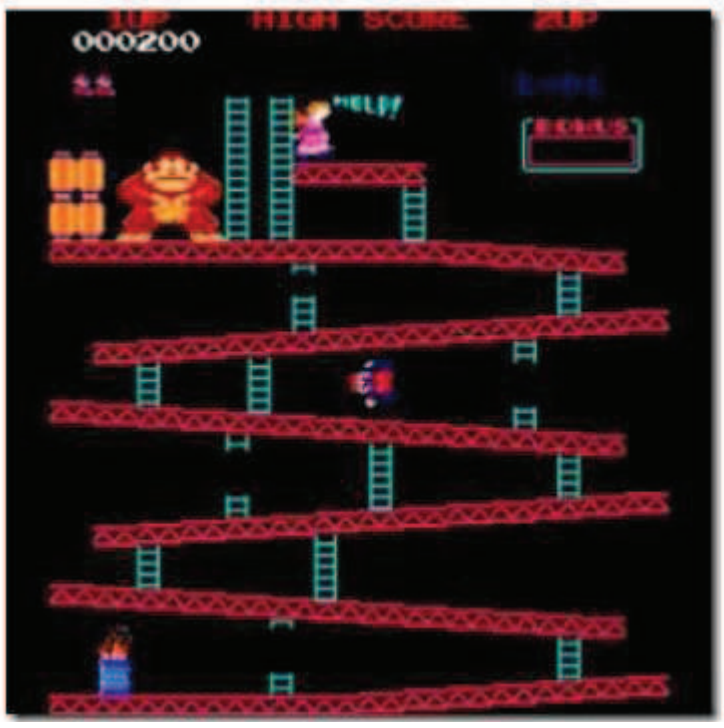


Figura 2: *Donkey Kong* (1981)

Aún pasarían unos pocos años antes de que saliera el videojuego de "Mario", llamado *Mario Bros* (1983). Sin embargo, no fue este el que llevó al personaje al estrellato. Fue *Super Mario Bros* (1985) que lo convirtió en la mascota de la compañía japonesa Nintendo.

>>

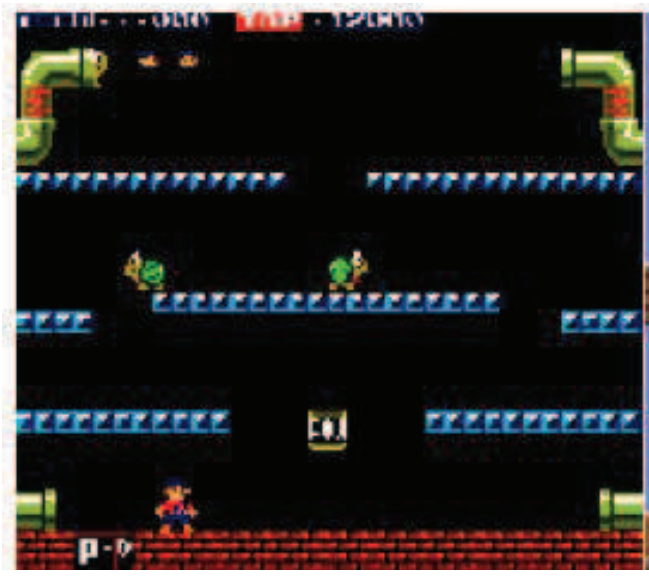


Figura 3: ***Mario Bross*** **(1983)**

Figura 4: ***Super Mario Bross*** **(1985)**

Situándonos ya en la época de los 90, Nintendo sigue produciendo videojuegos de éxito debido al personaje de "Mario", como *Super Mario World* (1990). Sin embargo, Sega consigue hacerle frente con la creación de *Sonic* (1991). Un videojuego que se distinguía de *Super Mario* en su gran velocidad y dinamismo, convirtiéndose en la nueva mascota de la compañía y en un gran éxito mundial.

Figura 5: *Sonic* (1991)

Un año más tarde aparece la segunda parte de *Sonic*, *Sonic 2* (1992), que incorporaba importantes novedades respecto a su predecesor, como el personaje del zorro *Tails*, además de mayor rapidez, niveles y dificultad. El éxito fue tal que se convirtió en el videojuego más vendido de Sega.

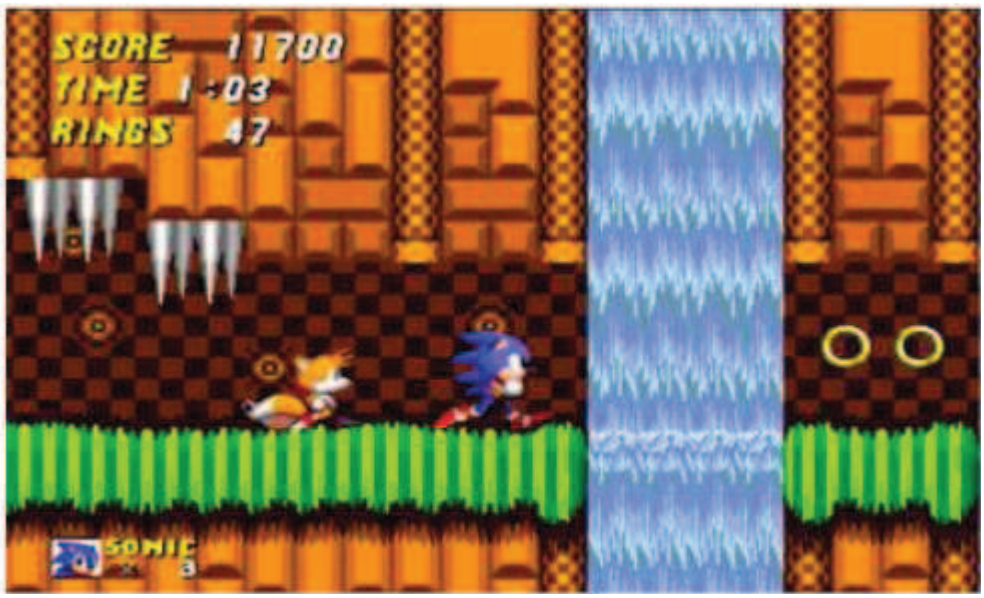


Figura 6: *Sonic 2* (1992)

Estos juegos anteriores no dejaban de ser en dos dimensiones. Es a partir de 1996 que empiezan a salir al mercado las versiones 3D tanto de *Mario* como de *Sonic*. La acogida fue muy buena pese al cambio que podía suponer el pasar de 2D al 3D. *Super Mario 64* llegó a vender más de 12 millones de copias.

Figura 7: ***Super Mario 64***

Figura 8: ***Sonic 3D***

Dejando atrás esta disputa entre Nintendo y Sony, entramos en el siglo XXI, y si hay algo que caracteriza la industria del videojuego de este siglo es su transformación en una industria multimillonaria de dimensiones inimaginables pocos años antes. En 2009 la industria de los videojuegos era uno de los sectores de actividad más importantes de la economía estadounidense. Como ya se ha comentado anteriormente, en España generaba más dinero que la industria de la música y el cine juntos.

El programador de videojuegos dejó de ser un aficionado a la electrónica que elaboraba en solitario con carácter artesanal y amateur sus programas. Pasó a ser un profesional altamente cualificado que trabajaba con otros profesionales especializados (grafistas, programadores, diseñadores, testers…) en equipos de desarrollo estructurados y a menudo bajo control directo o indirecto de grandes multinacionales.

Paralelamente surgió un desarrollo muy rápido de medios donde reproducir los videojuegos. Desde mediados de la década de 1990 la industria de las máquinas recreativas estaba en crisis. Habían siempre basado su éxito en disponer de tecnología y potencia audiovisual por encima de las capacidades de los microordenadores personales y las consolas domésticas. El éxito de la *PlayStation* y las consolas de su generación (*Nintendo 64* y *Sega Saturn*) desequilibró definitivamente la situación.

Figura 9: *PlayStation*

Se trataba de máquinas domésticas que igualaban e incluso superaban tecnológicamente a la mayoría de recreativas.

Al mismo tiempo, el rápido crecimiento del sector de la telefonía móvil, con aparatos cada vez más potentes que permitían ejecutar videojuegos de creciente complejidad, sentenció definitivamente las máquinas que habían protagonizado el inicio de la revolución de los videojuegos.

En 2000 Sony lanzó la anticipada *PlayStation 2* un aparato de 128 bits que se convertiría en la videoconsola más vendida de la historia, mientras que Microsoft hizo su entrada en la industria un año más tarde con su *X-Box*, una máquina de características similares pero que no logró igualar el éxito de la primera.

Figura 10: ***PlayStation 2***

Figura 11: ***X-Box***

Mientras tanto, el mercado de los PC seguía dominado por esquemas de juego que ya habían hecho su aparición con anterioridad. Triunfaban juegos de estrategia en tiempo real (*Warcraft*, *Age of Empires*, ...) y los juegos de acción en línea (*Call of Duty*, *Battlefield*, …).

Figura 12: ***Age of Empires***

En 2004 salió al mercado la *Nintendo DS*, primer producto de la nueva estrategia de la compañía renunciando a las videoconsolas clásicas. Poco después apareció la *Sony PSP*, consola similar que no llegó a alcanzar a la primera en ventas. En 2005 Microsoft lanzó su *Xbox 360*, un modelo mejorado de la primera y diseñado para competir con la *PlayStation 2*. Sin embargo, la respuesta de Sony no se hizo esperar y pocos meses después lanzó la *PlayStation 3*. La revolución de Nintendo llego en abril de 2006 cuando presentó su *Wii*, que presentaba un innovador sistema de control por movimiento y sencillos juegos que puso de nuevo a la compañía en el lugar que le correspondía dentro de la historia de los videojuegos.

Figura 13: ***Wii***

Para muchos aficionados, la profesionalización de la industria trajo consigo cierto estancamiento de la originalidad que había caracterizado el trabajo de desarrolladores de épocas anteriores. A pesar de ello, las compañías continuaron lanzando títulos que suponían un soplo de aire fresco, como *Guitar Hero* (2005) con un original sistema de control que abrió una lucrativa franquicia y cuyas ventas en 2009 se situaban en 32 millones de juegos vendidos. *The Sims* (2000) puso de moda los juegos de simulación social. Por su parte, *Grand Theft Auto III* (2001) iniciaba una serie de videojuegos que mezclaban dos de las tendencias más características de las nuevas generaciones: argumentos cada vez más complejos y libertad de movimientos y de acción.

Figura 14: *Sims*

Una innovación que llegó a la par que las conexiones a internet en los hogares fue la de los juegos en línea, en los que los jugadores que se encontraban en sus casas podían conectarse entre ellos a través del juego.

A Ultima Online (1997) se le acredita la popularización del género. El juego se caracterizaba por un abono de suscripción mensual en vez del tradicional pago por hora.

Figura 13: ***Ultima Online***

Para finales de los 90 el concepto de juegos en línea multijugador masivos había traspasado las fronteras hacia nuevos géneros como los juegos de estrategia o los juegos de acción en primera persona. Surgió la etiqueta MMOG (massively multiplayer online game) para agrupar a todos ellos con independencia de su género.

El 23 de noviembre de 2004 salió a la venta World of Warcraft, creado por la compañía Blizzard. Este juego batió récords en su género, ya que llegó a superar la increíble cifra de 12 millones de jugadores. World of Warcraft destaca por su facilidad de uso, su interactividad y por los escenarios gigantescos sin cargas que conforman un universo continuo.

Figura 14: ***World of Warcraft***

Otro fenómeno que ha caracterizado las prácticas de las compañías desarrolladoras en los últimos años ha sido la explotación de franquicias, series de programas basados en los personajes de un videojuego original. De esta forma encontramos títulos como *Halo: Combat Evolved* (2001), *Resident Evil* (1996), *Silent Hill* (1999), *Prince of Persia* (2003), *The King of Fighters* (1994), *Final Fantasy* (1987), *Street Fighter* (1987), *Call of Duty* (2003), *Metal Slug* (1996), *Medal of Honor* (2001) o las ya citadas *Guitar Hero* (2005) o *Grand Theft Auto III* (2001). Estos juegos dieron lugar a una interminable serie de secuelas que en muchos casos basaban su éxito no tanto en sus innovaciones como en el hecho de compartir su nombre con el del hit original.

Figura 13: ***Final Fantasy***

A principios de 2011 se asiste a una nueva era de creatividad gracias tanto a superproducciones de las grandes compañías multinacionales como a los esfuerzos de innovación de los desarrolladores más pequeños. Medio siglo después de que los primeros creadores de juegos experimentaran la sensación de estar jugando con una computadora, el concepto de videojuego se ha ido desarrollando con el tiempo para acabar convertido en un medio integral de entretenimiento que puede producir experiencias muy diversas.

Lejos de haber alcanzado su madurez creativa, los videojuegos siguen siendo una nueva forma de arte que parece estar dando aún, 50 años después de su aparición, sus primeros pasos.

2.2 Videojuegos en la actualidad

La actual generación de videojuegos, conocida como la "Octava Generación" comprende una gran diversidad de plataformas como el PC, las consolas más recientes desarrolladas por las principales Compañias: la PlayStation 4 de Sony, Xbox One de Microsoft y Wii U de Nintendo, además de diversas consolas portables (PS Vita, Nintendo 3DS…) y plataformas móviles iOS y Android.

Esta nueva generación de videojuegos ha dado un salto enorme en la fidelidad visual que es posible obtener en un videojuego debido al potente hardware disponible actualmente. Esta potencia ha permitido, sobre todo, elevar en número de polígonos (vértices y aristas que conforman un objeto

3D), así como mejorar la calidad de efectos de iluminación, y efectos de postprocesado, que en conjunto, permiten obtener una calidad gráfica muy superior a la existente en generaciones anteriores.

Además, en esta generación se han introducido grandes cambios en la forma que tiene el usuario de controlar o manejar un juego gracias a tecnologías como Kinect de Xbox o PlayStation Eye, que son capaces de recoger los movimientos corporales del usuario mediante un sistema de cámaras y utilizarlos como entrada a la hora de manejar lo que ocurre dentro del juego.

Kinect de Microsoft

A esta tecnología se le unen otras, que aunque todavía están menos desarrolladas, parecen tener un futuro realmente prometedor, como son Oculus Rift, Steam VR (HTC Vive) o Sony Morpheus. Estas tecnologías, que toman forma de visores que cubren por completo el campo visual del usuario pretenden llevar la inmersión de los videojuegos un paso más allá, haciendo que el usuario se sienta prácticamente "dentro" del juego. Aunque estos dispositivos no están del todo presentes de forma comercial, lo estarán en un futuro cercano y prometen introducir un cambio radical en la experiencia del usuario.

Oculus Rift

La industria de los videojuegos es a día de hoy un gigante que no para de crecer, que involucra grandes cantidades de dinero y recursos en el desarrollo de productos y cuyos beneficios van a parar también al sector tecnológico. Además, aunque las mayores cantidades de dinero y recursos sean gestionadas por las grandes compañías, los juegos independientes o "indies" de pequeñas y medianas empresas tienen una presencia muy importante en el sector, ya que existen casos de grandes éxitos por parte de las mismas que han sido capaces de obtener millones de ventas. Algunos de estos son por ejemplo *Minecraft*, *Don't Starve*, *Limbo* y *Bastion*.

2.3 Juegos educativos

Los juegos educativos son aquellos dirigidos a aportar unos conocimientos o habilidades al jugador durante el transcurso del mismo. Se combina así formación con entretenimiento. Existen los claramente diseñados para una función didáctica, orientados normalmente a niños, pero se consideran también a veces como educativos ciertos juegos de construcción de imperios o ciudades en los que el jugador, sin ser ese su objetivo principal, adquiere conocimientos del mundo real relacionados con esas áreas.

Introducir el videojuego como un material didáctico en el aula ha sido tema de discusión por muchos años en donde podemos encontrar sus partidarios y detractores, estos últimos impulsados por etiquetamientos

generalizados e injustificados hacia los videojuegos con temáticas de violencia, adicción, aislamientos y sexismos; pero realmente no hay estudios científicos que demuestren que el uso de este tipo de videojuegos pueda desencadenar conductas agresivas o patológicas en los jugadores. Según Fergus (2010), es todo lo contrario, el videojuego actuará como un medio para descargar tensiones produciendo un efecto tranquilizador que disminuirá las posibilidades del jugador para cometer un acto violento,

"(...) ¿Es el videojuego el que desencadena conductas violentas o son jugadores violentos los que acceden a este tipo de contenidos?"

Fergus, C.J. (2010). «¿Blazing Angels or Resident Evil?». *Review of General Psychology*

El uso de videojuegos en las aulas es coherente con una teoría de la educación basada en competencias que enfatiza el desarrollo constructivo de habilidades, conocimientos y actitudes. Considerando las múltiples dimensiones que forman parte del proceso de significación, que se establece tanto por el hecho de jugar como de los juegos como producto y material docente en el aula, podemos decir que los videojuegos permiten el desarrollo de habilidades sociales (*Dondi*, *Edvinsson* y *Moretti*, 2004), mejoran el rendimiento escolar, desarrollan habilidades cognitivas y motivan el aprendizaje (*Rosas*, et al, 2003). Además, mejoran la concentración, el pensamiento y la planificación estratégica (*Kirriemuir* y *Mcfarlane*, 2004) en la recuperación de información y conocimientos multidisciplinarios (*Mitchel* y *Savill-Smith*, 2004), en el pensamiento lógico y crítico y en las habilidades para resolver problemas (*Higgins*, 2001). Los alumnos deben de responder a estímulos variables y constantes, sobre todo en un mundo mediatizado como el actual, que ofrece amplia información y tecnología.

Los videojuegos por tanto pueden considerarse como un medio para lograr grandes ventajas, como posibilitar nuevos medios de interacción con el entorno, facilitar la introducción de tecnologías de la información y la comunicación (Hayes, 2007). En la siguiente tabla se resumen algunas de las áreas de aprendizaje en que los videojuegos pueden contribuir a su desarrollo:

Desarrollo personal y social	• Proporciona interés y motivación. • Mantiene la atención y la concentración. • Puede trabajarse corno parte de un grupo y se pueden compartir recursos.
Conocimiento y comprensión del mundo	• Conocer algunas cosas que pasan. • Uso temprano del control del software.
Lenguaje y alfabetización	• Anima a los niños a explicar lo que está pasando en el juego. • Uso del discurso, de la palabra para organizar, secuenciar y clarificar el pensamiento, ideas, sentimientos y eventos.
Desarrollo creativo	• Respuesta en formas muy variadas. • Uso de la imaginación a partir del diseño gráfico, la música, y la narrativa de las historias.
Desarrollo físico	• Control de la motricidad a partir del uso del ratón en la navegación y selección de objetos.

Tabla 1. Áreas de aprendizaje y la contribución de los videojuegos en ellas. Fuente: videojuegos: Conceptos, historia y su potencial como herramientas para la educación

Los juegos son entornos que implican libertad de actuación, la necesidad de fijar metas y propósitos y encaminarse a conseguirlos, contribuyendo a que el usuario se responsabilice del desarrollo personal. En el juego el individuo vive una historia propia en cuyo desarrollo y resolución participa activamente, convirtiéndose en un entorno donde puede poner en práctica la pluralidad de mecanismos y recursos, que le permitirán interactuar libre y espontáneamente dentro de un sistema social. En este sentido son remarcables los estudios que analizan los videojuegos como un laboratorio de identidades. Podemos tener tantas identidades como videojuegos en los que jugamos, el juego ofrece por tanto la posibilidad de experimentar con nuevas identidades. Resaltamos además, cuatro razones para utilizar videojuegos en estrategias constructivistas, donde la didáctica se centra en la acción mental mediada por instrumentos (Contreras, Eguia, Solano, 2011):

• Adquirir conocimientos y mejorar habilidades son aspectos básicos del desarrollo de la partida en el videojuego. En todo videojuego para poder avanzar es imprescindible el aprendizaje. Los juegos se apoyan en el aprendizaje constante y pueden disponer de alternativas con el fin de adaptarse a las capacidades de aprendizaje de los distintos jugadores.

• Un videojuego consigue colocar al usuario en el centro de la experiencia, alcanzando el nivel de estado óptimo caracterizado por la inmersión, concentración y aislamiento y toda su energía e interés está focalizada en el juego. En este punto el jugador se implica en la experiencia de aprender.

• El videojuego como vivencia narrativa, permite la construcción de la realidad a través de la narración, recurso cognitivo básico por el cual los seres humanos conocen el mundo.

• El juego ofrece la posibilidad de experimentar con nuevas identidades ya que podemos tener tantas identidades como videojuegos y el individuo vive una historia propia en cuyo desarrollo y resolución participa activamente, lo que le permite experimentar con el contenido y el contexto. [1]

Hoy en día contamos con una gran variedad de videojuegos educativos e incluso con plataformas dedicadas en exclusivo a las aplicaciones educativas para los más pequeños, algunos ejemplos son www.educapeques.com o www.juegosarcoiris.com.

2.3.1 Juegos educativos para niños discapacitados

La mayoría de juegos diseñados para niños con discapacidades tienden a ser versiones simples de los juegos para niños de desarrollo tipico. Dependiendo del tipo de discapacidad y del área de dificultad del niño, los juegos pueden diseñarse para ayudar con las materias que más le cuesta aprender. Los juegos interactivos son una herramienta de aprendizaje más útil que las hojas de trabajo para los alumnos con discapacidades.

Para los niños y jóvenes adultos que tienen discapacidades de un rango medio a severo, los videojuegos pueden ofrecer un gran número de beneficios. Investigadores de la universidad de Wheeling descubrieron recientemente que jugar a videojuegos deportivos o de lucha ayuda a distraer a niños y jóvenes adultos que sufren de dolor crónico. Y lo que es más, está demostrado que los videojuegos ayudan a los niños a enfrentarse a operaciones de cirugía de forma más efectiva y con menos efectos secundarios que los tranquilizantes.

Los videojuegos están siendo utilizados incluso en el tratamientos contra el cáncer; El hacer ejercicio, algo vital para recuperarse tras la quimioterapia, se ha visto alentado por el uso de videojuegos como "Just Dance" cuando los niños se niegan a participar en otras formas de actividad física.

Además, permitir a personas con discapacidades, especialmente niños, participar en actividades que la mayoría de individuos disfrutan y dan por sentadas puede ayudar a reducir el dolor emocional y el sentimiento de ser diferente.

