Desarrollo y gestión del videojuego The Condemned

Memoria del proyecto
de Ingeniería Técnica en
Informática de Gestión
realizado por
Daniel Barbadillo Dubon
y dirigido por
Òscar Cubillo Alonso

Escola d'Enginyeria

Sabadell, *Junio* de 2012

El abajo firmante **Òscar Cubillo Alonso**,
profesor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el trabajo al cual corresponde la presente memoria

ha sido realizado bajo su dirección

por **Daniel Barbadillo Dubon**

Y para que conste firma la presente.

Sabadell, **Junio** de **2012**

Firmado: **Òscar Cubillo Alonso**

HOJA DE RESUMEN – PROYECTO FIN DE CARRERA DE L'ESCOLA D'ENGINYERIA

Título del proyecto: Desarrollo y gestión del videojuego The Condemned

Autor: Daniel Barbadillo Dubon **Fecha:** Junio 2012

Tutor: Òscar Cubillo Alonso

Titulación: Enginyeria Tècnica d'Informàtica de Gestió

Palabras clave (mínimo 3)

- Català: Videojoc, ActionScript, Flash
- Castellà: Videojuego, ActionScript, Flash
- Anglès: Videogame, AcrionScript, Flash

Resum del projecte (extensión máxima 100 palabras)

• Català: The Condemned es un joc de lluita en dos dimensions desenvolupat en Flash CS4 i ActioScript 3.

El joc consta de quatre pantalles, a cada una d'elles el jugador s'enfronta a un enemic controlat per l'ordinador a traves d'una intel·ligència artificial.

En la creació d'aquest videojoc s'ha passat per totes les fases de desenvolupament: disseny gràfic de personatges i escenaris, programació i control d'errors.

• Castellà: The Condemned es un juego de lucha en dos dimensiones desarrollado en Flash CS4 y ActionScript 3.

El juego consta de cuatro pantallas, en cada una de ellas el jugador se enfrenta a un enemigo controlado por el ordenador a través de una inteligencia artificial.

En la creación de este videojuego se ha pasado por todas las fases de desarrollo: diseño gráfico de personajes y escenarios, programación y control de errores.

• Anglès: The Condemned is a fighting game in two dimensions developed in Flash CS4 and ActionScript 3.

The game consists of four screens, each player faces an enemy controlled by the computer through an artificial intelligence.

In creating this game has gone through all stages of development: graphic design characters and environments, programming and error handling.

Indice

1. Introducción, 1
   1.1. Motivaciones, 1
   1.2. Objetivos, 1
2. Historia de los videojuegos y estado del arte, 3
   2.1. Introducción, 3
   2.2. Factores que han influido en la expansión de los videojuegos, 3
   2.3. Historia de los videojuegos, 4
      2.3.1. Precedentes, 4
      2.3.2. Del primer videojuego hasta nuestros días, 5
      2.3.3. Evolución de las videoconsolas en imágenes, 8
3. Géneros de videojuegos, 11
4. Estudio de viabilidad, 19
   4.1. Introducción, 19
      4.1.1. Tipología y palabras clave, 19
      4.1.2. Descripción, 19
      4.1.3. Objetivos del proyecto, 19
      4.1.4. Partes interesadas (Stakeholders), 20
      4.1.5. Producto y documentación del proyecto, 20
   4.2. Estudio de la situación actual, 20
      4.2.1. Contexto, 20
      4.2.2. Usuarios y/o personal del sistema, 20
      4.2.3. Diagnóstico del sistema, 20
   4.3. Requisitos del sistema, 21
      4.3.1. Requisitos funcionales, 21
      4.3.2. Requisitos no funcionales, 21
      4.3.3. Restricciones del sistema, 21
      4.4.4. Catalogación y priorización de los requisitos, 21
   4.4. Planificación, 22
      4.4.1. Introducción, 22
      4.4.2 WBS (Work Breakdown Structure), 22
         4.4.2.1. Fases y actividades del proyecto, 22
         4.4.2.2. Diagrama WBS, 23
         4.4.2.3 Milestones, 23
      4.4.3 Planificación temporal, 24
         4.4.3.1. Calendario del proyecto, 24
         4.4.3.2. Dependencias, 24
         4.4.3.3. Diagrama de Gantt, 24
   4.5. Evaluación de riesgos, 25
      4.5.1. Lista de riesgos, 26
      4.5.2. Catalogación de riesgos, 26
      4.5.3. Plan de contingencia, 26
   4.6. Alternativas y selección de la solución, 27
      4.6.1. Alternativa 1, 27
      4.6.2. Alternativa 2, 27
      4.6.3 Solución propuesta, 28
5. Diseño, 29
   5.1. Gráficos, 29
   5.2. Tecnología Utilizada, 30
      5.2.1. Historia de Flash y ActionScript, 30
      5.2.2. Cronología de Flash y sus componentes, 33
      5.2.3. Tipos de archivo, 33
      5.2.4. La línea de tiempo, 33