Algunos ejemplos de videojuegos para niños con discapacidades son los siguientes:

- **El proyecto Azahar:** Se trata de un conjunto de 10 aplicaciones, de descarga gratuita, de comunicación, ocio y planificación que, ejecutadas a través del ordenador y/o del teléfono móvil, ayudan a mejorar la calidad de vida y la autonomía de las personas con autismo y/o con discapacidad intelectual.

  Las aplicaciones contienen pictogramas, imágenes y sonidos que se pueden adaptar a cada usuario, pudiendo utilizarse, además, nuevos pictogramas, fotos de las propias personas y de sus familiares, así como sus voces, etc., de cara a la máxima personalización de cada aplicación.

- **El proyecto Aprender:** Este proyecto va dirigido a alumnos con dificultades de aprendizaje cualquiera que sea su causa u origen. No hace referencia a elementos básicos del currículo para una etapa concreta o área específica sino que pretende dar respuesta según las necesidades que presentan los alumnos en función del nivel de competencia curricular que posean y del grado de autonomía que puedan presentar.

2.4 Software para la creación de videojuegos

El proceso de creación de un videojuego es largo y en el convergen varias disciplinas, entre las que se encuentran perfiles tanto creativos como técnicos. Es por ello que un amplio abanico de herramientas de software puede ser utilizado para crear un único videojuego. Nos encontramos herramientas para diseñar el propio juego, para programar, para la creación de imágenes... y un largo etcétera.

Nos centraremos en los distintos motores, entornos de programación, herramientas de creación de objetos 3d y herramientas de creación y edición de imágenes, ya que son las que nosotras hemos necesitado en nuestro proyecto.

Motores:

El motor de un videojuego consiste en un conjunto de librerías y rutinas de programación que funcionan como la base del mismo.
El objetivo básico del motor es el de proveer un motor de renderizado para los gráficos 2D y 3D del videojuego, así como de un motor físico o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes, streaming, administración de memoria y un escenario gráfico.

Existen motores de juegos que operan en consolas de videojuegos o en sistemas operativos, como pueden ser Windows o Mac OS, debido a las drásticas diferencias que existen entre los diversos sistemas.
Del mismo modo, el proceso de desarrollo de un videojuego puede variar notablemente en función del tipo de juego del que se trate, así como puede variar por reutilizar o adaptar un mismo motor de videojuego para crear diferentes tipos de juegos.

Es por esto que existen motores de videojuegos especializados en los distintos tipos de juegos que existen.

Hoy en día existen una gran variedad de motores completos y motores gráficos como OGRE 3D que es un motor gráfico gratuito con "open-source" para que el usuario pueda crear aplicaciones desde el lenguaje C++. Desarrolladoras grandes de videojuegos como Epic, Valve y Crytek han lanzado al público sus motores o SDKs para que los usuarios interesados en el desarrollo de videojuegos puedan descubrir cómo se elaboran y así tener una introducción amplia a la industria y el desarrollo. Otros ejemplos de motor de juego son el motor gráfico Doom Engine, Quake Engine, y GoldSrc, desarrollado por Valve y el cual fue utilizado para crear el exitoso juego Half-Life 1; otros motores famosos son Source, también creado por VALVe, y BLAM! Engine, desarrollado por Bungie, y en el cual se creó la famosa saga de Halo.

Entornos de programación:

Un entorno de programación es una herramienta que facilita a los desarrolladores o programadores el desarrollo de software, por medio de un editor de código fuente, herramientas de construcción automáticas, un depurador y un compilador. La mayoría de ellos proporcionan un auto-completado inteligente de código, un navegador de clases, un buscador de objetos y un diagrama de jerarquía de clases para su uso con el desarrollo de software orientado a objetos. Todas estas características agilizan y simplifican el proceso de desarrollo de software.

Algunos entornos de programación soportan múltiples lenguajes, tales como GNU Emacs basados en C y Emacs Lisp, y Eclipse, IntelliJ IDEA, MyEclipse o NetBeans, todos basados en Java, o MonoDevelop, basados en C#.

Normalmente, el soporte para lenguajes alternativos es proveído regularmente por un plug-in, permitiéndoles ser instalados en el mismo entorno al mismo tiempo. Eclipse, y Netbeans tienen plugins para C/C++, Ada, (por ejemplo AdaGIDE), Perl, Python,Ruby, y PHP, los cuales son seleccionados entre extensión de archivos, ambientes o ajustes de proyectos.

Herramientas de creación de objetos 3d:

Un Software de gráficos 3D es un conjunto de aplicaciones que permiten la creación y manipulación de gráficos 3D mediante el ordenador.
Al proceso de crear un objeto 3d se le llama modelado. El modelado 3D es el proceso de desarrollar una representación matemática de cualquier objeto tridimensional (ya sea inanimado o vivo) a través de un software especializado. Al producto se le llama modelo 3D.

Hay una amplia gama de Herramientas de creación de objetos 3D, algunos de ellos están especializados para una parte del sector u otra, por ejemplo, algunos están especializados en el 3d para videojuegos, otros en el 3d para películas de animación y otros en infoarquitectura.

Algunos ejemplos de herramientas de creación de objetos 3d son por ejempo 3d studio MAX o Maya, utilizados tanto por la industria del videojuego como por la del cine, asi como Lightwave 3D, programa que consiste en dos componentes: el modelador y el editor de escena. Es utilizado en multitud de productoras de efectos visuales. Tambien los hay de software libre, como Blender, que abarca desde el modelado y animación hasta la composición y renderización de complejas escenas en 3D.

Herramientas de creación y edición de imágenes:

Estas herramientas permiten al usuario crear y editar imágenes de forma interactiva y almacenarlas en el ordenador en un formato de archivo gráfico, como JPEG, PNG, GIF y TIFF.

Algunos editores están diseñados específicamente para la edición de imágenes fotorrealistas, como el popular Adobe Photoshop, mientras que otros están más orientados a las ilustraciones artísticas, como Adobe Fireworks.
Existen editores graficos vectoriales y editores graficos rasterizados,
Con frecuencia los editores de gráficos vectoriales y los editores de gráficos rasterizados contrastan, y sus características se complementan. Los editores de gráficos vectoriales son mejores para diseño gráfico ,diseño de planos, tipografía, logotipos, ilustraciones artísticas, ilustraciones técnicas, diagramación y diagramas de flujo. Los editores de gráficos rasterizados son más adecuados para manipulación fotográfica, ilustraciones fotorrealistas, collage, e ilustraciones dibujadas a mano usando una tableta digitalizadora.

3 Herramientas utilizadas

3.1 Microsoft XNA Game Studio

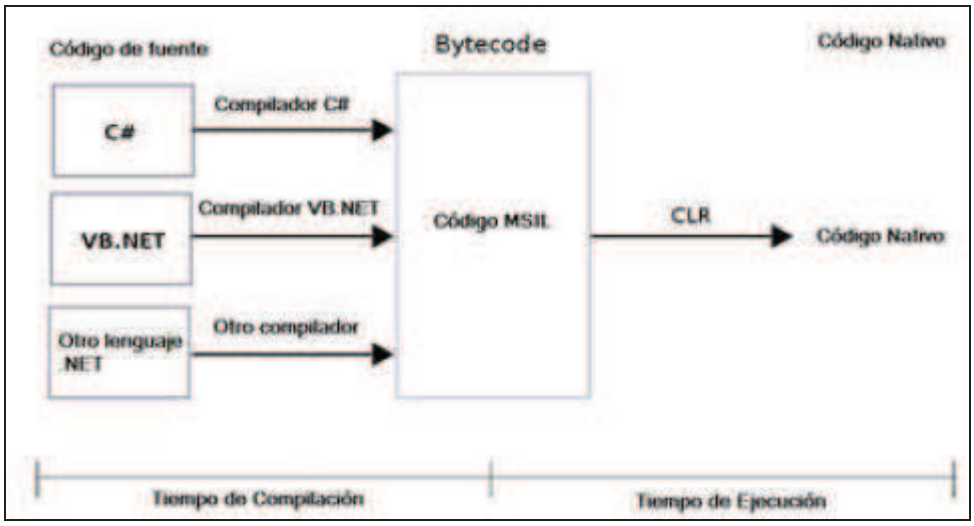
3.1.1 ¿Qué es XNA?

XNA es un Framework desarrollado por Microsoft para el desarrollo de videojuegos para las plataformas Xbox 360, Zune y Windows. Se desarrolló no sólo para facilitar este desarrollo de videojuegos, también para liberar a los desarrolladores de juegos de la creación de "código repetitivo" y traer diferentes aspectos de la producción de un juego a un único sistema conjunto.

Técnicamente es un Marco de Trabajo (Framework) basado en .NET Framework 2.0. Al igual que éste, XNA corre sobre CLR aunque con una implementación que provee un manejo optimizado para la ejecución de videojuegos. Para ello provee de un amplio conjunto de bibliotecas de clases, específicos para el desarrollo de juegos, que promueve la reutilización de código máximo a través de plataformas de destino.

El CLR, Common Language Runtime (Lenguaje común en tiempo de ejecución) constituye uno de los pilares de la tecnología .NET de Microsoft. Con la entrada de Java en el mercado de las tecnologías, surgió el concepto de Máquina Virtual ya que de esta manera el lenguaje de codificación era compilado a un lenguaje intermedio el cual podía ser ejecutado en toda la máquina con una máquina virtual. Microsoft adopta esta idea en .NET creando CLR.

La diferencia fundamental respecto a Java y su máquina virtual es que .NET no se limita a un único lenguaje. De esta forma los desarrolladores que usan CLR escriben el código en un lenguaje como C# o VB.Net y en tiempo de compilación, un compilador .NET convierte el código a MSIL (Microsoft Intermediate Language). Después, en tiempo de ejecución, el CLR convierte el código MSIL en código narrativo para el sistema operativo.



Paso de un código fuente a un código nativo

Fijándonos en las diferentes capas de XNA, vistas de arriba abajo, XNA Game Studio utiliza la funcionalidad del XNA Framework, y éste a su vez se basa en el .Net Framework.

Desde su salida, XNA Game Studio ha ido implementando diferentes versiones. La última y más reciente, la 4.0, es lo que le dará la competencia directa con el *iPod* de Apple ya que permite desarrollar juegos y aplicaciones para *Windows Phone 7* aparte de tener un mercado en línea donde los desarrolladores suben sus aplicaciones y los usuarios del teléfono pueden comprar o probar.

3.1.2 Ventajas de usar XNA

Utilizar esta plataforma concreta nos ofrece ventajas frente a otros competidores:

- Documentación: existe amplia documentación disponible para XNA, así como numerosas comunidades de usuarios que desarrollan para ella. Además se dispone de blogs y de foros activos donde consultar dudas o cuestiones surgidas durante el desarrollo.

- Facilidad de uso: la plataforma XNA fue desarrollada por Microsoft expresamente para que usuarios amateur desarrollaran juegos con ella. Por ello resulta una herramienta sencilla de usar en comparación con otras opciones. Además, está ampliamente testada contra fallos y es robusta. La continua revisión y actualización a la

que es sometida provoca su mejora en cada nueva versión incluyendo múltiples y novedosas opciones.

- Facilidad de conversión: con XNA no sólo es posible desarrollar juegos para PC, sino que es posible desarrollar juegos para la consola *Xbox 360*, para el dispositivo de audio *Zune* y, más recientemente, para móviles con sistema operativo *Windows Phone 7*. Además, si se desarrolla el juego para una u otra plataforma, la conversión posterior a otra de ellas es sencilla, ya que usan las mismas API's. De esta forma es posible reutilizar estos componentes si se desea realizar otro juego para otra plataforma.

- Gratuito: XNA es una herramienta gratuita que proporciona las mismas características que otras versiones de pago.

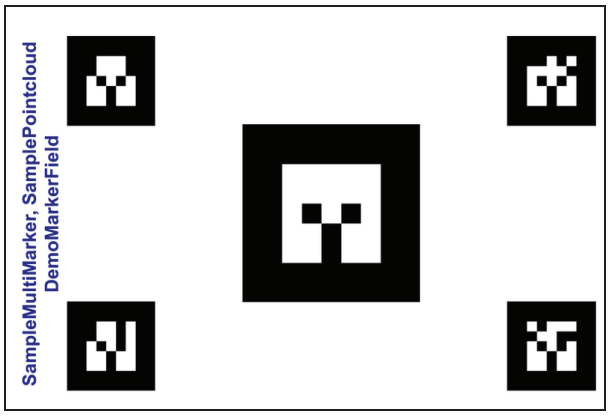
- Potente: XNA permite la creación no sólo de videojuegos independientes o amateurs, sino que es una herramienta potente con la que poder desarrollar videojuegos profesionales de gran calidad gráfica.

3.1.3 Goblin XNA

Goblin XNA es una plataforma para interfaces en 3D, incluyendo realidad aumentada para móviles y realidad virtual, con énfasis en el desarrollo de juegos. Está escrita en C# y basada en la plataforma de Microsoft XNA.

Goblin XNA hereda algunos de los objetivos de un proyecto anterior llamado *Goblin*, pero con un gran énfasis en relación a la funcionalidad de interfaces en 3D, subiendo el nivel de las ya existentes funcionalidades de motores de juegos y desarrollos de *Directx 3D*.

Actualmente, la plataforma soporta rastreo de posición y orientación 6DOF (Six Degree of Freedom) (Seis grados de libertad) mediante el uso de marcadores a través de ARTag o Alvar con *OpenCV* o *DirectShow*. Las físicas son suportadas a través de *BulletX* y *Newton Game Dynamics*. Goblin XNA también incluye un sistema 2D GUI para la creación de componentes de interacción clásicos en 2D.



Marcadores de Alvar 2.0.0

OpenCV

OpenCV (Open Source Computer Vision) es una lobrería software open-source de visión artificial. Tiene una licencia BSD, lo que le permite utilizar y modificar el código y además, tiene una comunidad de más de 47000 personas y más de 7 millones de descargas. Es una librería muy usada a nivel comercial, desde Google, Yahoo, Microsoft, Intel, Sony, Honda, etc.

La librería consta de más de 2500 algoritmos que permiten identificar objetos, caras, clasificar acciones humanas en vídeo, hacer tracking de movimientos de objetos, extraer modelos 3D, encontrar imágenes similares, eliminar ojos rojos, seguir movimiento de los ojos, reconocer escenario, etc. Se usa en aplicaciones como la detección de intrusos en vídeos, monitorización de equipamientos, ayuda a navegación de robots, inspección de etiquetas de productos, etc.

OpenCV está escrito en C++, tiene interfaces en C++, C, Python, Java y MATLAB interfaces. Además, funciona en Windows, Linux, Android y Mac OS.

3.1.4 Kinect SKD para Windows

El "software development kit" de Kinect para Windows (SDK) permite al usuario crear apps comerciales para Windows Store que soportan reconocimiento de gestos y voz usando C++, C#, Visual Basic o cualquier otro lenguaje .NET.

En la actualización 1.5 sacada en 2012 añadieron soporte en diversas lenguas, entre ellas el español, lo que aumentó el número de idiomas para el reconocimiento de habla.

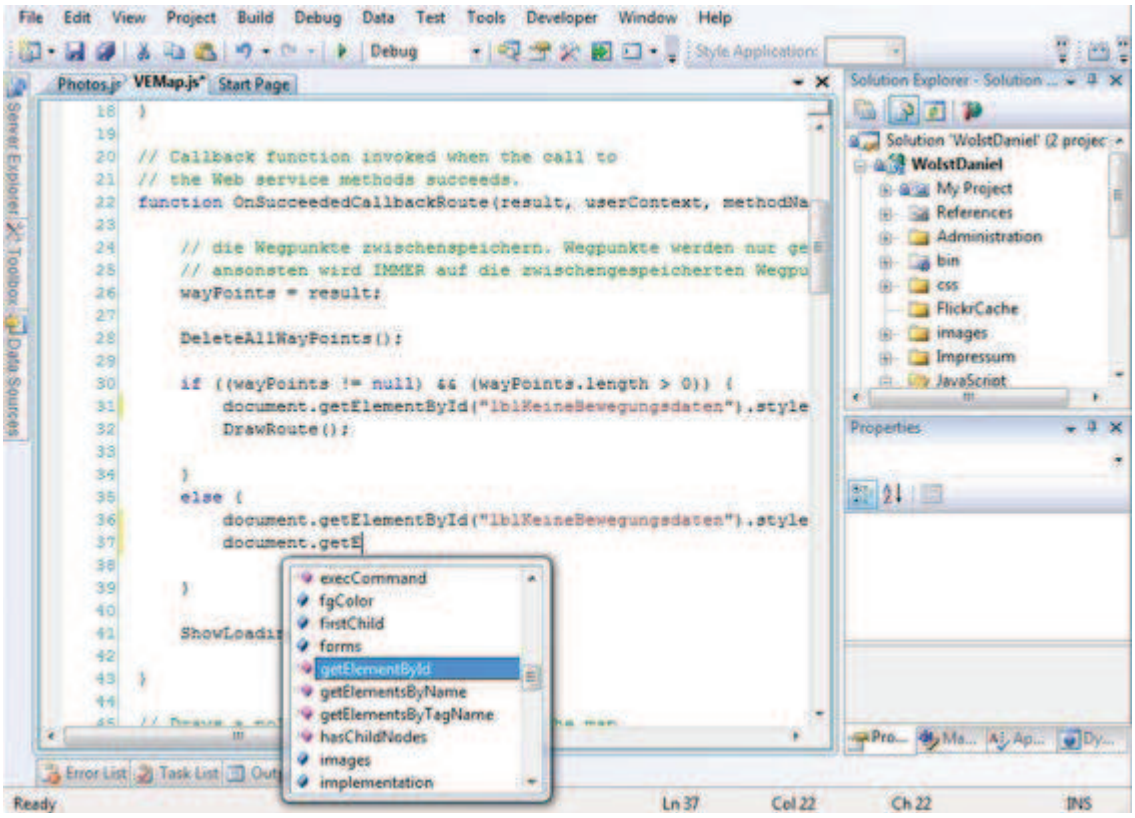
3.2 Microsoft Visual Studio Express

Microsoft Visual Studio Express es un conjunto de herramientas básico y gratuito que permite desarrollar y construir aplicaciones para la Web, Smartphone, escritorio o la nube.

Se trata de un software etiquetado para cualquier nivel de desarrollador, siendo una herramienta bastante buena para principiantes.

Funciona bien integrando la plataforma .NET junto a dos programas de programación soportados: Visual Basic y C#. Pero está limitado a estos lenguajes y por lo tanto es limitado.

Al ser una versión Express ofrece unas opciones más básicas que su versión de pago *Visual Studio*. Sin embargo, las características son las básicas para desarrollar un proyecto.



Vista de la interfaz de Visual Studio Express 2010

3.2.1 C#

C# o C Sharp se trata de un lenguaje orientado a objetos que permite a los desarrolladores compilar diversas aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Se utiliza C# para aplicaciones cliente de Windows, servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor, etc.

La sintaxis de C# es muy expresiva, pero también es sencilla y fácil de aprender. Cualquier persona familiarizada con C, C++ o Java, reconocerá esta sintaxis basada en signos de llave. Pero además, la sintaxis de C# simplifica muchas de las complejidades de C++ y proporciona características eficaces tales como tipos de valor que admiten valores NULL, delegados, enumeraciones, acceso directo a memoria y expresiones Lambda, que no se encuentran en Java.

C# admite tipos y métodos genéricos, proporcionando mayor rendimiento y seguridad de tipos, e iteradores, permitiendo a los implementadores de clases de colección definir comportamientos de iteración personalizados que el código cliente puede utilizar fácilmente.

Como se trata de un lenguaje orientado a objetos, C# admite conceptos de herencia, encapsulación y polimorfismo. Todos los métodos y variables, incluido el método *Main* que es el punto de entrada de la aplicación, se encapsulan dentro de definiciones de clase. Una clase puede heredar directamente sólo de una clase primaria, pero puede implementar cualquier número de interfaces. Para reemplazar a los métodos virtuales en una clase primaria se requiere la palabra clave *override* como medio para evitar redefiniciones accidentales.

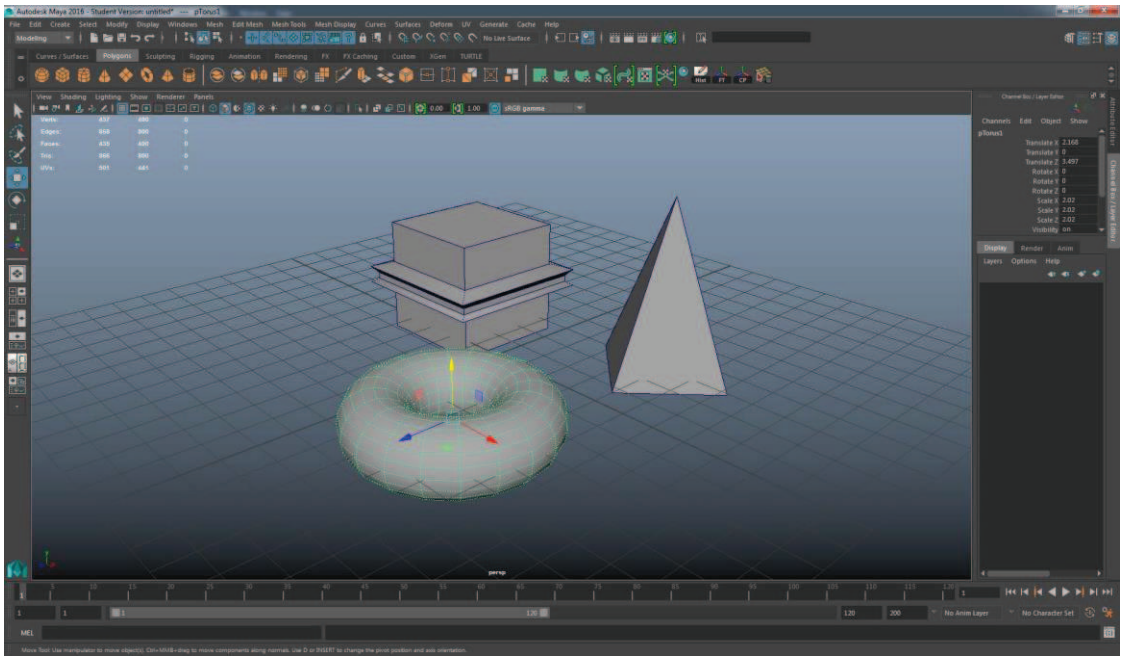
En C# una *struct* es como una clase sencilla: es un tipo asignado en la pila que puede implementar interfaces aunque no admite la herencia.

Si se necesita interactuar con otro software de Windows, como objetos COM o archivos DLL nativos de Win32, se puede hacer con C# mediante un proceso denominado *interoperabilidad.* La *interoperabilidad* habilita los programas de C# para que puedan realizar prácticamente las mismas tareas que una aplicación C++ nativa. C# admite incluso el uso de punteros y el concepto de código “no seguro” en las casos en el que el acceso directo a la memoria es totalmente crítico.

3.3 Maya

3.3.1 ¿Qué es Maya?

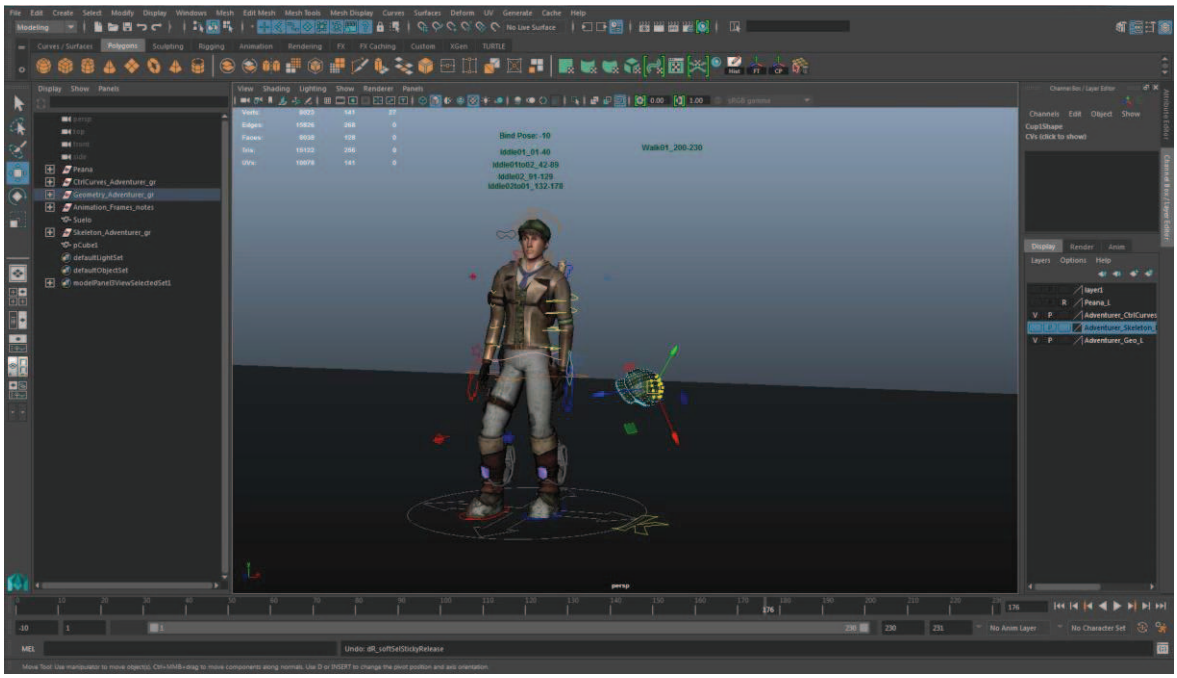
Maya es un programa informático, de la popular empresa en el sector, Autodesk, dedicado al desarrollo de gráficos 3D por ordenador, efectos especiales y animación. El programa está disponible para los siguientes sistemas operativos: Microsoft Windows, GNU/Linux, IRIX y Mac OS X.



Maya se caracteriza por su potencia y flexibilidad, nos proporciona un trabajo creativo completo con las herramientas necesarias para el modelado, la realización de animación, la simulación de ropa y cabello, creación de efectos visuales , renderizacion, dinámicas (simulación de fluidos), etc. Puede trabajar con cualquier tipo de superficie NURBS, Polygon y Subdivision Surfaces, incluyendo la posibilidad de convertir entre todos los tipos de geometría.

El uso de Maya está muy extendido debido a su gran capacidad de ampliación y personalización. El código que forma el núcleo de Maya está escrito en el lenguaje de programación MEL (Maya Embedded Language) y gracias a este se pueden crear scripts y personalizar el paquete, aunque actualmente también es posible crear estos scripts en el lenguaje de programación Phyton.

La característica más importante de Maya es lo abierto que es al software de terceros, el cual puede cambiar completamente la apariencia de Maya. El mismo software se puede transformar debido a sus opciones altamente personalizables.



3.3.2 Ventajas de usar Maya

A parte de su gran capacidad de ampliación y personalización, las ventajas que nos ofrece Autodesk Maya son las siguientes:

- Documentación: Al ser un programa muy utilizado en el mundo tanto de los videojuegos como del cine, existe una gran cantidad de información relativa al mismo en numerosas comunidades de internet, como en los foros del mismo Autodesk. Del mismo modo, en la página web de autodesk encontramos documentación sobre cada herramienta y cada técnica disponibles en Maya.

- Potencia: Se trata de un programa con mucha potencia, con el que tenemos muy pocas limitaciones. Nos permite crear tanto modelos como animaciones de gran calidad.

- Versión gratuita para estudiantes: Aunque Maya no es un programa gratuito, tiene una versión gratuita para que los estudiantes puedan aprender a usar el programa. El producto de tu trabajo en Maya con la versión para estudiantes no se puede comercializar, pero es una buena opción para proyectos sin ánimo de lucro

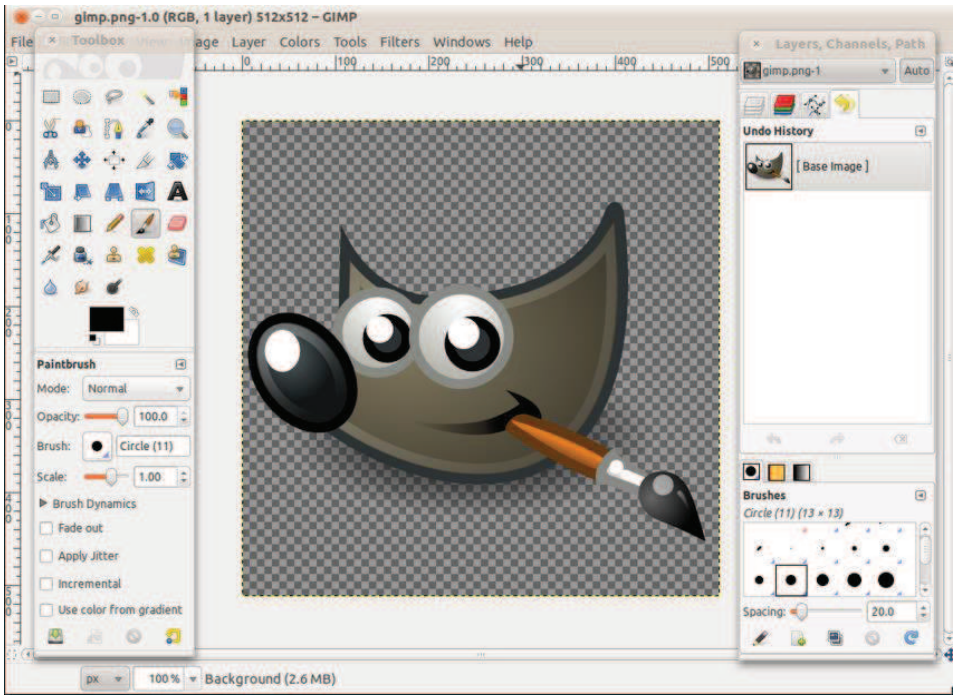
- Interoperabilidad: Maya nos permite la importación y exportación en una gran cantidad de formatos de archivos, lo que hace más sencillo poder intercambiar archivos con otros programas. Consta de la tecnología FBX, que nos permite exportar modelos 3D con textura y animación esqueletal, entre otras cosas.

3.4 GIMP

La siglas GIMP originalmente significan (General Image Manipulation Program) (Programa de manipulación de imágenes general). Este nombre cambio en 1997 a (GNU Image Manipulation Program) (Programa de manipulación de imágenes de GNU) para pasar a formar parte oficial del proyecto GNU.

GIMP es un programa que sirve para la edición y manipulación de imágenes. Actualmente se encuentra publicado bajo la licencia GPL (GNU General Public License). Además es un software multiplataforma ya que se puede utilizar en varios Sistemas Operativos. La primera versión fue desarrollada para sistemas Unix, inicialmente fue pensada específicamente para GNU/Linux, sin embargo actualmente existen versiones totalmente funcionales para Windows y para Mac OS X.

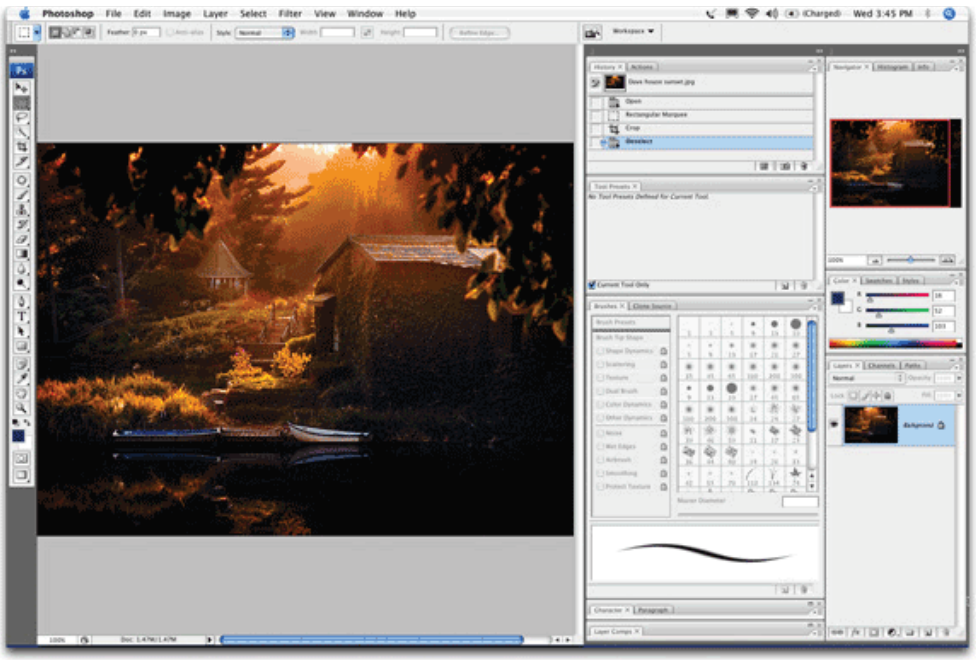
Este programa de licencia libre es una buena opción para los desarrolladores y diseñadores gráficos. Y muchos aseguran que es una buena competencia para el tradicional Adobe Photoshop.



Interfaz de GIMP

3.5 Photoshop CS3

Se trata de un editor de graficos rasterizados, desarrollado por Adobe Systems Incorporated. Adobe Photoshop fue creado en el año 1990 y hoy en dia es el programa líder mundial del mercado en aplicaciones de edición de imágenes. Soporta muchos tipos de archivos de imágenes, como BMP, JPG, PNG, GIF, entre otros, además tiene formatos de imagen propios.



Interfaz de Photoshop CS3

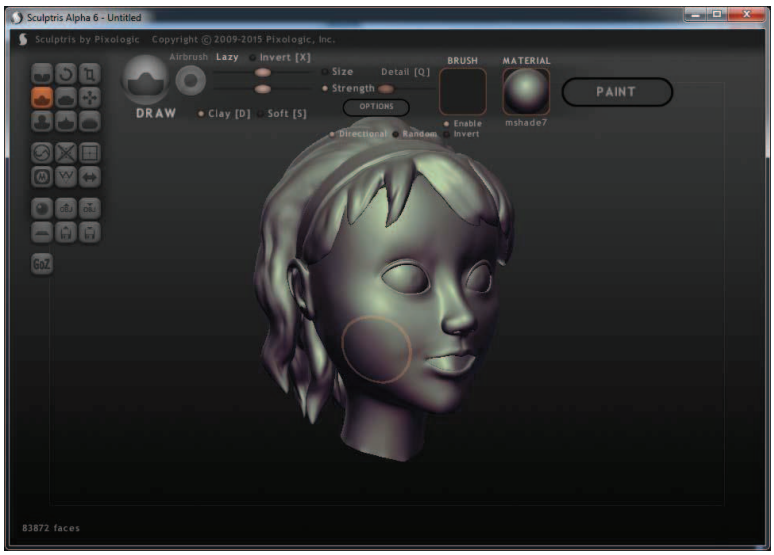
Se usa principalmente para el retoque de fotografías y gráficos, aunque sus aplicaciones son muchas y muy variadas. Se usa en multitud de disciplinas del campo del diseño y la fotografía.

La potencia de Photoshop radica en la gran variedad de herramientas que este ofrece, donde nos podemos encontrar desde las más simples hasta las más complejas, esto permite que nuestros resultados sean mejores y más rapidos.

Aunque Adobe Photoshop no tiene licencias gratuitas, disponemos en los ordenadores de la universidad de Photoshop CS3, lo que nos ha permitido utilizarlo sin problemas.

3.6 Sculptris

Sculptris es una herramienta software de escultura y pintura digital.
Nos permite esculpir objetos a partir de una malla incial, que podemos deformar hasta obtener el resultado deseado. Tambien nos permite definir las propiedades el material y pintar sobre el objeto tanto usando colores planos como usando texturas importadas o mapas de relieve.
Nos permite importar y exportar mallas 3d en el formato .obj, asi como generar mapas de normales y mapas de desplazamiento.



Aunque Sculptris no es un programa muy potente, tiene una interfaz simple que hace que sea una herramienta muy sencilla de usar en comparación con otras herramientas software de escultura digital. Además, es un programa totalmente gratuito.

4 Análisis del proyecto

4.1 Descripción general

Nadie puede discutir el gran potencial que tiene el juego como fuente de aprendizaje, independiente de la dificultad que conlleva a veces aplicarlo o incorporarlo a una dinámica escolar o académica. Es por este motivo que en los últimos años la industria de la informática está centrando muchos de sus esfuerzos en desarrollar juegos o plataformas educativas conscientes del buen uso que profesores y alumnos puedan hacer de estos.

Si nos referimos a videojuegos orientados a la educación, por lo general, suelen estar orientados a un público infantil. Si bien es cierto que existen títulos para gente más adulta como *Brain Training*, *Mind Quiz* y demás. Herramientas, en resumen, que sirven para potenciar la memoria e incrementar las destrezas comunicativas.

Lo primero y más importante es que el proyecto se divide en dos subproyectos:

- Safe Trip: Se trata de un juego para aprender a coger un avión en un aeropuerto. En él se ha trabajado en el desarrollo conjunto de un entorno 3D complejo con un sistema de inteligencia, limitado al uso de teclado/ratón como interacción principal.
  Para finalizarlo con éxito, el usuario ha de pasar ciertos puntos (checkpoints) para que el vuelo no se le escape. Así mismo contará con ciertas áreas con las que podrá interactuar y que cambiarán el transcurso de los eventos.

- ¡Sonríe!: Se trata de un juego para aprender a lavarse los dientes. En él se ha trabajado la adaptación de un juego ya existente en 2D a uno en 3D. Para ello, se hace uso de la realidad aumentada mediante la familiarización con marcadores.
  Para finalizarlo con éxito, el usuario debe quitar todas las bacterias de la boca utilizando un cepillo. Este cepillo es el elemento interactuable en la vida real y el juego captará su movimiento a través de la cámara.

Ambos proyectos se desarrollan en un entorno de tres dimensiones (3D). Además, ambos son problemas cotidianos para que un niño aprenda.

Con **Safe Trip**, se ha desarrollado toda una arquitectura de un juego complejo. Se ha puesto énfasis en un código genérico. Además todo el diseño 3D también ha tenido un gran estudio. Todo el diseño es propio y se ha ido desarrollando junto a la implementación por código. Cabe destacar que este proyecto se ha llevado a cabo entre dos personas, ocupándose una de la parte de diseño e implementación 3D y la otra del diseño e implementación del código. La capacidad de trabajo en conjunto también ha formado parte de este proyecto. Ambas partes se documentan por separado en su respectiva memoria.

En el caso de **¡Sonríe!** se centra en la interacción con un objeto real. La *realidad aumentada* es el término que se usa para definir una interacción a través de un dispositivo tecnológico, directa o indirecta, de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales consiguiendo una realidad mixta en tiempo real. Es decir, la *RA* convierte nuestro mundo físico en interactivo y digital. En este proyecto, y como se ha citado con anterioridad, el objeto del mundo real se trata de un cepillo de dientes, y mediante la realidad aumentada, ese cepillo de dientes interactua con las bacterias dentro del juego. Este proyecto se ha realizado en conjunto, no como el anterior que se separó en dos y fue la razón principal de escoger Goblin XNA.

Como se ve, ambos juegos tienen una carga educativa. Centrada más en el manejo de objetos de forma virtual en uno y más centrada en la superación de obstáculos en otro. Además, desde un principio, se idearon estas circunstancias de la vida en los juegos para que pudieran jugar todo tipo de niños, incluidos aquellos con ciertas discapacidades. Niños con problemas comunicativos encontrarán un medio para aprender a comunicarse mejor.

4.2 Descripción detallada

4.2.1 Safe Trip

"Safe Trip" es el título elegido para este juego educativo en el que el propósito es ofrecer una herramienta de aprendizaje.

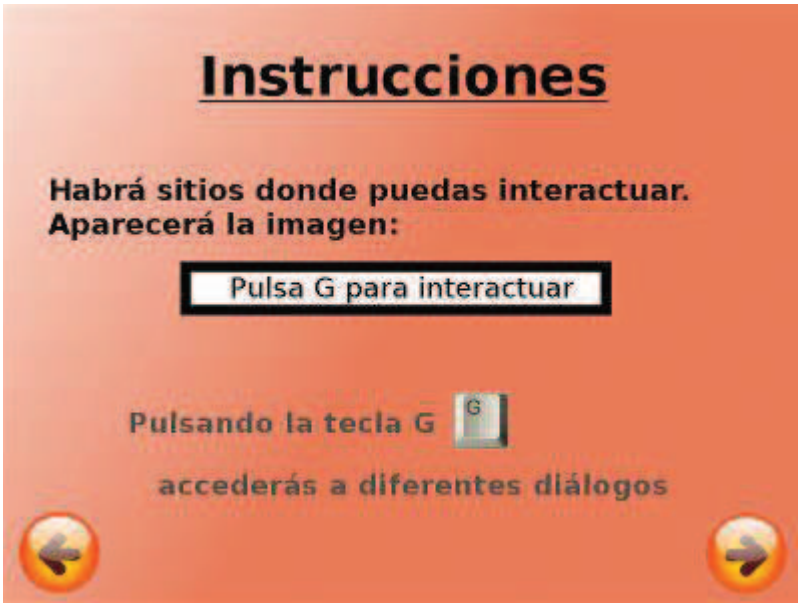
Como se ha explicado anteriormente, se trata de un juego en 3D (3 dimensiones) cuyo objetivo es pasar por ciertos *checkpoints* para llegar a tiempo a coger un avión.

Una vez ejecutado, se puede encontrar el Menú principal. En él hay dos posibles opciones a las que acceder:

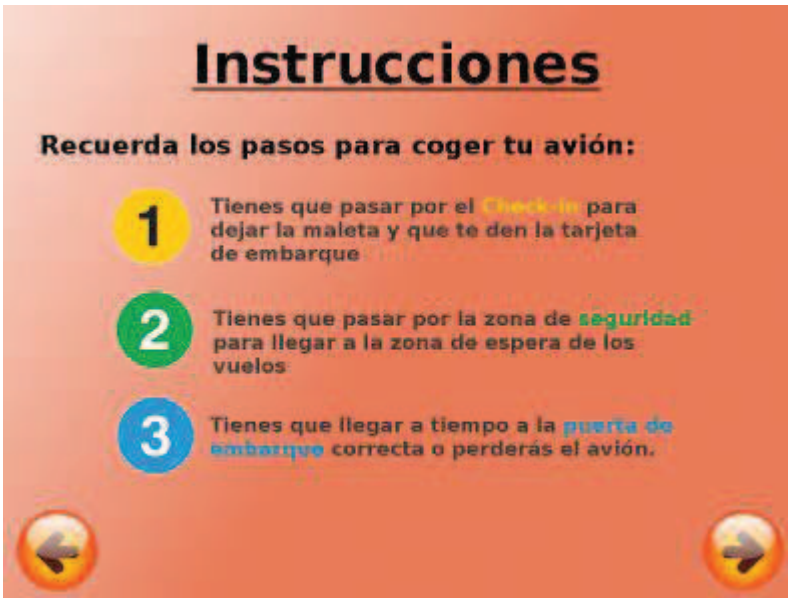
- Instrucciones: nos mostrará una serie de pantallas de explicación para poder interactuar con los elementos del juego.

Menú Principal del juego

Pantalla 1 de instrucciones



Pantalla 2 de instrucciones



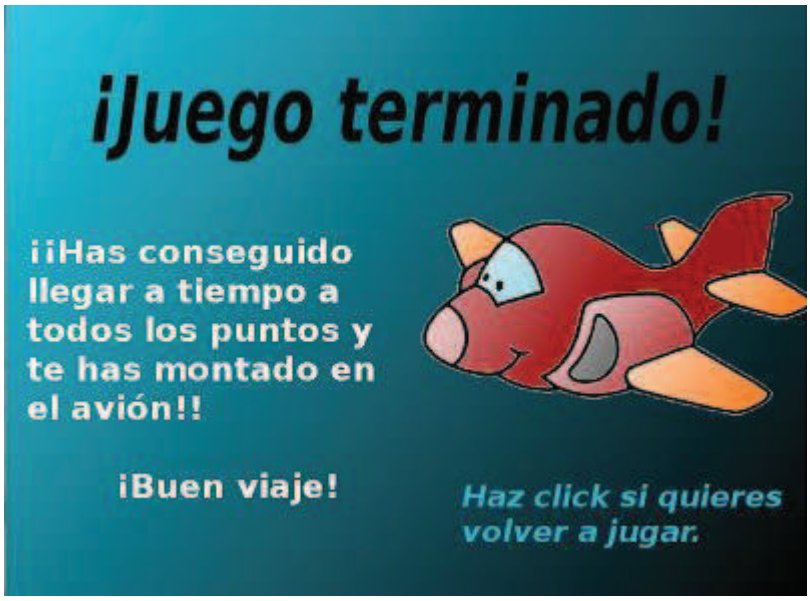
Pantalla 3 de instrucciones

- Comenzar a jugar: Empezará una nueva partida con una hora de salida de avión generada aleatoriamente. Habrá otros datos que también se generen aleatoriamente como el peso de la maleta, la cantidad de dinero, la puerta de embarque del avión, etc.