5.2.5. Los MovieClips, 36
5.2.6. Los Listeners, 37
5.2.7. Los timers, 40
5.2.8. Control de colisiones, 43
5.2.9. Los imports, 43
5.2.10. Los sonidos, 44
5.3. Conceptos de programación, 44
5.3.1. Herencia, 45
5.3.2. Patrones de diseño, 46
5.3.3. Factory Method, 46
5.4. Inteligencia artificial, 49
5.4.1 Introducción, 49
5.4.2. Historia de la I.A., 49
5.4.3. Esquema de la I.A en el videojuego The Condemned, 50
6. Pruebas, 53
6.1. Tipos de pruebas, 53
6.2. Pruebas realizadas, 53
6.2.1. Pruebas unitarias, 53
6.2.2. Pruebas de integración, 54
7. Guía del juego, 57
8. Conclusiones, 63
8.1. Objetivos, 63
8.2. Desviaciones sobre la planificación, 63
8.3. Líneas futuras, 64
8.4. Valoración personal, 64
Tipología y palabras clave, 67
Bibliografía, 68
Indice de ilustraciones, 69
Indice de figuras, 70

1. Introducción

Actualmente nos encontramos ante una nueva era dorada de los videojuegos. Los usuarios de este tipo de aplicaciones recreativas han crecido exponencialmente en los últimos años y ya suman millones de jugadores en todo el mundo.

La experiencia individual ha dado paso a una experiencia colectiva gracias a Internet y a los avances de la tecnología, factores que explotan al máximo las compañías de videojuegos para aumentar la calidad y diversidad de sus productos y así lograr captar nuevos perfiles de usuarios.

Enumerando casos de éxito como **Brain Training (Nintento)** o **Farm Ville (Facebook)** podemos observar como se han roto los viejos estereotipos y hoy en día todo tipo de personas son consumidoras habituales de videojuegos, el rango de edad ha aumentado notablemente y el porcentaje de mujeres que los utilizan es cada vez mayor.

Este proyecto se basa en la creación de un videojuego de lucha en dos dimensiones desarrollado en **Adobe Flash CS4 [1]** y **ActionScript 3 [2]**. El juego consta de cuatro pantallas, en cada una de ellas deberemos vencer a un enemigo controlado por el ordenador a través de una serie de algoritmos que forman su inteligencia artificial.

Para la creación de este proyecto se ha pasado por tres fases básicas en el desarrollo de un videojuego: **diseño de personajes**, **programación [3]** y **control de errores**, cada una de las etapas será explicada con detalle en la memoria.

1.1. Motivaciones

Desde niño siempre me han fascinado los videojuegos y he experimentado en primera persona las mejoras tecnológicas que han sufrido a lo largo de los años.

Por otra parte siempre he tenido una inclinación artística que me ha llevado a participar en talleres de dibujo donde creaba mis propios personajes e historias. Uniendo mis dos aficiones principales he decidido desarrollar mi propio videojuego como proyecto final de carrera.

Además durante la carrera me han interesado mucho las técnicas utilizadas en la ingeniería del software [4] para lograr no sólo programar sino hacerlo de una manera ordenada y con el máximo de eficiencia. Así pues, para la programación de este videojuego he seguido una serie de "buenas prácticas" referentes a estilo y estructura, todo el código es fácilmente interpretable y resultará fácil ampliarlo en un futuro.

1.2. Objetivos

- Creación de un videojuego pasando por todas las fases de desarrollo.
- Programación óptima de la aplicación para poder ampliarla fácilmente en el futuro.
- Desarrollo de una inteligencia artificial [5] dinámica.
- Ampliación de conocimientos de Flash y ActionScript 3.

2. Historia de los videojuegos y estado del arte

2.1. Introducción

Es necesario en todo proyecto, independientemente de a que sector pertenezca, hacer una reflexión sobre la evolución de su desarrollo e industria así como del estado actual del mundo al que pertenece.

En este caso, un breve repaso nos permitirá apreciar que aunque la industria de los videojuegos tiene una corta vida comparada a la del cine o a la de la música, ha experimentado una expansión tan importante que en los últimos años ha recaudado más dinero que estos dos sectores juntos.

En este tiempo, los videojuegos han dejado de ser exclusivos de unos pocos para convertirse en el entretenimiento principal de millones de jugadores en todo el mundo.

Para que esta impresionante expansión se haya efectuado, han influido diversos factores, los cuales se explican detalladamente a continuación.

2.2. Factores que han influido en la expansión de los videojuegos

- **El avance y la propagación de la tecnología**: Hace unos años tener un ordenador personal era algo atípico y estaba considerado un bien de lujo, hoy en cambio, lo extraño es encontrar un hogar sin una o varias computadoras. En pocos años hemos creado una nueva necesidad básica y resulta inimaginable encontrar una casa moderna sin ordenador, videoconsola u otros aparatos electrónicos, ya sea con una finalidad recreativa o funcional.

  Por otro lado desde hace unos años vivimos un auge de la tecnología, que crece a un ritmo exponencial, el móvil que tiene un adolescente en el bolsillo tiene mejor procesador que los ordenadores más potentes de hace unos años, estas mejoras tecnológicas permiten satisfacer más necesidades a más perfiles de usuario distintos.

- **Internet:** La red que todo lo une, a día de hoy no somos conscientes de las posibilidades que este gigante de la comunicación nos ofrece. En Internet el usuario se encuentra a un solo "click" de un abanico infinito de lugares donde buscar información, comunicarse o encontrar entretenimiento.

  La red de redes ha logrado "socializar" a los jugadores de todo el mundo con los juegos online, en los que podemos interactuar con jugadores de cualquier parte del planeta. De este modo las barreras físicas se ven superadas y jugar ya no es una experiencia individual.

  Este factor se ha hecho más importante desde que las consolas de última generación también nos brindan la posibilidad de jugar online, multiplicando el número de usuarios que utilizan Internet como principal modalidad de juego.

- **El trabajo creativo:** En unos tiempos en que el mundo del cine, la música y la literatura se encuentran estancados y faltos de ideas, reinventando la rueda una y otra vez y viviendo del pasado, el mundo de los videojuegos sufre una constante renovación.

Mejorando tanto técnica como artísticamente, los videojuegos son cada vez más realistas, divertidos y originales, hasta el punto que los guiones de muchos videojuegos superan con creces al de la mayoría de películas actuales. En este arte aún no ha llegado el día en que podamos mirar hacia atrás con nostalgia y decir "*cualquier tiempo pasado fue mejor*".

- **La eliminación de los estigmas sociales:** Vivimos en una sociedad donde cualquier actitud que destaque de la masa provoca un rechazo por su parte, de esta manera antes el típico aficionado a los videojuegos era considerado un bicho raro y podía ser excluido de determinados círculos.

  Hoy en cambio existe una gran variedad de productos y jugar a videojuegos ya no es algo raro. Podemos decir que hay un juego para cada tipo de persona, solo hay que pensar en productos como **Sing Star (Sony)** o **Brain Training (Nintendo)**, para darnos cuenta de que ya no podemos estereotipar al consumidor de videojuegos.

Gracias a estos y a otros muchos factores el mundo de los videojuegos es tal y como lo conocemos en la actualidad. Pasemos ahora a hacer un pequeño resumen cronológico desde los inicios de este arte, hasta nuestros días.

2.3. Historia de los videojuegos

Pese a que no hay consenso sobre cual fue el primer videojuego y existen distintas teorías, parece que **OXO**, juego creado en **1952** por **Alexander S.Douglas** es considerado como el primero de su género. De todas formas existen diversos precedentes que deben tenerse en cuenta ya que intervinieron de manera crucial al desarrollo de este arte como los conocemos hoy en día.

2.3.1. Precedentes

- En **1947** Thomas T.Goldsmith y Estle Ray desarrollaron un sistema electrónico llamado **Lanzamiento de misiles**. Este sistema estaba basado en pantallas de radar y permitía a los jugadores ajustar la velocidad y la curva del disparo pero los objetivos estaban sobre-impresionados en la pantalla, es decir no existía un movimiento de video, motivo por el cual es considerado como un precedente y no como un videojuego.

- Otro importante precedente es el juego de **ajedrez** desarrollado por **Alan Turing**, informático y uno de los padres de la computación moderna. Entre **1948** y **1952** programó el primer programa de ajedrez en el que una computadora emulaba la forma de jugar de un ser humano. Aunque el juego nunca llegó a ejecutarse puesto que en ese momento no existían computadoras suficientemente potentes varios de sus algoritmos son utilizados por simuladores actuales. Turing estaba convencido de que los juegos consistían un modelo ideal para el estudio de la inteligencia en maquinas y como estas debían conducirse, opinión compartida por muchos expertos de nuestros tiempos.

- El último precedente que se debe comentar es el del videojuego **Nimrod**, desarrollado por la compañía **Ferrari International** y presentado en el Festival de Gran Betraña en **1951**.

El juego debe su nombre al juego de estrategia (se cree de origen chino) llamado Nim. Este juego se suele tratar de manera habitual en la teoría de juegos de la matemáticas y la combinatoria y consiste en un juego para dos jugadores que se turnan para retirar objetos dispuestos en varios montones diferentes y al final, pierde el jugador que acaba retirando el último objeto del tablero.

Ferranti fabricó una computadora que jugaba al Nim contra un ser humano, implantando así una inteligencia artificial. Nimrod utilizaba un panel de luces a modo de display [6] donde se representaban los "montones" y cada luz representaba un objeto. Pese a todo al no tener imágenes en movimiento es considerado un precedente al igual que Lanzamiento de misiles.

2.3.2. Del primer videojuego hasta nuestros días

En **1952**, Alexander Douglas, un estudiante de la Universidad de Cambridge desarrolla **OXO**, el primer videojuego de la historia. Basado en el antiguo juego de el tres en raya, permitía a un humano competir contra un computador.

Fue desarrollado como forma de ilustrar su tesis en el Doctorado sobre la interacción entre el ser humano y un ordenador. Douglas utilizaba como control un dial telefónico y como salida una pantalla de osciloscopio [7].

Este videojuego esta considerado como el primero de la historia por tener una implementación con gráficos como salida, además de ser una de las primeras demostraciones prácticas de inteligencia artificial.

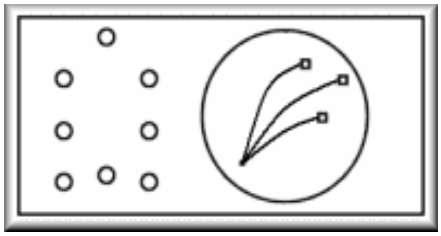

Ilustración 2.1: Lanzamiento de misiles

Ilustración 2.2: OXO

Después de seis años en los que los videojuegos no se veían fuera del ambiente de los talleres de ingeniería y de las universidades, William Higinbotham, un ingeniero norteamericano presentó **Tennis for Two**, un juego de tenis que había desarrollado con la ayuda del ingeniero Robert Dvorak utilizando un programa para el cálculo de trayectorias y un osciloscopio.