Cargando la partida

Cuando el juego finaliza puede ser porque se haya perdido el avión por alguna razón o bien porque se ha conseguido coger el avión a tiempo. De cualquier forma se podrá volver a jugar una nueva partida diferente a la anterior.

Pantalla tras perder el avión



Pantalla tras finalizar con éxito el juego

El juego tiene como protagonista a un niño que se ve en la pantalla en medio en todo momento. El personaje tiene un inventario al que puede acceder para ver los objetos.

Protagonista principal con el inventario abierto

El juego se desarrolla en el entorno de un aeropuerto. Se pueden distinguir áreas típicas de tal sitio donde se podrá interactuar o no, como las que aparecen en las siguientes imágenes.

Información del aeropuerto. Carteles informativos en el techo.

Zona de Check-in del aeropuerto

Máquinas expendedoras del aeropuerto

Cafetería del aeropuerto

Papalera del aeropuerto

Panel de vuelos y avión de fondo

Sala de espera y puertas de embarque

4.2.2 ¡Sonríe!

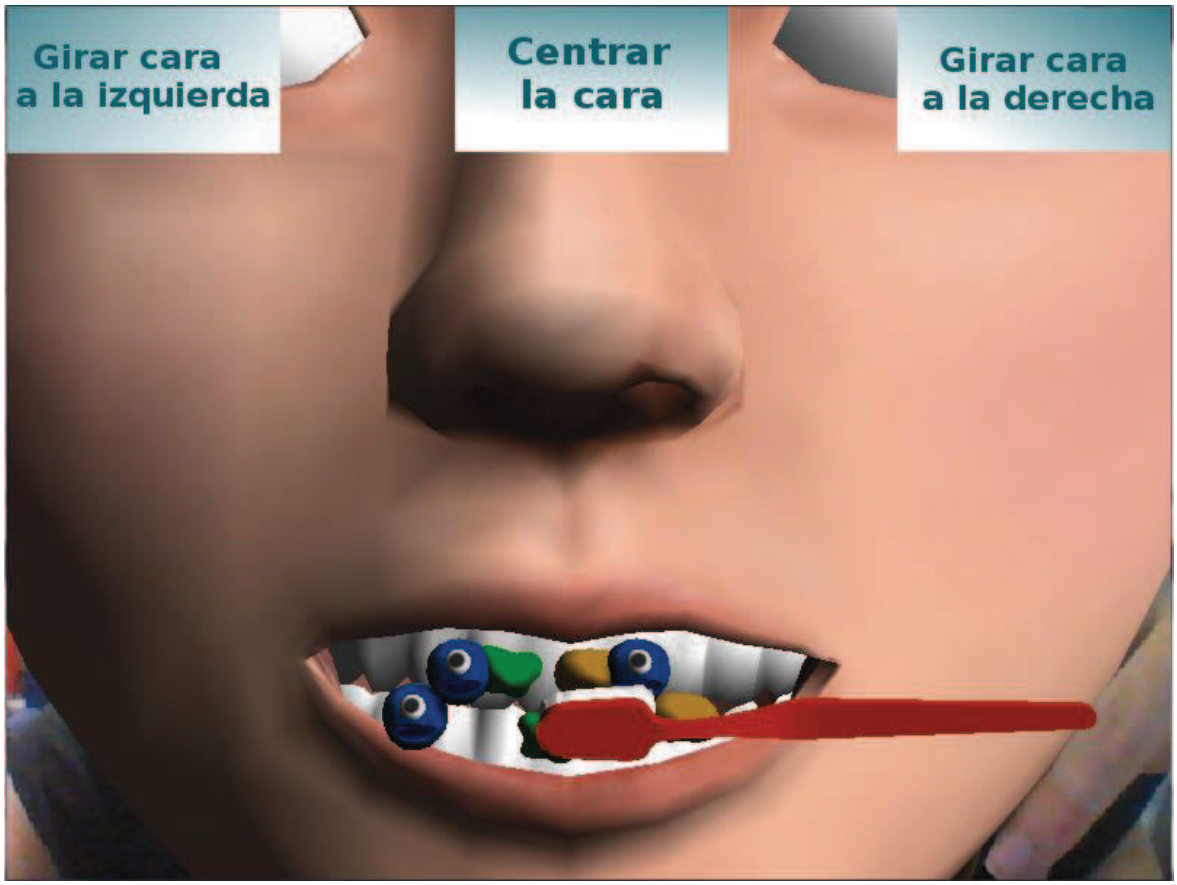
"Sonríe" es el título elegido para este segundo proyecto educativo pero enfocado a la realidad aumentada.

Como se ha explicado anteriormente, se trata de un juego en 3D (3 dimensiones) cuyo objetivo es aprender a cepillarse bien los dientes.

Cuando se ejecuta el juego aparece la pantalla principal.

Pantalla inicial del juego

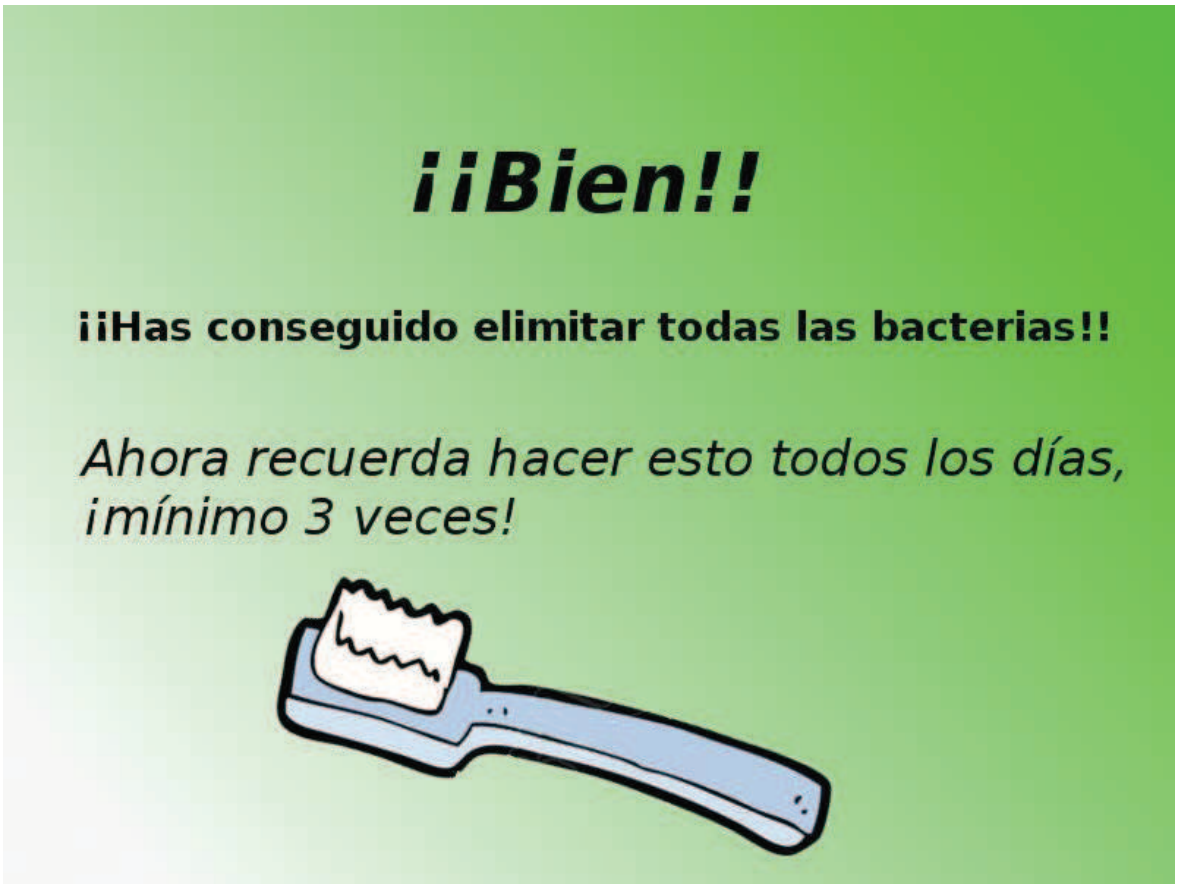
Para acceder al juego basta con pulsar sobre la pantalla. De esta forma se accede al contenido del juego. Se trata de la cata de un niño con la boca abierta donde se pueden identificar bacterias.



Captura del juego

Mediante el uso de un cepillo de dientes real enfocado a la cámara se podrán ir eliminando las diferentes bacterias de los dientes mediante un movimiento de arriba abajo. Este movimiento se captará gracias a un marcador adherido al cepillo de dientes.

Cuando se consiguen eliminar todas las bacterias, el juego finaliza mostrando una pantalla final.



Pantalla final del juego

5 Safe Trip

5.1 Diseño del proyecto

Diseño gráfico:

❖ **Diseño de interfaces:**

En este videojuego hay varios tipos de interfaces:
-Las pantallas de inicio, final y carga.
-Las pantallas que contienen las instrucciones del juego.
-la interfaz del inventario.

Para crear las distintas interfaces de las que se compone este juego, se ha de tener en cuenta la edad media del público que será usuario final del mismo. Al tratarse de niños de 7 a 14 años se han de hacerse interfaces con colores vivos e imágenes que les puedan llamar la atención. También deben utilizado fuentes de texto grandes para facilitar la lectura y se debe intentar poner poco texto en cada diapositiva, de esta forma se evita que los usuarios se aburran leyendo un gran texto y se mantiene un ambiente amigable y divertido.
La interfaz del inventario ha de ser limpia y clara.

❖ **Diseño 3d:**

- Estructura del escenario:

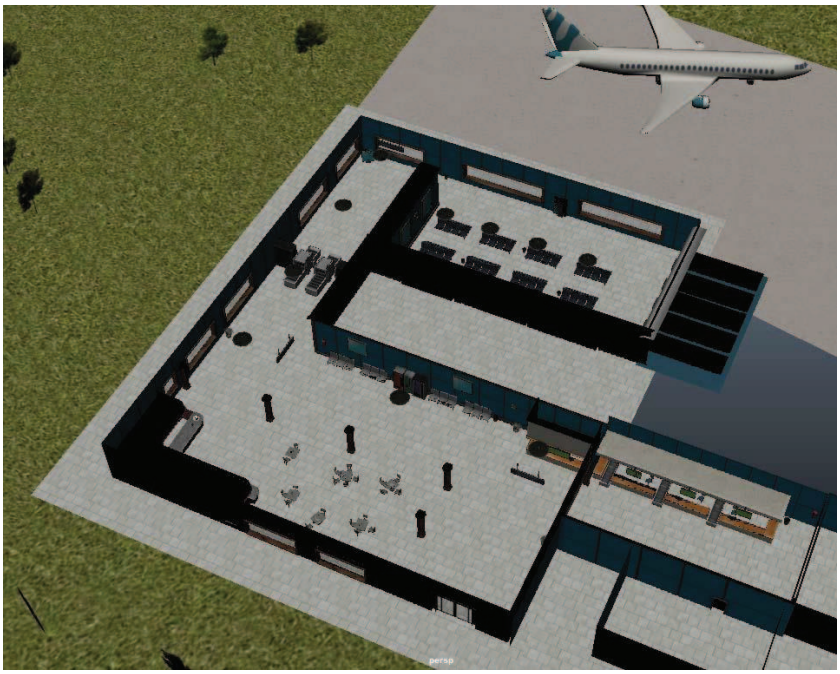
  - Interior: El interior del edificio cuenta con las siguientes zonas: Entrada, ventanilla de información, ventanillas de check-in, cafetería, zona en la que se realiza el control de seguridad, sala de espera y puertas de embarque.
    Varias cosas se han tenido en cuenta a la hora de diseñar la estructura del edificio:

-La zona de inicio ha sido diseñada de forma que al entrar al juego veas fácilmente la ventanilla de información, en la que se te informara de ir al check-in, que se encuentra al lado de la misma.

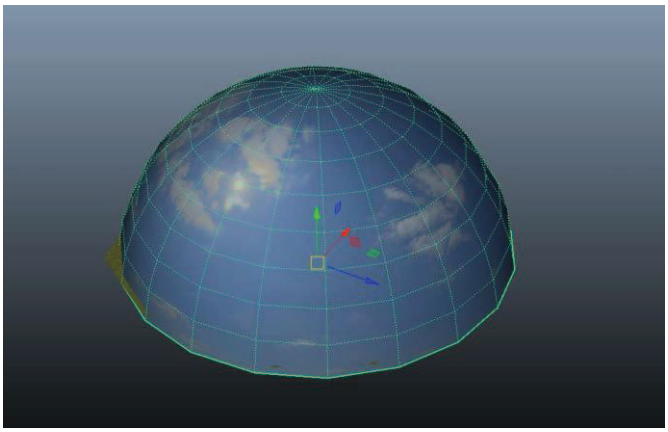
-Al necesitar una zona que diera vida y realismo al escenario se decidió implementar la zona de la cafetería. Actualmente el único objetivo de esta zona es el ya comentado, aunque está preparada para la futura implementación de interacciones con sus objetos.

-La zona de seguridad se ha diseñado dividiendo el escenario en dos. Tras la misma nos encontramos con un largo pasillo tras el cual vemos un avión a través de las ventanas, lo que da a entender al jugador que nos acercamos a la fase final del juego.

-El área total del edificio es pequeña, aunque ligeramente enrevesada, con ello se pretende que los usuarios aprendan a informarse y busquen, en los carteles que hay repartidos por el aeropuerto, la forma de llegar a su destino.



- Exterior: La zona exterior del escenario no es accesible al jugador, su única función es la de dotar de ambiente al escenario. Consta de una base de hierba, vegetación y un cielo despejado.
  Para la creación del cielo se decidió usar una cúpula con una textura de cielo.



- Personajes:

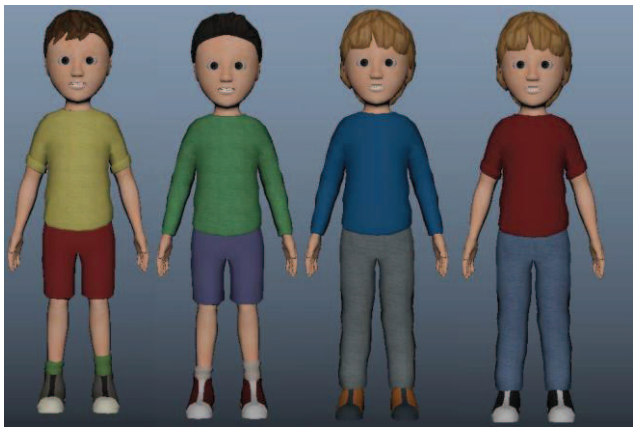
  Los personajes han sido diseñados en distintas partes, es decir, de cada tipo de personaje se han creado varios peinados, una cabeza, y varios cuerpos con distintas ropas y texturas. Esto nos permite tener una gran variedad de personajes distintos, ya que al generar cada personaje se elegirán aleatoriamente un peinado y un cuerpo. De esta forma, si contamos con 3 peinados diferentes y 3 cuerpos diferentes, tendremos 9 (3x3) personajes distintos.

Variaciones en el cuerpo y el peinado de los personajes adultos masculinos.

Variaciones en el cuerpo y el peinado de los personajes adultos femeninos.

Se han diseñado 3 cuerpos y 3 peinados diferentes para cada hombre adulto, 3 cuerpos y 3 peinados diferentes para cada mujer adulta, 4 cuerpos y 3 peinados diferentes para cada niño y 4 cuerpos y 3 peinados diferentes para cada niña.

Variaciones en el cuerpo y el peinado de los personajes niños femeninos.



Variaciones en el cuerpo y el peinado de los personajes niños masculinos.

- Objetos: El aeropuerto cuenta con una gran cantidad de objetos, algunos son necesarios para el desarrollo del juego mientras que otros, aunque prescindibles, se encargan de darle vida y variedad al escenario y lo convierten un lugar más acogedor. Por ello, en los objetos no imprescindibles del escenario, se ha buscado que fueran comunes en el entorno y que estuvieran dispersos de la forma más natural posible.

En cuanto a la cantidad de polígonos que han de utilizarse, los objetos se deben ser creados en low-poly, ya que hay una gran cantidad de objetos y de texturas. De esta forma evitaremos ralentizaciones en el juego.
Los objetos que cuentan con más detalle son los personajes, puesto que son los más complejos.

Todos los objetos estáticos del escenario, así como la estructura interior y exterior del mismo, se han exportado en un mismo archivo. De esta forma la implementación de los mismos en el videojuego es más sencilla. Con una excepción, aquellos objetos preparados para llevar texturas transparentes se han exportado en un archivo separado del resto de objetos, ya que el motor los trata de forma diferente.

- **Animaciones:** Las animaciones de los personajes son esenciales para un videojuego. Aportan un grado de realismo necesario para la credibilidad del mismo.
  Para dar al aeropuerto un ambiente bullicioso y activo, se decidió dotar a los personajes de una animación de caminar, de esta forma, estos se mueven constantemente por el escenario.

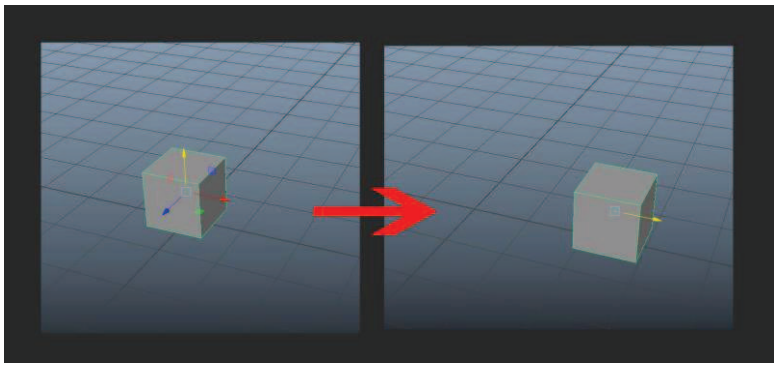
5.2 Implementación del proyecto

5.2.1 Modelado

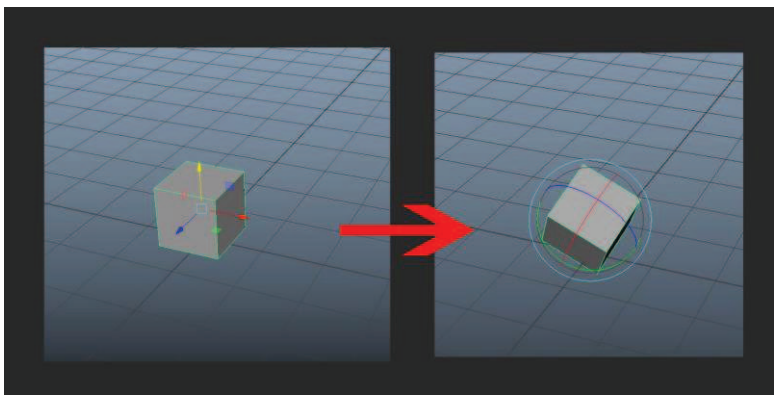
El software utilizado para el modelado de los objetos es Autodesk Maya.

Para el modelado de los objetos del escenario se han utilizado principalmente las siguientes herramientas básicas:

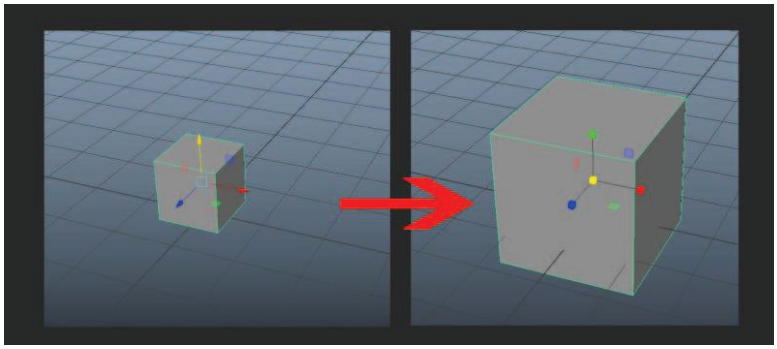
Mover: Esta herramienta nos permite desplazar un objeto, vértice, arista o cara por la escena.



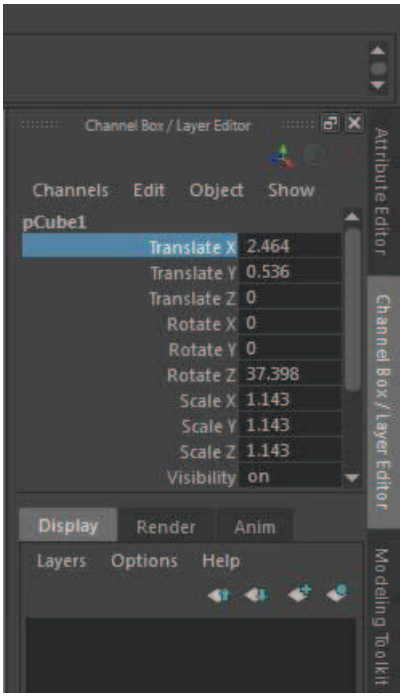
Rotar: Esta herramienta nos permite girar un objeto, vértice, arista o cara.