El juego consistía en una linea horizontal que representaba el campo de juego y una vertical que simulaba la red, los jugadores podían elegir el ángulo de disparo para golpear la bola. Algunas fuentes señalan a este juego como el primero de la historia en vez de OXO, de lo que no hay duda es de que se trata del primer videojuego desarrollado para dos jugadores.

Cuatro años después, en **1962**, Steve Rusell**,** un estudiante del Instituto Tecnológico de Massachussets, desarrolló **Spacewar**, un videojuego del espacio para dos jugadores en el que cada jugador debía destruir la nave del contrario.

El juego, que utilizaba gráficos vectoriales y ocupaba 9k de memoria, causo sensación en la época. Asimismo la idea de este juego es una de las más copiadas, existiendo versiones para multitud de plataformas.

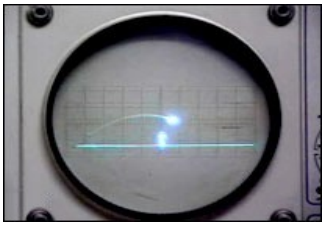

Ilustración 2.3: Tennis for Two

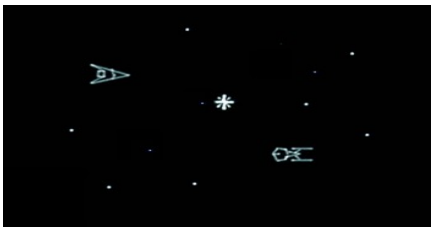

Ilustración 2.4: Spacewar

Años más tarde, en **1966** Ralph Baear desarrolló la idea de videoconsola, se trataba de **Brown Box,** un dispositivo que conectado a un televisor doméstico permitía jugar al usuario. Junto a Bill Harrison y Bill Rusch siguió trabajando y en 1967 finalizaron el primer prototipo que incorporaba una serie de juegos entre los que se encontraba el ping-pong y un juego para dos jugadores en los que se debía acorralar al contrario. Baer y sus colaboradores también diseñaron un rifle que conectado al dispositivo permitía disparar a una serie de objetivos emulando el tiro al plato, este rifle esta considerado el primer **periférico [8]** de la historia.

Esta videoconsola no pasó de ser un prototipo por problemas en las negociaciones entre Baer y distintas compañías y no está considerada una videoconsola como tal porque los juegos no eran de lógica programada y la mayoría eran distribuidos en la misma maquina, siendo solo placas que entrelazaban los circuitos configurando su comportamiento.

Pese a todo, gracias a Brown Box, **Magnavox** lanzo la **Oddyssey**, un modelo mucho más cuidado en plástico que se produjo en serie unos años más tarde.

En **1972**, **Atari**, fundada en Estados Unidos por Nolan Bushnell y Ted Dabney desarrolló una versión de Tennis for Two bajo el nombre de **Pong**, el cual apareció en las primeras **máquinas recreativas**. A su vez en ese año se presentó la consola Odissey, esta si considerada la primera videoconsola de la historia aunque era tan primitiva que los jugadores tenían que anotar sus puntuaciones en un papel al carecer de dispositivo de memoria.

En **1977**, la firma Atari lanzó al mercado el primer sistema de videojuegos en cartucho **(Atari VCS/2600)**, que alcanzó un gran éxito en Estados Unidos y provocó, al mismo tiempo, una primera preocupación sobre los posibles efectos de los videojuegos en la conducta de los niños. El enorme éxito de la máquina de Atari impulsó las ventas de la Odyssey, la consola que le había dado origen, y a finales de 1977 había cerca de 100.000 máquinas arcade solamente en Estados Unidos que generaban más de **250 millones de dólares** anualmente. La industria de los videojuegos había nacido definitivamente.

En **1978 Magnavox** presentó al mercado la **Odissey II** para hacer la competencia a la VCS/2600 de Atari. Poco más tarde Bushnell es despedido y sustituido por Ray Kassar, un ejecutivo sin experiencia en el mundo de los videojuegos que se centró en el mercado de los ordenadores personales, poniendo fin a la era dorada de Atari.

Mientras tanto, en Japón Tomohiro Nishikado desarrolla para **Taito Space Invaders**, un juego que originariamente consistía en disparar contra tanques y aviones pero que cambió la estética por el gran éxito de la película **Star Wars**, adoptando así una forma de batalla espacial.

El juego tuvo una repercusión descomunal. No sólo inició uno de los géneros más importantes de la industria del videojuego (los **Shoot 'em up** o "**matamarcianos**") sino que situó a la industria japonesa en el lugar que le correspondía e impulsó definitivamente la fiebre de los videojuegos a nivel mundial, iniciando la que en la literatura especializada se conoce como "*La Edad dorada de los videojuegos".*

Ilustración 2.5: Pong

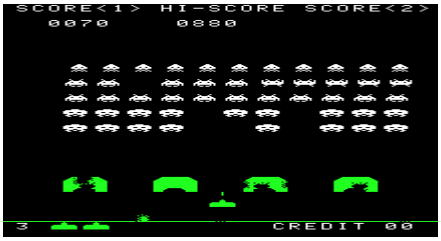

Ilustración 2.6: Space Invaders

En esos años se añadieron al mercado domestico sistemas como **Intellvision (Mattel)**, **Colecovision (Coleco)**, **Atari 5200 (Atari)**, **Commodore 64 (Commodore)** y **Turbografx (NEC)**.

Mientras aparecían estos sistemas domésticos en las maquinas recreativas triunfaron juegos tan famosos como **Pacman (Namco), Battle Zone (Atari)**, **Pole Position (Namco)**, **Tron (Midway)** o **Zaxxon (Sega)**.

El negocio asociado a esta nueva industria alcanzó en poco tiempo grandes cifras. Sin embargo en 1983 comenzó la que los especialistas llaman "L*a crisis del videojuego"*, la cual afectó principalmente a Estados Unidos y Canadá, y que llegó a su fin en 1985.
Mientras tanto en el resto del mundo se produjo una segmentación dentro de la industria de los videojuegos, **Japón** apostó por la fabricación de consolas domésticas con el éxito de la **Famicon**, consola lanzada por **Nintendo** en **1983** y conocida en occidente como **NES** (Nintendo Entertainment System). Por su parte Europa apostaba por los micro-ordenadores como el **Commodore 64** o el **Spectrum**.

Cuando Estados Unidos salió de su crisis siguió el camino de los japoneses y adoptaron la NES como principal sistema de videojuegos. A lo largo de la época fueron apareciendo nuevos sistemas domésticos como la **Master System (Sega)**, el **Amiga (Commodore)** y el **7800 (Atari)**, desarrollándose juegos considerados hoy en día clásicos como el **Tetris** de **Alekséi Pázhitnov** .

A principios de los **90** las videoconsolas sufrieron una revolución tecnológica gracias a la llamada "**generación de 16 bits**" compuesta por la **Mega Drive**, la Super Famicon de Nintendo, en occidente **Super Nintendo**, la **PC Engine** de NEC conocida como **Turbografx** en occidente y la **CPS Changer** de Capcom.

Junto a ellas apareció la **Neo Geo (SNK)** consola que igualaba las prestaciones de una máquina recreativa pero que no llegó de forma masiva al público debido a su elevado precio.

Mientras tanto diversas compañías trabajaban en videojuegos con entornos tridimensionales, sobretodo en el campo de los **PC**, desarrollando juegos como **Doom**, **Alone in the Dark** o **Wolfestein**.

Los videojuegos en 3D ocuparon rápidamente un lugar importante en el mercado gracias a la llamada "**generación de 32 bits**" que empezó en **1994** con la salida al mercado de la **Sega Saturn**, la **Nintendo 64** , la **Atari Jaguar** y la conocida **Play Station** de **Sony**. En cuanto a los PC se crearon aceleradoras 3D que permitían un gran salto en la capacidad gráfica de los videojuegos para esta plataforma.

Por su parte el mercado de consolas portátiles experimentaba un verdadero auge, a la conocida **Game Boy (Nintendo)** se le unieron máquinas como la **Game Gear (Sega)**,

la Lynx **(Atari)** o la **Neo Geo Pocket (SNK)** aunque ninguna de ellas pudo hacer frente a la popular Game Boy, dominadora del mercado.

En esa época eran muy populares en PC los **FPS** como **Quake (id Software)**, **Unreal (Epic Megagames)** o **Half-Life (Valve)** y los **RTS** como **Command & Conquer (Westeood)** o **Starcraft (Blizzard)**. Además el hecho de que se empezaran a cada vez más usuarios tuvieran Internet en sus hogares originó el crecimiento del juego multijugador, siendo la opción predilecta de muchos jugadores, y se creó el genero de los **MMORPG** como **Ultima Online (Origin)**. En el año **1998** apareció en Japón la **Dreamcast** de **Sega**, que dio comienzo a la "**generación de 128 bits**".

En el **2000 Sony** contraatacó con la esperada **PlayStation 2** y **Microsoft** entró en la industria de las consolas con la **Xbox**. Por su parte **Nintendo** lanzó la **Gamecube**, consola que no tuvo demasiado éxito comparada con las otras de su generación, y la portátil **Gameboy Advance** que tuvo mejor acogida.

En **2002** Sega se retiró de la fabricación de consolas al ver que era incapaz de competir con el gigante de **Sony** y se convirtió exclusivamente en desarrolladores de software.

En **2004** se lanzaron dos nuevas consolas portátiles con una avanzada tecnología, la **Nintendo Ds** y la **PlayStation Portable (PSP)**, ambas han tenido gran éxito en todo el mundo.

A finales de **2005** salió al mercado la **Xbox 360**, la primera de la séptima generación de consolas de videojuegos. En el **2006** salieron a su vez la **PlayStation 3** de **Sony** y la **Wii** de **Nintendo**.