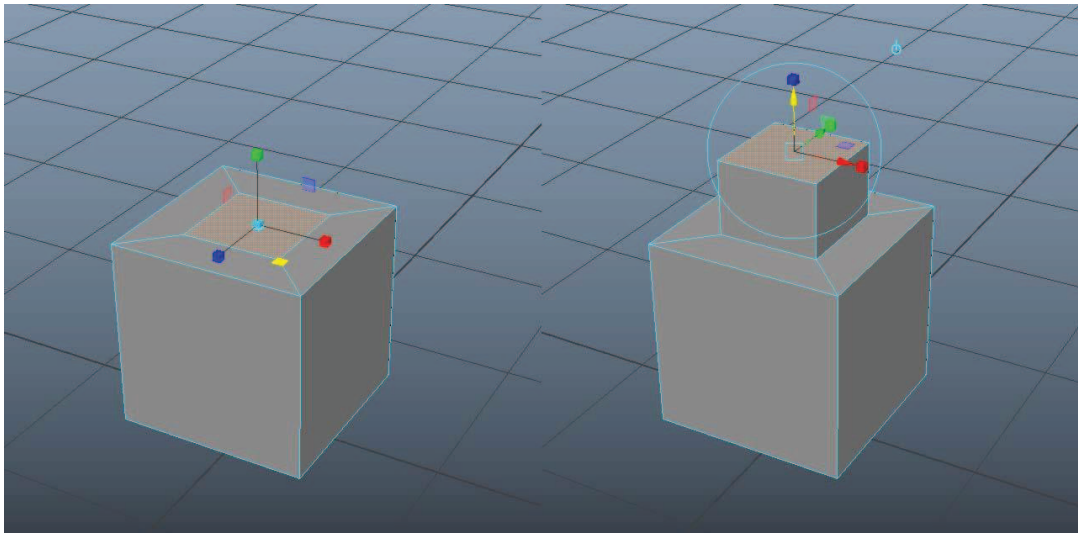
Escalar: Esta herramienta nos permite modificar el tamaño de un objeto, vértice, arista o cara.



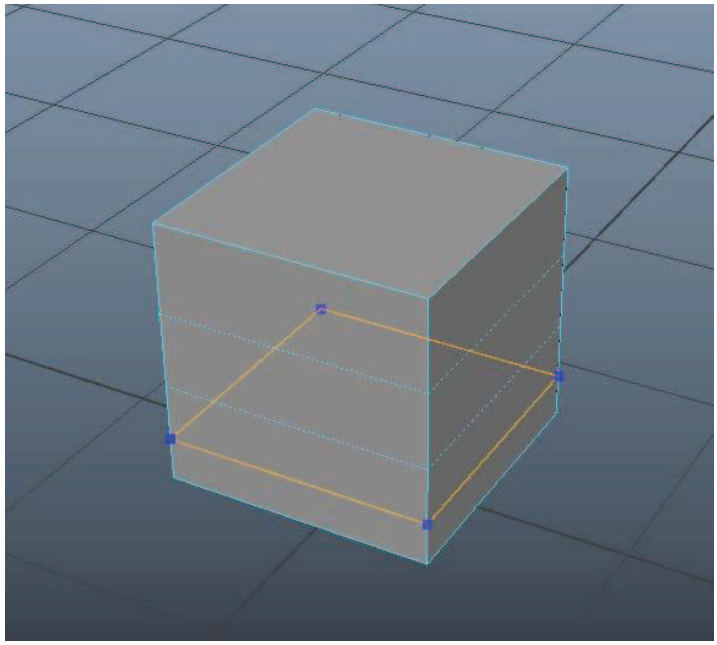
Visor de atributos: Este visor nos permite tanto ver los cambios anteriores en forma numérica, como modificarlos. De esta forma, si queremos mover un objeto a un punto exacto de la escena, o girarlo una cantidad de grados exacta, se hará desde este visor.



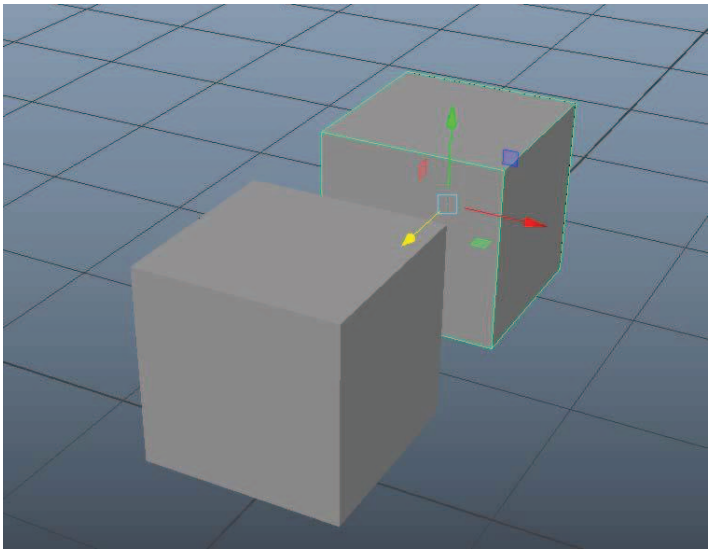
Extruir: Esta herramienta nos permite “alargar” un polígono y de esta forma crear más geometría en el objeto, con la forma deseada.



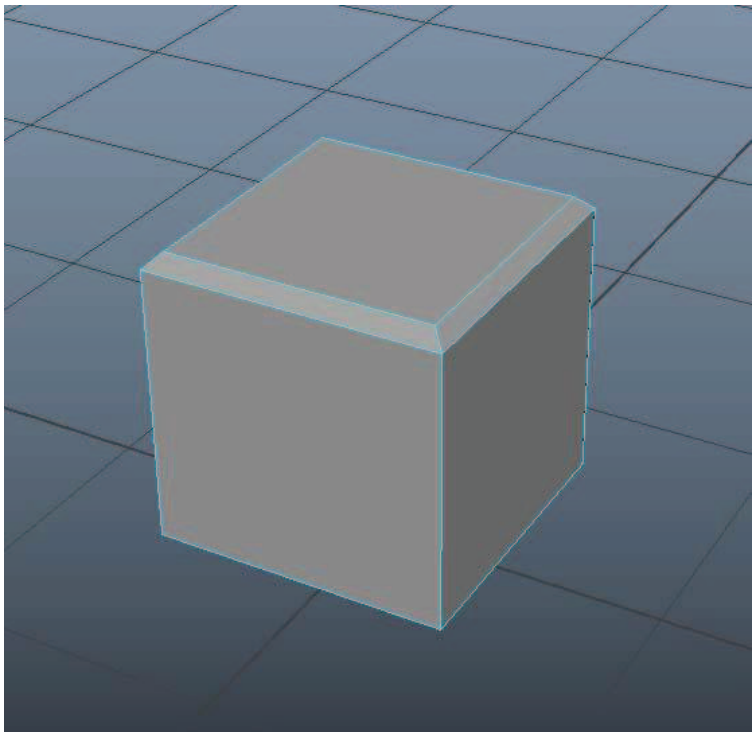
Cortar: Esta herramienta permite añadir cortes en el lugar que se desee del objeto, añadiendo así más geometría.



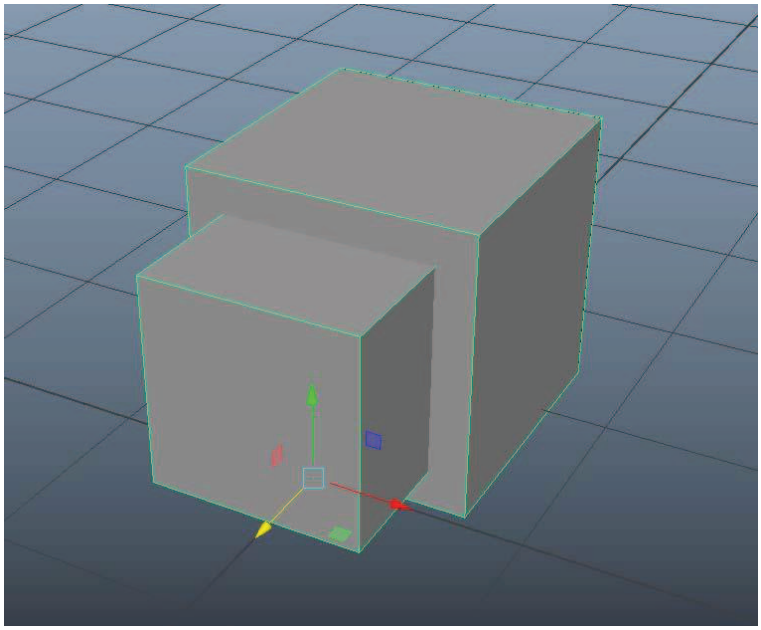
Duplicar: Esta herramienta permite obtener una copia exacta de un objeto.



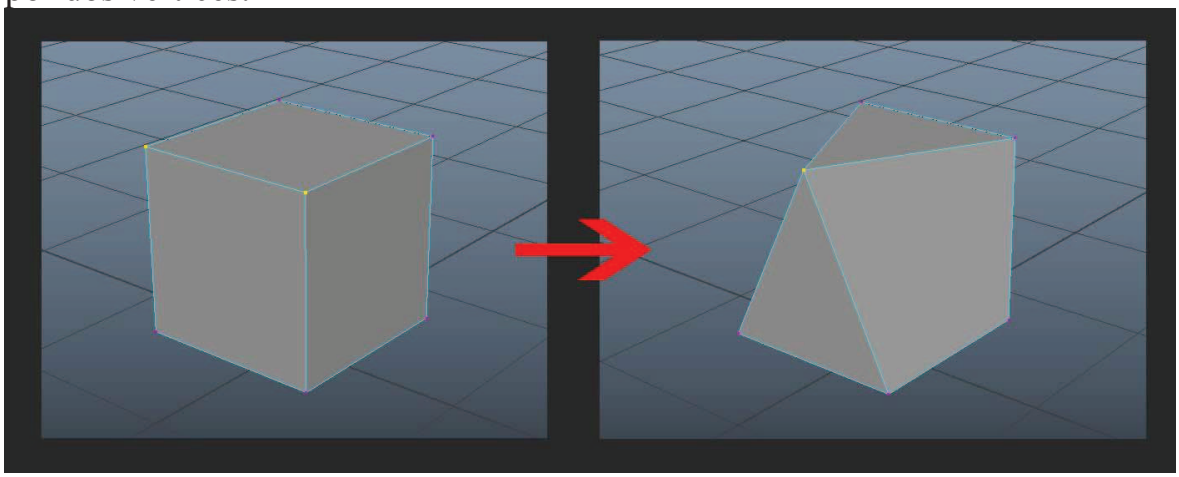
Biselar: Con esta herramienta es posible redondear una "esquina" de un objeto, añadiendo más artistas a los lados de la misma, de forma que quede más suavizada.



Combinar: Esta herramienta nos permite unir dos objetos en uno solo.

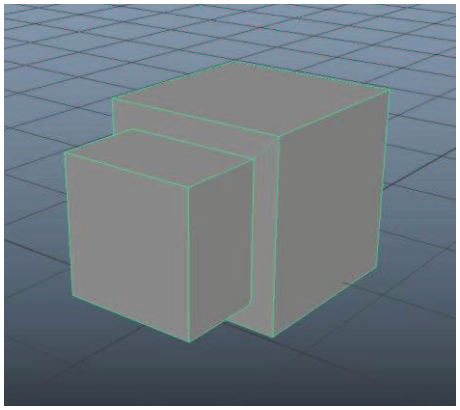


Unir: Esta herramienta nos permite colapsar dos vértices en uno, se puede utilizar seleccionando dos vértices o una arista, ya que una arista está formada por dos vértices.

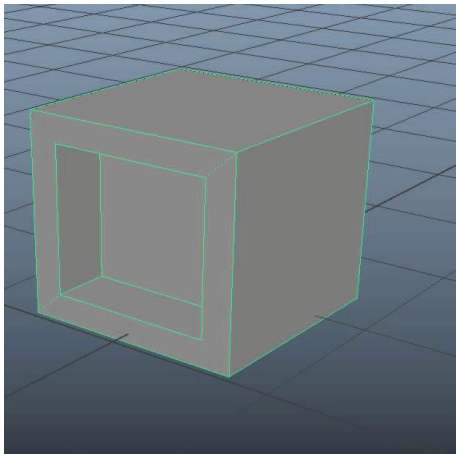


Herramientas Booleanas: Existen tres herramientas booleanas: unión, diferencia e intersección. Estas se utilizan siempre entre dos o más objetos

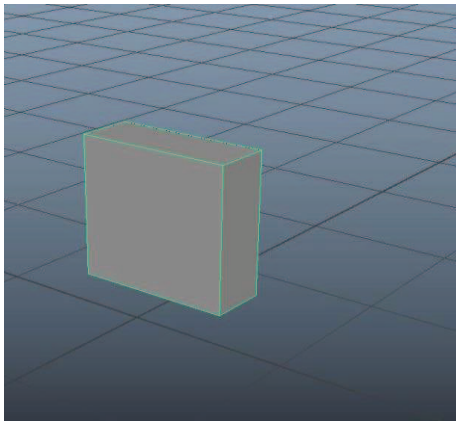
La unión une dos objetos en uno, cambiando las geometrías de los mismos y creando aristas que permitan esta unión.



La diferencia crea un objeto resultante de restar uno de los objetos al otro.



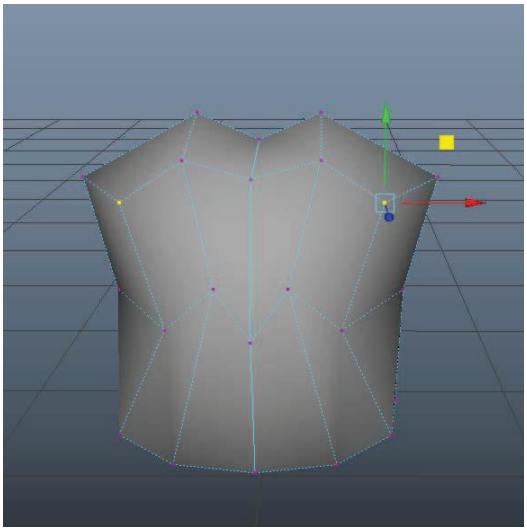
La intersección crea un objeto con las zonas de los dos objetos que se solapaban la una a la otra.



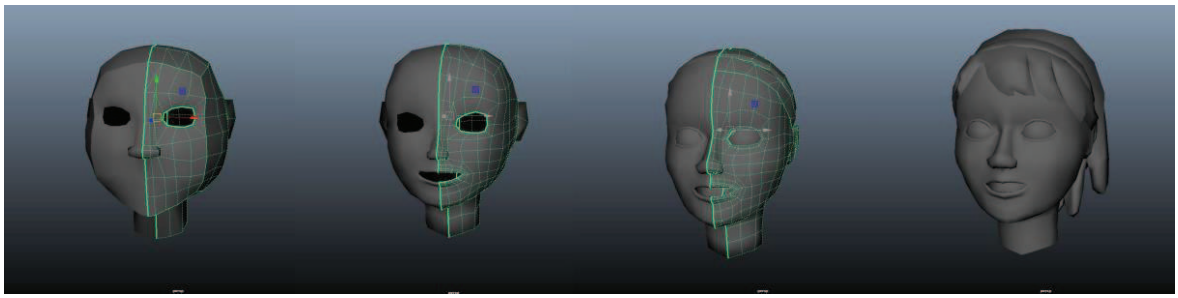
A parte de estas herramientas básicas, Maya cuenta con una gran cantidad de herramientas adicionales, algunas de ellas han sido utilizadas puntualmente.

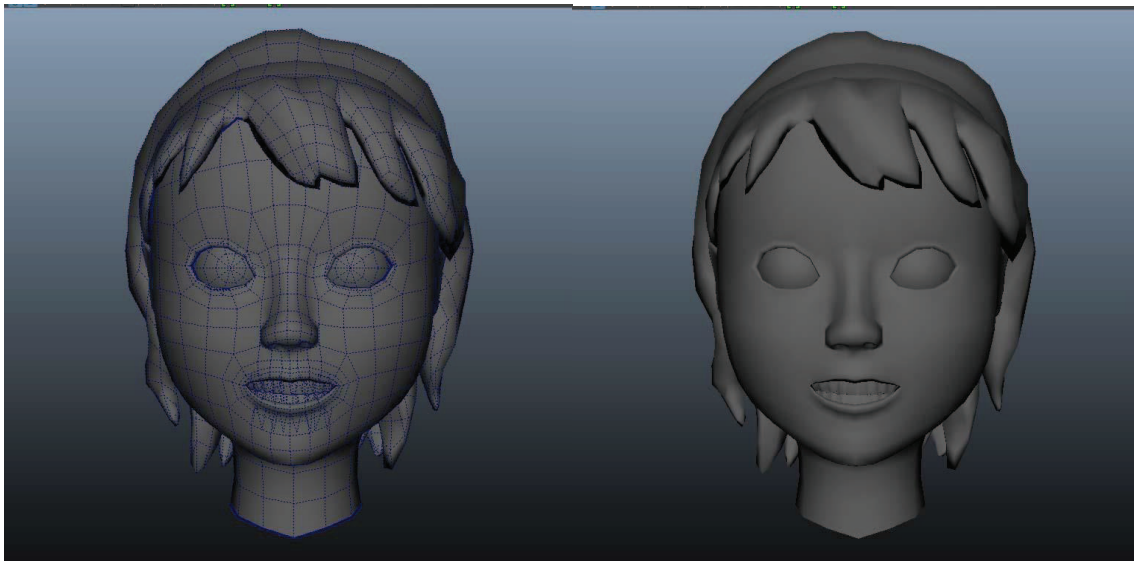
Inorgánico: Para modelar los objetos inorgánicos se ha partido casi siempre de un cubo al que se le han aplicado algunas de las herramientas anteriormente explicadas. Así, se le ha dado al objeto la forma deseada.
Los objetos se han ido ordenado en el escenario conforme eran creados.

Orgánico: Para modelar los objetos orgánicos se ha partido también de un cubo, aunque esta vez, se ha divido el cubo en dos mitades y se le ha aplicado un modificador de simetría, de forma que todo aquello que sea modificado en uno de los lados del cubo, lo será también en el otro lado. De esta forma solo deberemos modelar la mitad del objeto, lo cual nos facilita mucho el proceso, dada la dificultad del modelado orgánico.



Una vez que la simetría esta activada, el proceso que se sigue para darle la forma deseada al objeto consiste en utilizar las herramientas de mover, escalar y rotar en los distintos vértices, aristas y caras del mismo. Así se ira consiguiendo la forma deseada. En algunas ocasiones se usaran algunas de las herramientas anteriormente indicadas, dada la necesidad de añadir polígonos al objeto conforme se avanza en el modelado del mismo.

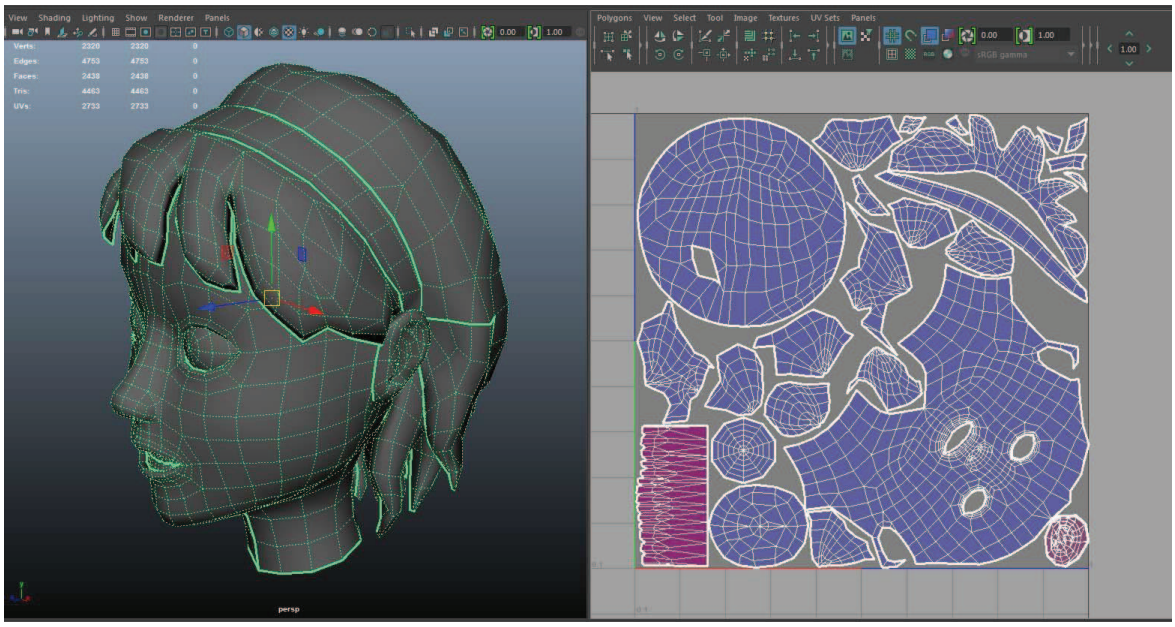




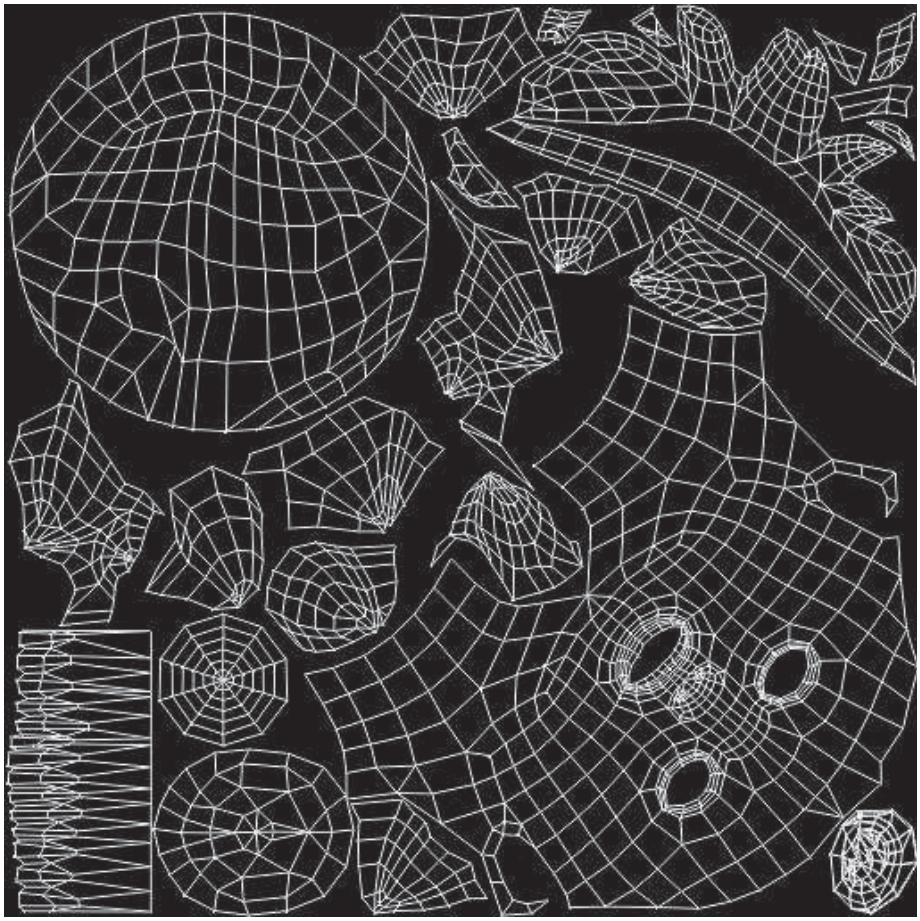
5.2.2 Texturizado

Mapeado UV: Consiste en el proceso de preparación de los objetos para poder aplicarles más adelante una textura de color.