2.3.3. Evolución de las videoconsolas en imágenes

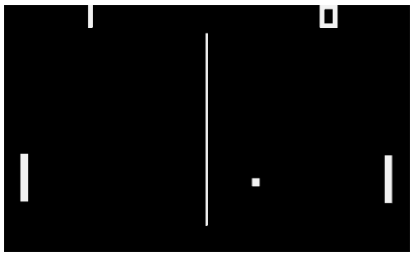

Ilustración 2.7: Primera generación (Tennis for two)

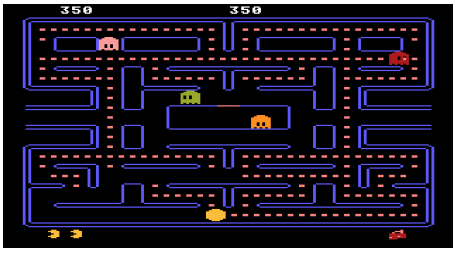

Ilustración 2.8: Segunda generación (Pacman)

Ilustración 2.9: Tercera generación (Mario Bros)

Ilustración 2.10: Cuarta generación (S. Fighter II)

Ilustración 2.11: Quinta generación (Metal Gear)

Ilustración 2.12: Sexta generación (FIFA)

Ilustración 2.13: Séptima generación (Gears of War)

Ilustración 2.14: Octava generación

Viendo la evolución de la industria de los videojuegos en tan pocos años es complicado hacer una previsión de como serán las próximas generaciones de videoconsolas. Por ahora la única videoconsola de octava generación presentada es la consola de **Nintendo Wii U** y su salida al mercado está prevista para este año **2012**.

La principal característica es su nuevo mando, que incorpora una pantalla táctil de 16:9 (6.2 pulgadas) y botones de control tradicionales, incluyendo dos controles (*pads*) analógicos con almohadillas. Esta combinación elimina las barreras tradicionales entre el videojuego, los jugadores y el televisor, mediante la creación de una segunda ventana al mundo ficticio.

De nuevo Nintendo apuesta por un modelo de consola más interactivo y para todos los públicos, apuesta que le ha salido bien tanto en la consola de sobremesa **Wii** como en la recientemente lanzada **3DS**, consola portátil y primera en 3D de la historia.

En el sector de los videojuegos de ordenador, los juegos online, sobretodo los **MMORPG** como **World of Warcraft** han recaudado miles de millones y cada día captan nuevos usuarios, siendo el genero más popular en PC seguido de los juegos de estrategia y los simuladores.

Por otro lado, los nuevos terminales **móviles** son otro foco de mercado para las empresas de videojuegos, que cada vez presentan juegos más atractivos para estos dispositivos. Utilizando el hardware especifico de estos terminales, por ejemplo las pantallas táctiles, se han creado miles de juegos con un genero propio, como el conocido **Angry Birds**, videojuego que ha alcanzado récord de descargas.

Ilustración 2.15: WOW

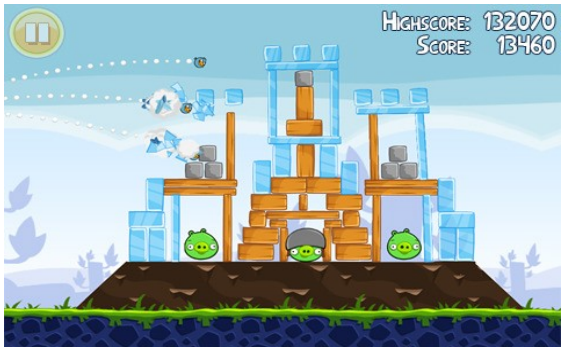

Ilustración 2.16: Angry Birds

3. Géneros de videojuegos

La gran variedad de videojuegos existentes difumina los límites entre un genero y otro y resulta imposible crear una clasificación al gusto de todos. Las nuevas tendencias en videojuegos crean a su vez nuevos géneros al resultar estos cada vez más completos y evolucionados. Por otro lado hay géneros en desuso que han quedado solo para el recuerdo y que mantienen unos pocos nostálgicos. Una breve clasificación nos da una idea de lo amplio que es este arte que cada vez tiene más adeptos entre distintos tipos de personas.

- **Lucha**: los videojuegos de lucha recrean combates entre dos jugadores o entre un jugador y la computadora en un escenario limitado (simulando un ring o una tarima). Existen juegos de lucha sin armas en los que se suele emplear algún tipo de arte marcial y con armas blancas como espadas y cuchillos. Este género apareció a mediados de los años **1980** pero no se hizo popular hasta la llegada de **Street Fighter II**. Con la llegada de los gráficos en tres dimensiones el estilo de juego perduró pero se cambió un poco el modo de jugarlos. El mundo de las tres dimensiones separó al público de este genero, quedando los leales a los juegos en 2D, mucho más rápidos y espectaculares y los adeptos a los juegos en 3D, con gráficos de última generación. Este es el género al que pertenece **The Condemned**, el videojuego en 2D desarrollado en este proyecto.

Ilustración 3.1: Capcom vs Marvel 2

- **Beat'Em Up** : Este género es una variación de los juegos de lucha en los que no solo te enfrentas a un enemigo sino que desplazándote por un escenario ilimitado, que suele recrear una calle u otro tipo de paisaje, van surgiendo enemigos que debes vencer. Normalmente al final de cada escenario hay un enemigo final que se debe vencer para pasar a la siguiente fase. Este género está desaparecido hoy en día ya que las nuevas generaciones de videoconsolas no han apostado por este tipo de juegos, aunque existen algunas variantes.

Ilustración 3.2: Final Fight 3

- **Shooters**: Los videojuegos **de disparos** o ***shooters*** son un género que engloba un amplio número de subgéneros que tienen la característica común de permitir controlar un personaje que, por norma general, dispone de un arma (mayoritariamente de fuego) que puede ser disparada a voluntad

Los subgéneros más importantes son los siguientes:

- **First Person Shooter**: Los videojuegos de **disparos en primera persona** o **FPS**, son aquellos juegos donde únicamente puede verse el extremo del arma del personaje jugador, adoptando este una visión subjetiva. Este género es uno de los más jugados en PC, sobretodo en la modalidad online.

  Una variación de este género son los denominados **juegos con pistolas de luz**, que son FPS en los que el jugador emplea un periférico que simula una pistola u otra arma de fuego como mando, consiguiendo interaccionar de un modo mas realista. Este género tuvo gran acogida sobretodo en máquinas recreativas, que luego fueron importadas a todas las generaciones de consolas y ordenadores.

Ilustración 3.3: Quake 2

- **Third Person Shooter**: Los juegos de **disparos en tercera persona** o **TPS** son juegos donde vemos al personaje que maneja el jugador en tercera persona. Normalmente hay una interacción con el entorno mucho mayor que en los **FPS** al haber gran libertad de movimientos aunque se pierde realismo al no jugar con una vista subjetiva. Dentro de este subgénero cada vez ganan más fuerza los juegos como **Grand Theft Auto** donde las posibilidades que tiene el personaje son casi infinitas, estos juegos son llamados "**sandbox**" aludiendo así que las posibilidades de juego son como las de moldear una caja de arena.

Ilustración 3.4: Grand Theft Auto IV

- **Shoot 'em up**: La diferencia de este genero con los otros es que el jugador se pasa la mayor parte del tiempo disparando. Existen varios tipos y algunos expertos difieren en que juegos pueden considerarse un shoot 'em up y cuales no. Para los puristas solo lo son aquellos juegos

como el clásico **Spacewar** en el que manejamos una nave espacial (o un aeroplano o tanque), pero también hay defensores de incluir en este género juegos como **Metal Slug** en el que el protagonista va a pie y se mueve horizontalmente. En lo que sí que hay consenso es en que este género es únicamente en dos dimensiones, (aunque los gráficos puedes estar dibujados en tres dimensiones).

Ilustración 3.5: Metal Slug

- **Plataformas:** Uno de los géneros que tuvo su auge en las generaciones de **8** y **16 bits** pero que continua hoy en día para los nostálgicos sobretodo en las consolas de **Nintendo**. En estos juegos se deben sortear todo tipo de obstáculos físicos ya sea saltando, agachándose o escalando. Los más famosos son sin duda **Super Mario Bros** de **Nintendo** y **Sonic** de **Sega**, personajes que siguen siendo hoy en día imagen corporativa de estas compañías.

Ilustración 3.6: Super Mario Wii

- **Simuladores**: Este género se caracteriza por llevar un aspecto de la vida real a un videojuego, siendo el realismo el factor diferenciador entre un buen y un mal simulador. Dentro de este genero encontramos los siguientes subgéneros:

    - **Deportivos**: son los que simulan un deporte, entre ellos los más famosos son los de fútbol, baloncesto y tenis aunque podemos encontrar simuladores de casi todos los deportes existentes. Una variedad de estos juegos es la de encarnar el manager de un equipo en vez de al equipo en si, estos juegos se podrían clasificar como de estrategia deportiva.

Ilustración 3.7: Pro evolution soccer 5

- **De carreras**: Dentro de los juegos de conducción existe una ligera linea que separa los simuladores al resto de juegos que podríamos denominar **arcade**, normalmente el realismo del juego suele ser el parámetro a seguir para catalogarlo en un género o en otro, siendo por ejemplo **Gran Turismo** o **Colin McRae Rally** ejemplos de simuladores y **Burnout** o **Mario Kart** ejemplos de arcade.

Ilustración 3.8: Gran Turismo 5

- **De vuelo**: Al igual que con las carreras existen multitud de juegos de pilotaje, algunos se clasifican como simuladores ya que su cometido es el de recrear lo más fielmente posible el manejo de un avión o helicóptero. Estos juegos normalmente no tienen una dinámica de matar enemigos como por ejemplo en **Space Invaders**, imperando siempre la intención de plasmar una situación real. El juego mas famoso de este genero es el **Flight Simulator** de **Microsoft**, que ya va por la décima edición.

Ilustración 3.9: Flight Simulator X

- **De construcción**: Estos juegos nos ofrecen la posibilidad de crear una ciudad a nuestra manera, gestionando un sinfín de opciones. Los más representativos son sin duda los **Sim City**, aunque también existen diversas versiones como el **Theme Hospìtal** o el **Theme Park**, en el que debes crear y controlar un hospital o un parque de atracciones. Este tipo de juegos ha sido siempre mucho más popular en computadoras, no encontrando nunca un lugar en el mercado de las videoconsolas hasta el momento.