Para poder aplicar sobre un objeto 3D una imagen 2D como textura, antes debemos darle a nuestro objeto 3D una representación 2D.
Para conseguir esto contamos en maya con herramientas que nos permiten Definir aristas del objeto como “costuras”. Si nos imaginamos que el modelo 3d fuera una tela, se podría cortar por estas costuras y estirar cada sección de tela resultante, esto sería estirado sobre una superficie plana, y de esta forma el objeto 3d habría sido convertido a 2d.
El proceso que se utiliza es el mismo, y Maya nos proporciona herramientas tanto para para definir las costuras, como para estirar las secciones resultantes y muchas más.



Una vez que se han construido los UV, se exportaran a una imagen que más tarde será coloreada, para dotar de textura al objeto.



Texturizado: Para texturizar la mayoría de los objetos, se ha utilizado Photoshop CS3, se ha importado la imagen exportada de maya, con la información de los UV, y esta ha sido coloreada utilizando los pinceles y las herramientas de selección basicas de photoshop.

Para texturizar los peinados de cada personaje se ha utilizado una técnica diferente. Para poder pintar correctamente una textura de pelo, es necesario tener grandes dotas artísticas y es un proceso largo y complicado.

Se ha utilizado una alternativa para este tipo de texturas, utilizando el programa sculptris, al que se han importado los objetos en formato .obj.

El proceso de esculpido es sencillo, ya que se cuenta con una serie de pinceles con los que se les va dando más o menos volumen a cada una de las zonas.

Así pues, se han esculpido en el propio programa cada uno de los peinados con la forma deseada, y con una cantidad mucho mayor de polígonos, para, más tarde, pintar sobre la propia escultura.

Es un proceso más sencillo ya que sculptris nos permite hacer selecciones por cavidad de los modelos esculpidos, de forma que podemos dar automáticamente un color más oscuro a las zonas más interiores del peinado. A partir de ahí, se han ido pintando los distintos mechones de pelo con los colores deseados. Cuando el proceso ha finalizado y estamos satisfechos con el resultado, exportamos una textura .jpg desde el propio programa, que obtiene la información de los UV del propio objeto.

5.2.3 Animación

La primera parte de la animación es el RIG, que consiste en el proceso de preparación del modelo para poder animarlo más adelante.

La creación del Rig consta de las siguientes partes: la creación del esqueleto, la creación de las curvas de animación y la asignación de los huesos a las curvas de animación.

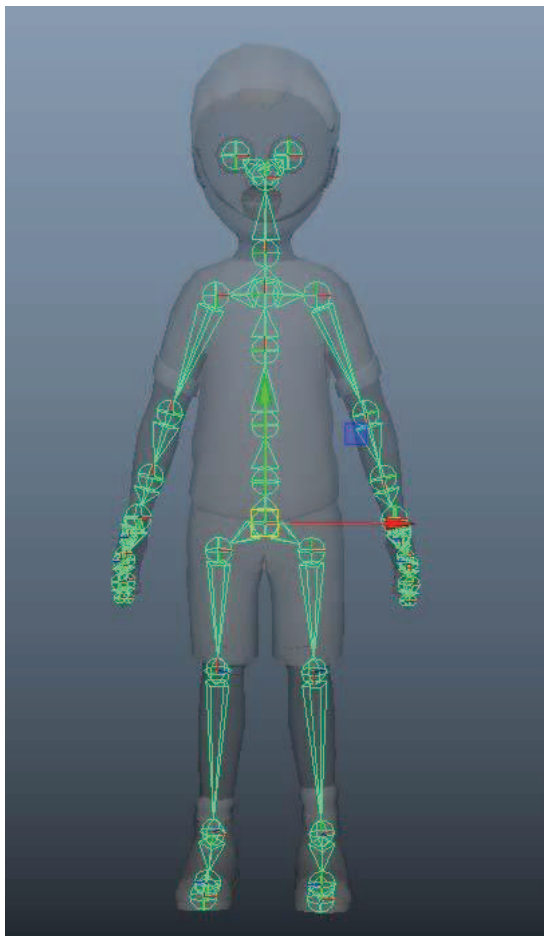
Un esqueleto es una estructura de jerarquías que nos permite controlar una malla compleja de una forma sencilla. Este hace que sus partes se muevan acorde con unas determinadas zonas más o menos rígidas o flexibles. Los esqueletos pueden usarse para animar directamente sobre ellos o bien para ser usados con cuervas de animación, lo que facilita significativamente el proceso de animación.

Esqueleto: Lo primero que ha de hacerse cuando se prepara un rig es crear un esqueleto.

En maya se crean esqueletos con la herramienta “crear hueso”, con la que se deben crear cada uno de los huesos en el lugar deseado.

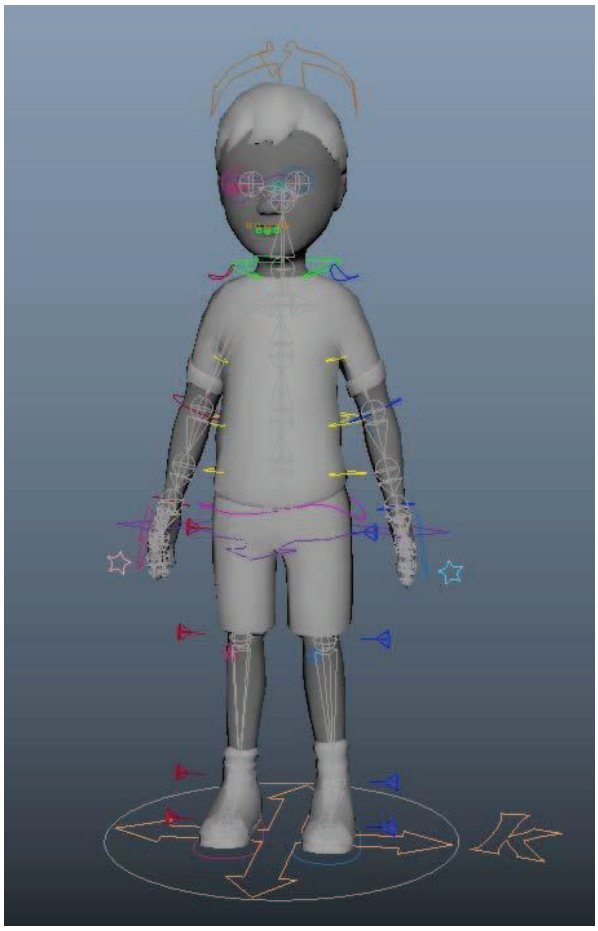
Se han de colocar los huesos desde la base de la columna hacia arriba y desde el fémur hacia el pie.

Se establece una relación parental entre los huesos, de forma que unos huesos sean padres de otros. Si el hueso A es hijo del hueso B, cuando el hueso B se mueva, el hueso A seguirá el movimiento del hueso B.



Usando el “Hypergraph” de Maya, herramienta en la que podemos ver todos los objetos de la escena con su correspondiente jerarquía, se crean las dependencias necesarias entre los huesos

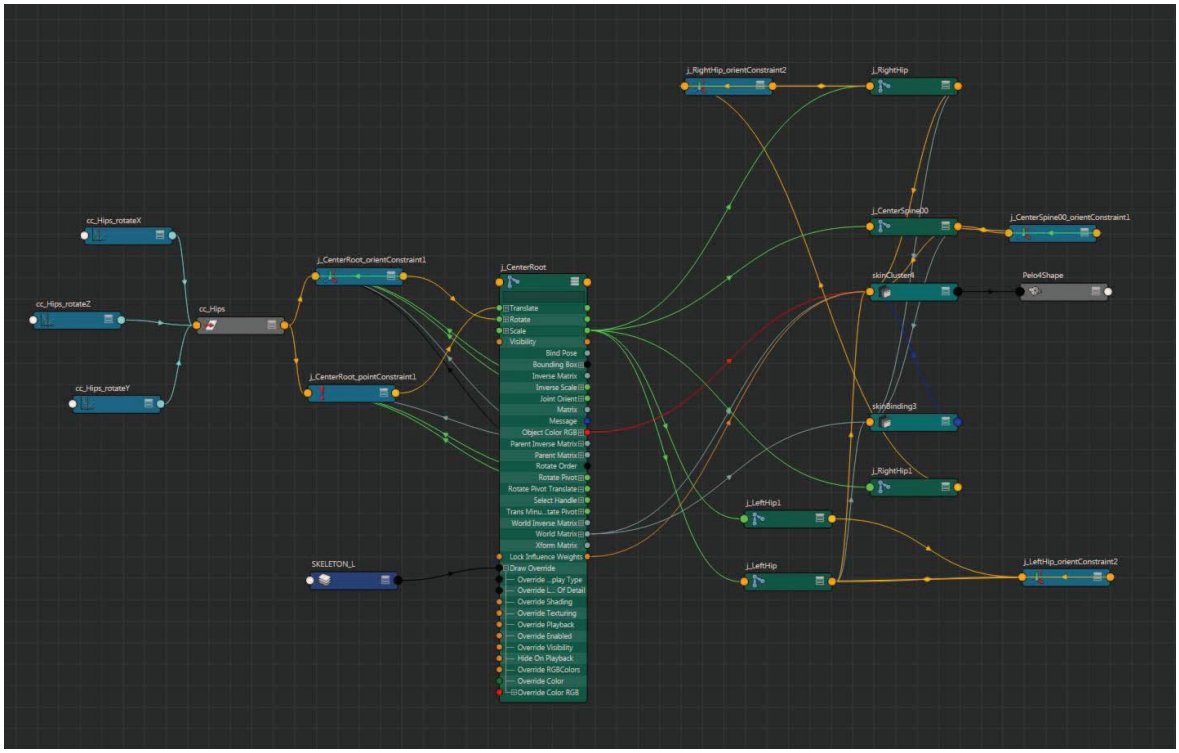
Curvas de animación: Se crean un conjunto de curvas, con la herramienta de maya “Crear curva”. Esta herramienta permite crear curvas mediante la creación de puntos cuya unión formara la propia curva.



Estas curvas se ligan al movimiento de los huesos, de forma que cuando una curva se mueva, ese hueso se moverá a su vez.

Hay varias formas de ligar las curvas a los huesos. Se pueden ligar directamente mediante una herramienta llamada “añadir clave”, en la que indicamos que curva queremos ligar a que hueso, y que propiedades del hueso se van a modificar al interactuar con la curva.

Una forma más complicada pero que permite más control es a través del editor de nodos de Maya. Este editor nos muestra cada objeto que nosotros le indiquemos, junto con todos sus atributos y el resto de objetos con los que está ligado de una forma u otra. Gracias a esta herramienta podemos editar las conexiones entre cada objeto que tengamos en nuestra escena.



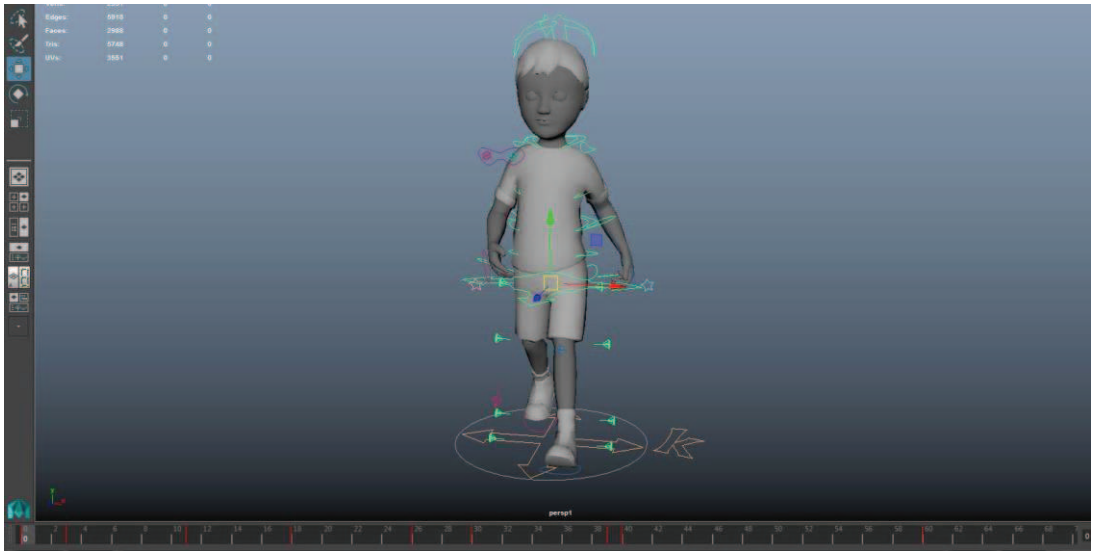
Otra forma de hacerlo es mediante las herramientas “Constrain Orient”, “Constrain Point” y “Constrain Aim”. Estas Herramientas nos permiten crear dependencias entre los huesos y las curvas de la siguiente forma:

“Constrain Orient”: Mediante esta herramienta el hueso tomara como propia la rotacion de la curva a la que lo hemos ligado.

“Constrain Point”: Mediante esta herramienta el hueso tomara como propia la traslacion de la curva a la que lo hemos ligado.

“Constrain Aim”: Mediante esta herramienta el hueso rotara, en su propio eje, orientando siempre su propio eje Z hacia la curva a la que lo hemos ligado. Este constraint es muy usado para la animación de los ojos, ya que podemos ligar los dos ojos a una curva y a partir de ese momento, siempre que movamos la curva por la escena, los ojos miraran hacia el lugar en el que esta la curva.

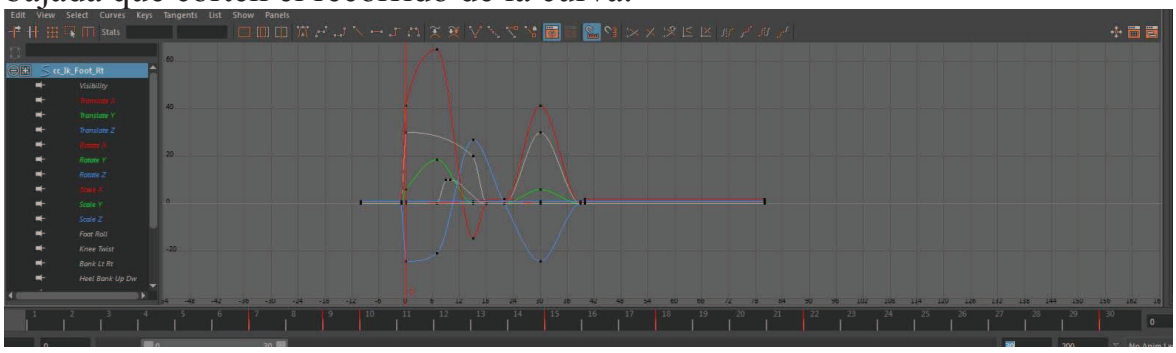
Una vez que el Rig está construido es posible empezar con la animación.
En maya contamos con una línea de tiempo, en la que podemos marcar, para un instante, una posición concreta del esqueleto, a través de nuestras curvas de animación.



En la parte inferior de la imagen superior podemos ver la línea de tiempo, que nos indica el frame en el que nos encontramos y las posiciones que hay guardadas, de la curva seleccionada, para ese frame.

Así pues, el proceso de animar consiste en darle al personaje la pose adecuada en los frames claves. En los frames que quedan entre uno con una pose y otro con otra pose diferente, el propio programa calculara las poses de transición para cada frame. De esta forma, al reproducir todos los frames, obtendremos una animación fluida.

Hay herramientas en Maya que nos facilitan el proceso de animación, una de ellas es el editor de curvas de animación, que nos muestra el movimiento de cada hueso para cada frame. Este editor es muy útil para identificar el problema cuando la animacion no es fluida, ya que en vez de mostrarnos una curva suave, nos mostrara una curva con alguna que otra pequeña subida o bajada que corten el recorrido de la curva.



Una de las partes más complicadas de la animación, aunque parezca contradictorio, es su exportación e importación a otro programa.
En este caso, fue uno de los mayores problemas del proyecto.
Los archivos fueron exportados en el formato fbx, este formato es capaz de almacenar toda la información de la animación. Por problemas de incompatibilidades entre Maya y XNA, a la hora de importar la animación al videojuego, esta se veía distorsionada.

Tras buscar información en foros de ayuda y realizar numerosas pruebas con los dos programas, se llegó a la conclusión de que algunas rotaciones extremas en determinados huesos no eran correctamente gestionadas por XNA, que les daba un valor diferente al indicado.

La única solución mediante la cual fue posible arreglar este problema fue rehacer todas las animaciones, probándolas cada pocos cambios en el videojuego y corrigiendo cualquier problema que surgiera durante el proceso.
Se trató de un proceso largo y complicado.

5.2.1 Formato de los ficheros

En este apartado, se detallan los formatos de archivos utilizados.

.jpg: Este es el formato utilizado para las texturas de los distintos objetos.

.png: Este formato se ha utilizado para exportar las imágenes con transparencia, ya que es capaz de guardar un canal "alpha" con la información de la transparencia.

.fbx: Este formato ha sido usado para exportar e importar todos los objetos de la escena, es un formato que nos permite exportar en un solo archivo tanto la geometría con la información de las texturas a las que está ligada, como información esqueletal y de animación.

obj : Este formato ha sido utilizado para exportar e importar los objetos entre maya y Sculptris. A diferencia de fbx, este formato no guarda información sobre esqueletal ni de animación.

6 Sonríe

6.1 Diseño del proyecto

6.1.1 Diseño Grafico

❖ **Diseño de interfaces:**

Para crear las distintas interfaces de las que se compone este juego, se ha tenido en cuenta la edad media del público que será usuario final del mismo. Al tratarse de niños de 7 a 14 años se han hecho interfaces con colores vivos e imágenes que les puedan llamar la atención. También se han utilizado fuentes de texto grandes para facilitar la lectura y se ha intentado poner poco texto en cada diapositiva, de esta forma evitaremos que los usuarios se aburran leyendo un gran texto e intentaremos mantener el ambiente amigable y divertido.

❖ **Diseño 3d:**

Este juego consta de unos pocos objetos 3d, que son:

- Cepillo de dientes: El cepillo de dientes es el objeto que utilizaremos con nuestro marcador, se trata de un cepillo de dientes de un tamaño pequeño, ya que siempre estará en primer plano y si fuera hecho de un tamaño mayor, taparía partes esenciales de la escena como las bacterias y se convertiría en un problema.
- Bacterias: Las bacterias son los villanos en este videojuego, pero al tratarse de un público tan joven, es preferible que no tengan un aspecto temible para evitar miedos o rechazos hacia el juego.
- Dientes: Los dientes tienen un protagonismo especial en este videojuego, y aunque normalmente se modelan con la menor cantidad de polígonos posibles, en este caso se han hecho con más definición.
- Cabeza: Al tratarse de un videojuego para niños se ha creado una cabeza de estilo estilizado, que resulta más divertido y amigable para este público.

❖ **Diseño de objetos reales:**

- Cepillo de dientes: Al ser el cepillo de dientes el objeto que contiene el marcador y es el encargado de mover el cepillo de dientes en la realidad virtual por el escenario, este objeto necesita tener el tamaño suficiente para contener el marcador, de forma que este sea reconocible por la cámara, por lo tanto se trata de un objeto grande. Ha sido creado con colores vivos para llamar la atención de los usuarios y hacer que sea un objeto divertido.

6.1.2 Modulado de la Arquitectura

No se trata de una arquitectura complicada. *Sonríe* consta de una única clase **Game1** donde se encuentra toda la lógica del juego.

Esto es debido a que la complejidad de este juego ha consistido en la implementación de la realidad aumentada como su punto más fuerte.

6.1.2.1 Identificando los métodos

Como ya se ha mencionado, *Sonríe* no se trata de un juego de una gran carga de código (dispone de una única clase) por lo que se detallan todos los métodos.

- ***Game1:*** Método principal del juego. Donde se inicializan valores importantes como la resolución de la pantalla, el directorio que contendrá los archivos que se utilicen y demás.

- ***Initialize:*** Método donde se inicializan los elementos principales del juego como las interfaces o la escena.

- ***LoadContent***: Método que inicializa los valores necesarios desde el principio del juego como el estado del ratón. También carga el contenido.

- ***StartGame:*** Método que inicia todos los objetos necesarios para jugar. Esto conlleva todos los objetos 3D, la cámara, las luces…

- ***BactCollision:*** Método evento que se llamará cuando se produzca la colisión entre el cepillo de dientes y una bacteria.

- ***Update:*** Método que irá actualizando la información del juego según se encuentre en la pantalla inicial, jugando o la pantalla final.

- ***Draw:*** Método que irá actualizando todo lo que deba dibujarse en la pantalla: interfaces y objetos 3D.

- ***DisposeObjects:*** Quita de la escena todos los objetos 3D de cara a finalizar el juego.

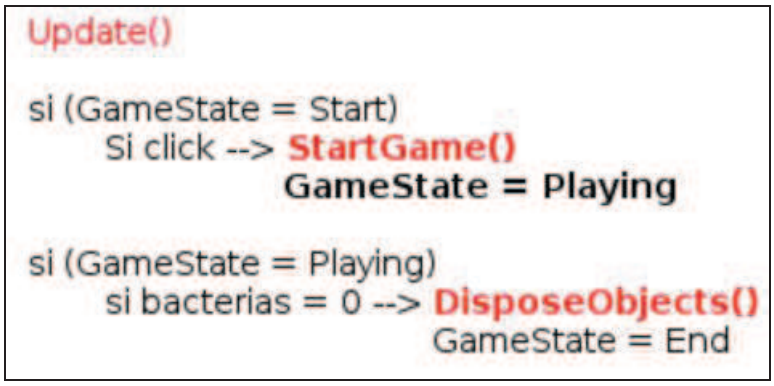
- ***Dispose:*** Método que libera la escena.

- ***UnloadContent***: Método que libera el contenido.

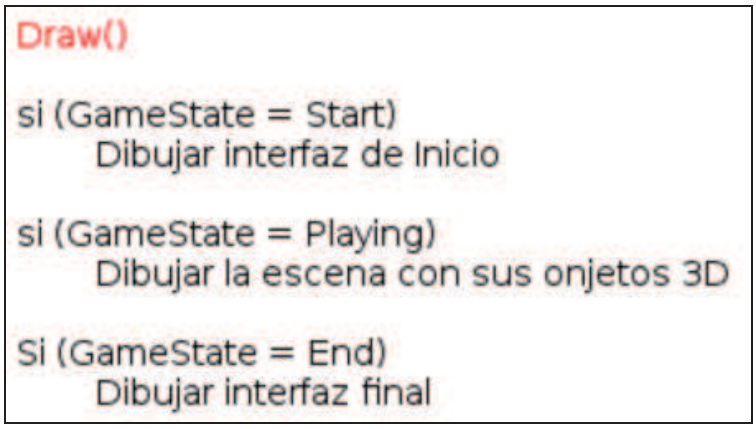
6.1.2.2 Relación entre métodos

Los métodos descritos se relacionan entre sí de una forma u otra. Si bien es cierto que los métodos propios de un juego en XNA como son *Initialize, Update, Draw, LoadContent, UnLoadContent, Dispose* se llaman automáticamente.

En la siguiente imagen se puede ver cómo funcionan los métodos *Draw* y *Update* en este juego.



Diseño del método Update en *Sonríe*



Diseño del método Draw en Sonríe

6.2 Implementación del proyecto

Para la creación de los objetos 3d se han utilizado las mismas técnicas que para la creación de los objetos del proyecto Safe trip, por lo que no se detallaran en este apartado.

6.2.1 Formato de los ficheros

En este apartado, se detallan los formatos de archivos utilizados. Como en su mayoría se han explicado anteriormente en la implementación de *Safe Trip* sólo se entra en el detalle de los siguientes tipos de archivo:

Mapas (.xml) → *Extensible Markup Language* o Lenguaje de marcas extensible, no es un lenguaje en sí, sino un metalenguaje extensible de etiquetas que permite definir lenguajes para diferentes necesidades. Se puede emplear en bases de datos, editores de texto, hojas de cálculo, etc. y permite la compatibilidad entre sistemas para compartir información de una manera segura, fiable y sencilla.

Librerías dinámicas (.dll) → Dynamic-Link Library o Librería de enlace dinámico, son archivos que contienen funciones con código ejecutable que se cargan bajo la demanda de un programa u otras dll. Estos formatos son exclusivos de los sistemas operativos Windows, no así el concepto.

6.2.2 Cámara y marcadores

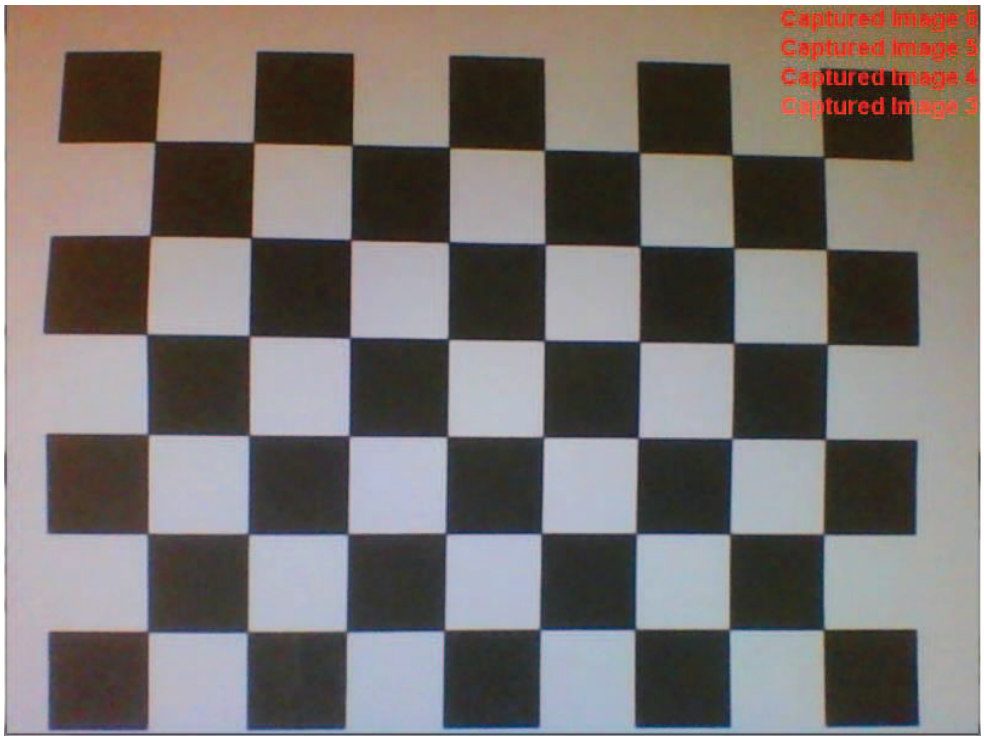
Para poder implementar el uso de la realidad aumentada mediante marcadores se tiene primero que configurar la cámara para que capture bien los marcadores y también se tiene que definir qué marcadores deberá identificar el programa.

Cámara

Para poder calibrar la cámara GoblinXNA ofrece un programa llamado *CameraCalibration* que se encuentra en la carpeta *GoblinXNA4.0\tools*.Se trata de una implementación en C# de ese mismo programa ofrecido por ALVAR en C++.

Una vez que se ejecuta el programa, hay que situar delante de la cámara que se desea calibrar un folio, propocionado por ALVAR, que consta de cuadrados blancos y negros.

El programa pide que se mantenga este folio con la imagen hasta que haga 50 capturas y finalizará. Nos irá avisando arriba a la derecha de la cantidad de capturas tomadas.



Calibración de una cámara para GoblinXNA

Una vez terminado, se habrá creado un archivo llamado *calib.xml* que se encuentra en la ruta *tolos/CameraCalibration/bin/x86/Debug* propia de la carpeta de GoblinXNA.

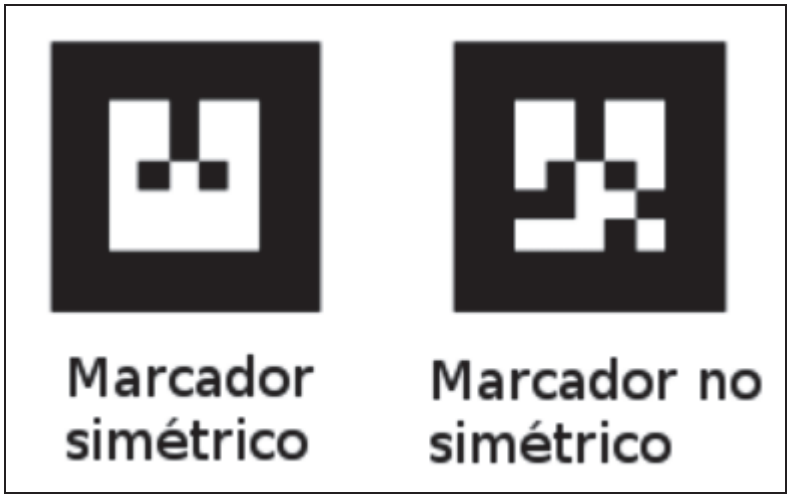
Para finalizar, se debe copiar este archivo dentro de la carpeta del proyecto en la ruta *bin/x86/Debug*.

Si no se realiza esto, se cogerá un archivo por defecto que contiene el proyecto y la calibración de la cámara no será la más adecuada por lo que no reconocerá bien todos los marcadores.

Marcadores

Como ya se ha explicado en un punto anterior de la memoria, los marcadores proporcionados por ALVAR ya están registrados en el programa y se pueden usar sin problemas.

Para este proyecto sólo se ha necesitado de un marcador para identificar el cepillo de dientes. Sin embargo, éste ha debido de ser simétrico. Esto es así porque la cámara se encuentra de manera invertida y cualquier otro marcador no simétrico no lo identifica.



Marcador simétrico y no simétrico de ALVAR

Todos los marcadores se identifican por un número. Para el cepillo se eligió el marcador número 15. Para definir en el programa los marcadores que han de ser captados, se escribe en un archivo *.xml* el marcador de la forma que se presenta en la siguiente imagen.

Toothbrush.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<multimarker markers="1">
  <marker index="15" status="2">
    <corner x="9.6" y="-16.2" z="0" />
    <corner x="42" y="-16.2" z="0" />
    <corner x="42" y="16.2" z="0" />
    <corner x="9.6" y="16.2" z="0" />
  </marker>
</multimarker>
```

Marcador para el cepillo de dientes

6.2.3 Implementación del código

6.2.3.1 Estructura básica de un juego en XNA

Antes de comenzar la explicación del código del juego es importante conocer cómo funciona XNA.

El código inicial generado en estos proyectos consta de la clase estática Program que simplemente crea una instancia de la clase Game y la pone en funcionamiento. Es en esta clase Game, que a su vez hereda de la clase "padre" Game del Framework XNA, donde están los cinco métodos más importantes y que formarán el esqueleto del proyecto, además del propio constructor de la clase. Estos métodos son los siguientes:

- Método Initialize: este método es el primero que automáticamente se llama cuando el juego es ejecutado. En él se deben introducir todas las variables que necesitan ser inicializadas una vez al comienzo del juego.

- Método LoadContent: este método es automáticamente llamado una vez por juego, y en él se carga todo lo relacionado con los gráficos, como las texturas, las fuentes, etc. o los sonidos.

- Método UnloadContent: este método es usado para liberar los recursos cargados en el método LoadContent. Es llamado automáticamente.

- Método Update: este método es llamado constantemente (por defecto 60 veces por segundo) en tiempo de ejecución, y permite actualizar todo lo que está siendo ejecutado en ese momento. En este método se deben incluir cosas como el movimiento del personaje, comprobación de colisiones, actualización de los dispositivos de entrada, o ejecución de archivos de audio.

- Método Draw: en este método se deben colocar los objetos que tienen que ser dibujados en cada frame, es decir, todo aquello relacionado con la interfaz gráfica. Al igual que Update es llamado constantemente.

6.2.3.2 Implementación del código

Como ya se ha comentado, el código de *Sonríe* no es muy extenso. Sin embargo, se van a explicar sus singularidades y complejidades además de repasar de forma general todo el código.

Inicio del juego

El juego tiene un enumerador para saber en qué estado se encuentra.

```
public enum GameState
{
    StartMenu,
    Playing,
    End
}
GameState _GameState;
```

Dependiendo de este estado, en los métodos *Update* y *Draw* se cambiará lo que hace el programa y lo que se dibuja en pantalla.

- En el caso del estado *StartMenu*, se espera por un evento de ratón para pasar al siguiente estado y se dibuja una pantalla de inicio.
- En el estado *Playing* se actualiza la escena con todos los objetos 3D y se dibuja la escena.
- En el estado *End* se dibuja la pantalla final.

Cuando se recibe un evento de teclado en el estado *StartMenu* se llama al método *StartGame* donde crea la luz, la cámara y los objetos 3D.

Objetos 3D

Los objetos 3D utilizados se tratan de la cara de una niña con la boca abierta, las bacterias y el cepillo de dientes.

Una vez que se llama a *StartGame* indicando que el juego ha comenzado, se llama al método *CreateObject*. En él se define primero el modelo de la niña y del cepillo de dientes con sus respectivos *GeometryNode* y *TransformNode*. El primero contiene la información del modelo 3D, el segundo contiene al primero como hijo y sirve para poder realizar movimientos.

```
ModelLoader loader = new ModelLoader();

//Cargamos el modelo fbx
Model CepilloDientesModel = (Model)loader.Load("", "CepilloDientes");
Model BocaModel = (Model)loader.Load("", "Niña");
```

Los modelos se inician dándoles valores a sus físicas y al propio modelo (como es las sombras).

```
//Creamos el cepillo de dientes:
CepilloDientesNode = new GeometryNode("CepilloDientes1");
CepilloDientesNode.Model = CepilloDientesModel;
CepilloDientesNode.AddToPhysicsEngine = true;
CepilloDientesNode.Physics.Shape = ShapeType.Box;
CepilloDientesNode.Physics.ShapeData = new List<float> {8,8,8};
CepilloDientesNode.Physics.Collidable = true;
CepilloDientesNode.Model.ShadowAttribute = ShadowAttribute.ReceiveCast;
CepilloDientesNode.Model.Shader = new SimpleShadowShader(scene.ShadowMap);
CepilloDientesNode.Model.ShaderTechnique = "DrawWithShadowMap";
```

Por último, cada *TransformNode* se añade como hijo a la escena.

```
scene.RootNode.AddChild(CepilloDientesTransNode);
```

Al nodo de la cara se le da una posición bastante acercada con respecto a la cámara para que se puedan ver bien los dientes.

En cuanto a las bacterias, se generan en un bucle pasándoles una lista de posiciones donde se colocarán:

```
rand = 7;
//Creamos array de las posiciones de los dientes
List<Vector3> dientes = new List<Vector3>();
dientes.Add(new Vector3(0, -19, 20));
dientes.Add(new Vector3(1, -20.5f, 20));
dientes.Add(new Vector3(2, -19, 20));
dientes.Add(new Vector3(4, -20, 20));
dientes.Add(new Vector3(3, -19, 20));
dientes.Add(new Vector3(-1, -19, 20));
dientes.Add(new Vector3(-2, -20, 20));

//Creamos un array con posiciones aleatorias para
int count = 0;

for (int i = 0; i < rand; i++){
    CreateBact(dientes, count);
    count++;
```

En el método *CreateBact* crea el model 3D de las bacterias eligiendo una bacteria aleatoria por cada vez (hay 3 tipos de bacterias diferentes). Además crea el TransformNode para los movimientos y escalado de la bacteria. Se le da una posición inicial marcada por el array *dientes*.

Dentro de *CreateBact* se define también el par de colisión entre el cepillo de dientes y la bacteria

```
//Creamos el evento de la colisión
NewtonPhysics.CollisionPair pair = new NewtonPhysics.CollisionPair(BacteriaNode.Physics, CepilloDientesNode.Physics);
((NewtonPhysics)scene.PhysicsEngine).AddCollisionCallback(pair, BactCollision);
```

Para ello se utilizan las físicas de los *GeometryNode* de cada modelo.

Colisión entre objetos

Como se ha comentado en el punto anterior, se crea un evento que salta cuando los *GeometryNode* se cruzan.

El método *BactCollision* se llama cuando se produce tal colisión recibiendo como parámetro el par de colisiones.

Dentro del método se recoge la posición actual del marcador (que será la del cepillo de dientes y que se explicará en el siguiente punto). Para similar un movimiento de arriba abajo del cepillo de dientes sobre las bacterias se restan las posiciones de ambos en las *y* y se calcula el valor absoluto de la diferencia.

Si se ha producido una colisión, se consigue de entre el par de colisiones el *GeometryNode* de la bacteria actual afectada. Con este se saca su padre que es un *TransformNode*.

```
private void BactCollision(NewtonPhysics.CollisionPair pair)
{
    //Cogemos la posición del marcador
    Matrix markerPosition = markerNode.WorldTransformation;
    markerPosition.Decompose(out scale, out rotation, out translation);

    dif = posY - translation.Y;
    if (Math.Abs(dif) >= 2)
    {
       //Obtenemos el objeto geometrynode asociado a esas físicas
        GeometryNode BacteriaActual = new GeometryNode();
        BacteriaActual = (GeometryNode)pair.CollisionObject1.Container;

        //Obtenemos el padre de BacteriaActual, que será un TransforNode
        TransformNode BacteriaTransActual = new TransformNode();
        BacteriaTransActual = (TransformNode)BacteriaActual.Parent;
```

Tras esto se obtiene el tamaño de la bacteria. Si el tamaño es ya lo suficientemente pequeño, la bacteria se elimina de la boca. Si el tamaño aún no es lo suficientemente pequeño, se escala su tamaño.

```
//Obtenemos el tamaño de las x de la bacteria
bacteriaSize = BacteriaTransActual.Scale.X;

//Bajamos el tamaño de la bacteria, sólo si ya no es demasiado pequeña
if (bacteriaSize <= 0.3)
{
    BocaTransNode.RemoveChild(BacteriaTransActual);
    //actualizamos el número total de bacterias
    rand--;
}
else {
    BacteriaTransActual.Scale = BacteriaTransActual.Scale * 0.75f;
    bacteriaSize = BacteriaTransActual.Scale.X;
}
```

Marcador – Cepillo de dientes

Como ya se ha explicado anteriormente, el cepillo de dientes se identifica con un marcador. Para implementar esto, primero se llama a la función *SetUpMarkerTracking* que es la que inicia todos los elementos necesarios para que el programa empiece a captar marcadores.

```
private void SetupMarkerTracking(){

    IVideoCapture captureDevice = null;
        captureDevice = new DirectShowCapture();
        captureDevice.InitVideoCapture(0, FrameRate._60Hz, Resolution._800x600,
            ImageFormat.R8G8B8_24, false);
    scene.AddVideoCaptureDevice(captureDevice);

    ALVARMarkerTracker tracker = new ALVARMarkerTracker();
    tracker.MaxMarkerError = 0.02f;
    tracker.InitTracker(captureDevice.Width, captureDevice.Height, "calib.xml", 32.4f);

    scene.MarkerTracker = tracker;
    scene.ShowCameraImage = true;
}
```

Una vez llamada esta función, se crea una instancia de la clase MarkerNode y se añade a la escena.

```
markerNode = new MarkerNode(scene.MarkerTracker, "Toothbrush.xml");
scene.RootNode.AddChild(markerNode);
```

Una vez definidos estos atributos, se puede acceder al valor del marcador en todo momento mediante *markerNode*.

Con ello se consigue dar al cepillo de dientes la posición del marcador. En el método *Update* que se ejecuta todo el rato, se actualiza este valor.

```
//Actualizamos la posición del cepillo en la pantalla con la del marcador
Matrix markerPosition = markerNode.WorldTransformation;
markerPosition.Decompose(out scale, out rotation, out translation);
scale = new Vector3(scale.X * 30f, scale.Y * 30f, scale.Z * 30f);
CepilloDientesTransNode.Scale = scale;
CepilloDientesTransNode.Rotation = rotation;
CepilloDientesTransNode.Translation = translation;
posY = translation.Y;
```

Posición de la cara

Para facilitar al usuario el cepillado de los dientes, se gira la cara a la izquierda o hacia la derecha si movemos el cepillo hacia esas direcciones. Controlamos esto con la posición en el eje de las *x* del marcador.

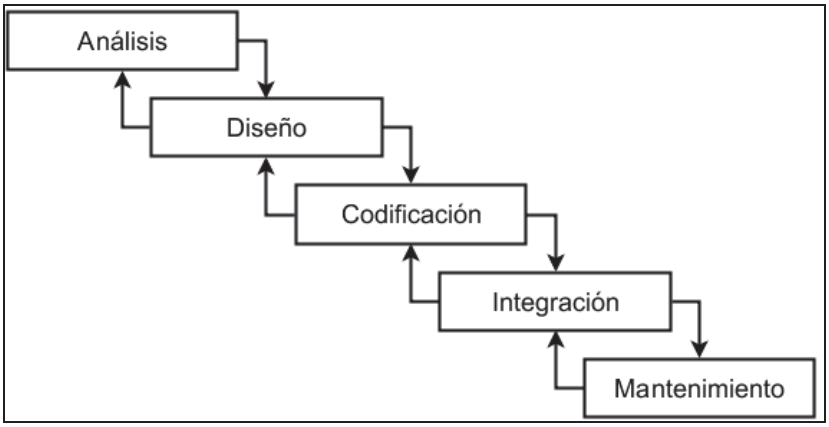
```
if (Estado == 0)//Si está en estado central, dos posibles movimientos
{
    if (translation.X < -90)//Si el cepillo se va muy a la izq, cambiamos el estado de la man
    {
        Estado = -1;
        //Vamos a darle una nueva rotacion y traslación a la boca
        rotation = Quaternion.CreateFromAxisAngle(Vector3.UnitY, MathHelper.ToRadians(60));

        BocaTransNode.Rotation = rotation;
        BocaTransNode.Translation = BocaTransNode.Translation + new Vector3(-220, 0, 100);

    }
    else if (translation.X > 90)//Si el cepillo se va muy a la dcha, cambiamos el estado de l
    {
        Estado = 1;
        rotation = Quaternion.CreateFromAxisAngle(Vector3.UnitY, -MathHelper.ToRadians(60));
        BocaTransNode.Rotation = rotation;
        BocaTransNode.Translation = BocaTransNode.Translation + new Vector3(220, 0, 100); ;
    }
}
```

7 Gestión del proyecto

7.1 Ciclo de vida del proyecto

7.1.1 Sonríe

Para este primer proyecto se siguió un modelo en cascada. Se trata de un modelo de desarrollo que ordena las etapas del proceso de desarrollo de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, se comprueba mediante una revisión si el proyecto está listo para avanzar a la siguiente. Es el modelo más básico de todos los existentes y todos los demás se basan en él.



Ciclo de vida en cascada

La fase de Análisis fue la más importante puesto que hubo que iniciarse en el uso de ciertas herramientas, destacando el uso de marcadores y todo lo ello conllevó.

En la fase de Diseño se define la arquitectura del sistema y se desarrolla el diseño detallado y el modelo estático y dinámico de la aplicación, además de modelarse los elementos en 3D. En el caso de este proyecto, se trató de una fase sencilla puesto que en análisis previo y gracias al modelo en cascada, facilitó mucho esta fase.

La fase de Codificación es aquella en la que se implementa el diseño anterior. También fue una fase sencilla para este proyecto porque se encontraron fácilmente la solución a los requisitos del juego en la fase anterior.

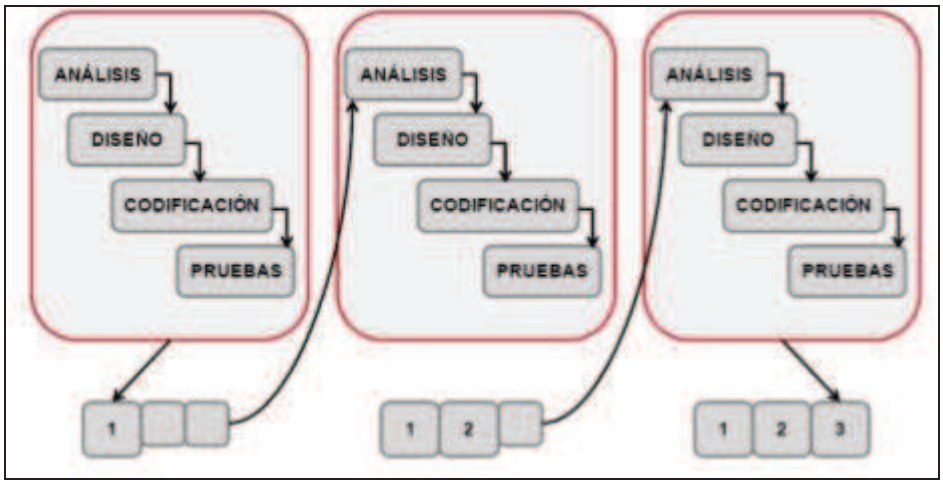
La fase de Integración y la sucesiva de Mantenimiento tienen el fin de validar el trabajo realizado. En este caso las pruebas no dieron problemas y se llegó fácil a la solución final del proyecto.

7.1.2 Safe Trip

Para el desarrollo de este proyecto se ha seguido un ciclo de vida iterativo, debido a que existen funcionalidades que deben estar implementadas antes que otras, y para ir validando el sistema según se va construyendo.

Se trata de un modelo de desarrollo con un conjunto de tareas agrupadas en pequeñas etapas repetitivas (iteraciones). Es uno de los modelos más utilizados en los últimos tiempos ya que, como se relaciona con novedosas estrategias de desarrollo de software y una programación extrema, es empleado en metodologías diversas.

El modelo consta de diversas etapas de desarrollo en cada incremento, las cuales inician con el análisis y finalizan con la instauración y aprobación del sistema.



Ciclo de vida iterativo

El diagrama previo muestra de manera bastante aproximada el modo de desarrollo seguido, ya que los requisitos funcionales han ido incorporándose a lo largo de todo el proceso. Hay que tener en cuenta en todo momento que había dos personas trabajando en el proyecto y que las fases se han ido acomodando en función de las dos.

La fase de Análisis ha sido posiblemente la más importante en las iteraciones, ya que además de capturar los requisitos funcionales de la aplicación, ha sido necesario un estudio del proyecto y una formación en las distintas herramientas empleadas.

La fase de Diseño cambió de una iteración a otra debido a la complejidad inherente en cada iteración. El modelaje 3D supuso más trabajo en unas iteraciones que en otras. Del mismo modo, el modelado de la arquitectura fue mayor en unas iteraciones que en otras.

La fase de Codificación es la fase en la que más tiempo se ha empleado. Generalmente, el diseño en cada iteración fue un diseño complejo que conllevó una implementación también compleja.

Por último, la fase de Integración y Pruebas se ha ido realizando a medida que se codificaban las funcionalidades de la aplicación, con el fin de ir validando el trabajo realizado.

7.2 Coordinación

Como ya se ha explicado anteriormente, el proyecto se ha realizado algunas partes en conjunto. En concreto las partes de diseño e implementación del juego *Safe Trip*. Esto ha supuesto una coordinación del trabajo que ha sido clave para la gestión del proyecto.

Lo primero que se realizó fue el plan de trabajo para que los dos miembros estuvieran conformes y, tras esto, se empezó con el desarrollo. Sin embargo las tareas de planificación y coordinación no terminaron una vez que se empezó con el desarrollo del videojuego, sino que se extendieron a lo largo de todo el proceso de implementación.

Para que esto fuera posible y se pudiera mantener al tanto a ambas partes del estado de desarrollo y de la detección de variantes con respecto al plan se siguieron ciertos puntos:

- Mantener actualizado el plan de trabajo y corregir y adaptar la planificación continuamente según el transcurso del desarrollo.

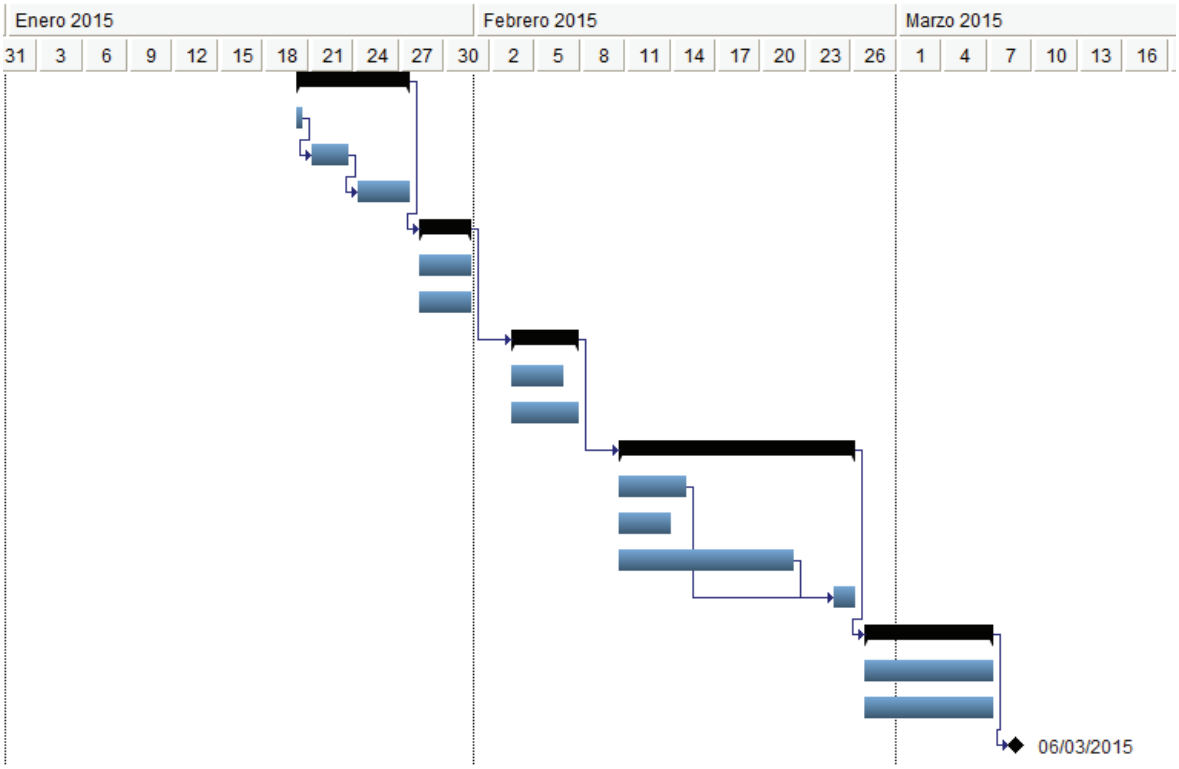
- Prácticamente en todo momento trabajo en un mismo laboratorio. Facilitando así la comunicación entre los dos miembros y la discusión y desarrollo de todas las fases de cada iteración.

- Comunicación diaria a través de chat en caso de no poder realizarse el trabajo conjunto en un mismo lugar.

7.3 Planificación del proyecto

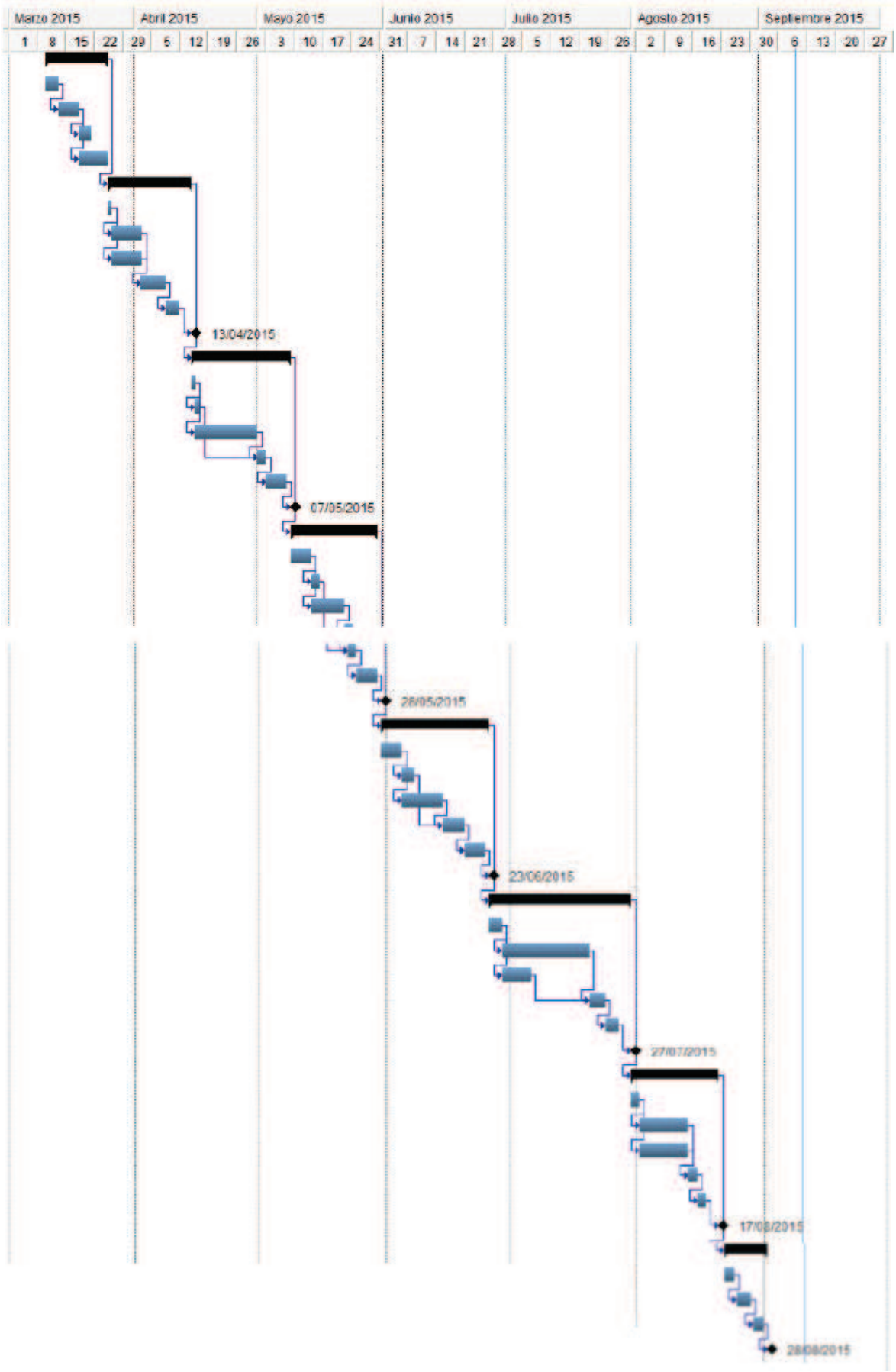
7.3.1 Sonríe

		Nombre	Duración	Inicio	Fin	Predecesoras
1		**⊟Fase de inicio**	**6d**	**19/01/2015**	**26/01/2015**	
2		Análisis general del proyecto	1d	19/01/2015	19/01/2015	
3		Definición de requisitos	3d	20/01/2015	22/01/2015	2
4		Familiarización/instalación de herramientas necesarias	2d	23/01/2015	26/01/2015	3
5		**⊟Fase de Análisis**	**4d**	**27/01/2015**	**30/01/2015**	**1**
6		Definición de la estructura del juego	4d	27/01/2015	30/01/2015	
7		Familiarización con marcadores	4d	27/01/2015	30/01/2015	
8		**⊟Fase de Diseño**	**5d**	**02/02/2015**	**06/02/2015**	**5**
9		Diseño del cepilo de dientes 3D	4d	02/02/2015	05/02/2015	
10		Diseño de atributos/métodos en código	5d	02/02/2015	06/02/2015	
11		**⊟Fase de Implementación**	**12d**	**09/02/2015**	**24/02/2015**	**8**
12		Implementación 3D del cepillo de dientes	5d	09/02/2015	13/02/2015	
13		Implementación física del cepillo de dientes	4d	09/02/2015	12/02/2015	
14		Implementación del código del juego	10d	09/02/2015	20/02/2015	
15		Unión de parte gráfica y parte programada	2d	23/02/2015	24/02/2015	12,14
16		**⊟Fase de Pruebas y correcciones**	**7d**	**25/02/2015**	**05/03/2015**	**11**
17		Pruebas con diferentes cámaras	7d	25/02/2015	05/03/2015	
18		Corrección de la posición de objetos respecto a la cámara	7d	25/02/2015	05/03/2015	
19		Juego finalizado	1d?	06/03/2015	06/03/2015	16


Enero 2015
Febrero 2015
Marzo 2015
31 3 6 9 12 15 18 21 24 27 30 2 5 8 11 14 17 20 23 26 1 4 7 10 13 16
06/03/2015

7.3.2 Safe Trip

		Nombre	Duración	Inicio	Fin	Predecesoras	Recursos
1		**Fase de inicio**	**15d**	**09/03/2015**	**24/03/2015**		
2		Análisis General del Proyecto	4d	09/03/2015	12/03/2015		
3		Definición de requisitos	4d	12/03/2015	17/03/2015	2	
4		Familiarización/instalación de herramientas necesarias	4d	17/03/2015	20/03/2015	3	
5		Creación de la estructura del juego	7d	17/03/2015	24/03/2015	3	
6		**Primera fase: Motor gráfico**	**19d?**	**24/03/2015**	**13/04/2015**	**1**	
7		Análisis de un aeropuerto	1d?	24/03/2015	25/03/2015		
8		Diseño/implementación 3D básico	7d	25/03/2015	01/04/2015	7	
9		Diseño/implementación en c# básico	7d	25/03/2015	01/04/2015	7	
10		Unión de parte gráfica y parte programada	5d	01/04/2015	07/04/2015	8,9	
11		Pruebas y correcciones	5d	07/04/2015	10/04/2015	10	
12		Motor gráfico finalizado	1d?	10/04/2015	13/04/2015	11	
13		**Segunda fase: Personaje principal**	**24d**	**13/04/2015**	**07/05/2015**	**6**	
14		Análisis del personaje	1d	13/04/2015	14/04/2015		
15		Diseño/implementación 3D del personaje	2d	14/04/2015	15/04/2015	14	
16		Diseño/implementación programada del personaje	15d	14/04/2015	29/04/2015	14	
17		Unión de parte gráfica y parte programada	3d	29/04/2015	01/05/2015	15,16	
18		Pruebas y correcciones	4d	01/05/2015	06/05/2015	17	
19		Personaje principal finalizado	1d	06/05/2015	07/05/2015	18	
20		**Tercera fase: Interacciones con áreas**	**20d**	**07/05/2015**	**28/05/2015**	**13**	
21		Análisis de las interacciones	4d	07/05/2015	12/05/2015		
22		Diseño/implementación 3D de las áreas	3d	12/05/2015	14/05/2015	21	
23		Diseño/implementación programada de las áreas	8d	12/05/2015	20/05/2015	21	
24		Unión de parte gráfica y parte programada	3d	20/05/2015	22/05/2015	22,23	
25		Pruebas y correcciones	4d	22/05/2015	27/05/2015	24	
26		Interacciones finalizadas	1d	27/05/2015	28/05/2015	25	
27		**Cuarta fase: Inventario/interacción con objetos**	**24d**	**28/05/2015**	**23/06/2015**	**20**	
28		Análisis de los objetos de inventario	4d	28/05/2015	02/06/2015		
29		Diseño/implementación 3D de los objetos	4d	02/06/2015	05/06/2015	28	
30		Diseño/implementación programada de interacción con objet	10d	02/06/2015	12/06/2015	28	
31		Unión de parte gráfica y parte programada	5d	12/06/2015	17/06/2015	29,30	
32		Pruebas y correcciones	4d	17/06/2015	22/06/2015	31	
33		Interacciones finalizadas	1d	22/06/2015	23/06/2015	32	
34		**Quinta fase: Personajes no interactivos**	**32d**	**23/06/2015**	**27/07/2015**	**27**	
35		Análisis de los personajes del aeropuerto	4d	23/06/2015	26/06/2015		
36		Diseño/implementación 3D de los personajes	20d	26/06/2015	17/07/2015	35	
37		Diseño/implementación programada de los personajes	7d	26/06/2015	03/07/2015	35	
38		Unión de parte gráfica y parte programada	2d	17/07/2015	21/07/2015	36,37	
39		Pruebas y correcciones	5d	21/07/2015	24/07/2015	38	
40		Personajes finalizados	1d	24/07/2015	27/07/2015	39	
41		**Sexta fase: Animación de los personajes**	**20d**	**27/07/2015**	**17/08/2015**	**34**	
42		Análisis de la animación de los personajes	3d	27/07/2015	29/07/2015		
43		Diseño/implementación 3D de la animación	10d	29/07/2015	10/08/2015	42	
44		Diseño/implementación programada de la animación	10d	29/07/2015	10/08/2015	42	
45		Unión de parte gráfica y parte programada	3d	10/08/2015	12/08/2015	43,44	
46		Pruebas y correcciones	3d	12/08/2015	14/08/2015	45	
47		Animación finalizada	1d	14/08/2015	17/08/2015	46	
48		**Fase Final**	**9d?**	**18/08/2015**	**28/08/2015**	**41**	
49		Análisis de todo el contenido	3d	18/08/2015	20/08/2015		
50		Idfentificación de fallos	2d	21/08/2015	24/08/2015	49	
51		Correcciones	3d	25/08/2015	27/08/2015	50	
52		Producto final	1d?	28/08/2015	28/08/2015	51	



8 Conclusiones y líneas futuras

8.1 Conclusiones

Como conclusión general, cabe destacar que se ha conseguido la implementación de dos juegos complejos cumpliendo en uno de ellos con los objetivos de implementación de realidad aumentada.

Se ha estudiado en profundidad el funcionamiento de Microsoft XNA, demostrando que es un *framework* que ofrece la oportunidad de desarrollar de forma relativamente sencilla juegos no tan sencillos en 3D con la posibilidad de introducir elementos de la vida real.

Además, para poder implementar los dos proyecto, se ha tenido que estudiar el lenguaje de programación C# logrando un buen nivel de programación del mismo. Este conocimiento será útil en un futuro para implementar otros posibles proyectos.

De la misma forma, el nivel de conocimientos y técnicas de Maya, así como la rapidez de su uso ha incrementado de forma notable Debido a la gran cantidad de objetos y personajes que se han creado con el programa. Además, se ha profundizado en el modelado orgánico, adquiriendo un nivel superior de maestría.

También se ha familiarizado con la utilización de una variedad de software como Visual Studio, Sculptris, Photoshop, etc. Proporcionando en algunos casos una iniciación a los mismos y enriqueciendo la variedad de software conocido.

Por último, la realización de partes por separado del proyecto ha conllevado un aprendizaje de trabajo en equipo. Es un aprendizaje importante para en un futuro realizar más trabajos en equipo.

Como experiencia personal, la realización del proyecto ha supuesto una gran satisfacción. Me ha proporcionado conocimientos sobre nuevas herramientas tecnológicas, un nuevo conocimiento sobre realidad virtual, utilización de marcadores y el lenguaje de programación C#, así como un gran avance en los conocimientos sobre 3d, Maya y Sculptris.

8.2 Líneas futuras

Se ha diseñado el proyecto de forma que sea sencillo implementar nuevas funcionalidades en el mismo. En cuanto al diseño 3d, hay zonas especialmente preparadas para un uso futuro, como la cafetería o la zona de tiendas, que por el momento no tienen más función que la de dar ambiente.

Un posible trabajo futuro es el de añadir tanto escenarios como funcionalidades diferentes que amplíen la experiencia. En concreto, se contemplan dos escenarios previos a coger el avión:

-Preparación en casa: En esta fase el usuario deberá preparar en su casa todo lo necesario para viajar en avión, tanto preparar la mochila como asegurarse de que todos sus documentos oficiales (DNI, pasaporte...) estén en regla, y en caso de no estarlo, deberá comunicárselo a un adulto.

-Viaje en autobús: Esta fase contempla el viaje desde casa hasta el aeropuerto. En ella el usuario deberá interaccionar en la calle con un paso de cebra y semáforo, esperar al autobús, entrar en el mismo, pagar y sentarse en el sitio que le corresponda.

Un gran añadido que vemos para el juego “Safe Trip”, y que seria un punto de unión entre “Safe Trip” y “Sonríe” es la implementación de marcadores y realidad virtual, tal y como ha sido incluido en “Sonríe”.
De la misma forma, en la fase de “Safe Trip” en la que el usuario ya ha pasado por seguridad y debe esperar a que se abran las puertas de embarque, un posible añadido futuro es el de incluir mini juegos a los que se pueda jugar en el tiempo de espera, por supuesto, con el correspondiente cambio en la gestión del tiempo de “Safe Trip”. Uno de los mini juegos añadidos a esta fase podría ser “Sonrie”.

9 Bibliografia

Microsoft XNA Game Studio 4.0

https://msdn.microsoft.com/es-ES/library/Bb200104(v=XNAGameStudio.40).aspx

- Foro de ayuda:

http://stackoverflow.com/

Visual Studio Express 2010

https://www.visualstudio.com/es-es/products/visual-studio-express-vs.aspx

Autodesk Maya 2016

http://www.autodesk.es/products/maya/features/new/list-view

Sculptris

http://pixologic.com/sculptris/

Goblin XNA

https://goblinxna.codeplex.com/

https://channel9.msdn.com/coding4fun/blog/Theres-Goblins-in-my-XNA-No-not-that-kind

Juegos educativos

José Luis Eguia Gómez, Ruth S. Contreras-Espinosa,Lluís Solano-Albajes "Videojuegos: Conceptos, historia y su potencial como herramientas para la educación"
http://www.educacontic.es