Ilustración 3.10: Sim City 4

- **De vida**: Este tipo de juegos tiene como estandarte a **The Sims**, juego que causó furor hace pocos años. En estos juegos debemos gestionar todos los aspectos de la vida del personaje que controlamos, siendo la única finalidad la felicidad, estos juegos se caracterizan por no tener un final ya que no hay hilo argumental que seguir, envejecemos junto a nuestro personaje. Una evolución de este juego es **The Spore**, juego en el que empezando como una bacteria, podemos evolucionar de infinitas maneras según las decisiones tomadas.

Ilustración 3.11: The Sims 3

- **Simulación musical**: Un subgénero relativamente nuevo, gracias a los periféricos para las consolas de nuevas generaciones podemos emular a un cantante, a un guitarrista, batería o incluso un DJ, otra modalidad son los videojuegos de baile en los que utilizando una alfombrilla el juego capta nuestros movimientos. Los juegos más famosos de este género son **Sing Star** y **Guitar Hero**.

Ilustración 3.12: Guitar Hero 5

- **Aventuras**: Dentro de este género encarnamos a un personaje que debe resolver incógnitas y puzzles utilizando diversos objetos que nos vamos encontrando mientras avanza la historia. Existen dos subgéneros a tener en cuenta:

    - **Las aventuras gráficas**: A principios de los años **1990**, el uso del ratón fue incrementando y dio lugar al tipo de aventura tipo "**Point and click**" también llamados aventura gráfica. En estos géneros la historia cumple un papel fundamental y hay una gran interacción con el entorno. Aunque este genero ha perdido mucha popularidad siguen habiendo muchos adeptos a sus clásicos como **Monkey Island** o **Broken Sword**.

Ilustración 3.13: Monkey Island

    - **RPG**, en los juegos de rol, mas conocidos como RPG (**Rol Playing Game**) el jugador encarna un personaje, que debe ir evolucionando y mejorando mientras transcurre una historia. Los clásicos como **Final Fantasy** o **Dungeons & Dragons** estaban basados en turnos, donde primero se atacaba y después se recibía el ataque del contrario quedando obligatoriamente estático. Hoy en día los RPG ya no usan este tipo de combate en su mayoría y están experimentando una época dorada gracias a los **MMORPG (Massive Multiplayer Online RPG)** como **World of Warcraft** o **Guild Wars**, con millones de jugadores en todo el mundo ya forman parte de la cultura popular.

Ilustración 3.14: Guild Wars

- **Estrategia**: En los videojuegos de estrategia el jugador debe poner en practica sus habilidades de planteamiento y pensamiento para conseguir la victoria, los más famosos son los que recrean batallas, tanto de antiguos imperios como **Age of Empires**, de la segunda guerra mundial como **Comand & Conquer** o del futuro como **StarCraft.**

  Actualmente hay una variación de este tipo de juegos, los llamados videojuegos **de estrategia en tiempo real multijugador masivos en línea** o **MMORTS**. Estos juegos son en su mayoría de navegador, es decir no es necesaria instalación alguna y son gratuitos, lo que les ha dado mucha fama en los últimos tiempos.

Ilustración 3.15: Command & Conquer: Red Alert 3

También existen juegos de estrategia en los que se debe encarnar al dirigente de un país como **Yo Presidente** e incluso hay quien considera los managers deportivos como juegos de estrategia deportiva.

- **Educativos**: Un género que esta muy de moda en estos tiempos, gracias al super éxito de ventas **Brain Training** los juegos que prometen mejorar nuestras capacidades mentales están en boca de todos. Se ha diversificado mucho esta idea y encontramos numerosos títulos cada vez más específicos como **Maths Training**, **Profesor Layton**, etc.

Ilustración 3.16: Brain Training

- **De habilidad:** En este genero el jugador demuestra sus reflejos y concentración al completar puzzles o resolver acertijos. El más famoso es sin duda **Tetris**, creado en **1984** es uno de los videojuegos más jugados de todos los tiempos. El crecimiento exponencial de juegos para móviles de última generación a vuelto a posicionar este género entre los más jugados.

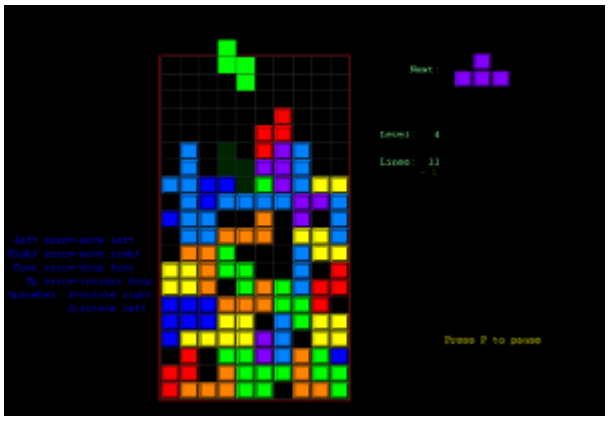

Ilustración 3.17: Tetris

4. Estudio de Viabilidad

4.1. Introducción

4.1.1. Tipología y palabras clave

Tipología: Desarrollo y gestión

Palabras clave: Videojuego, Flash CS4, ActionScript 3, Inteligencia Artificial, Videojuego de lucha en 2 dimensiones.

4.1.2. Descripción

Actualmente el mundo de los videojuegos se encuentra en su mejor momento, los juegos multijugador online y la redes sociales han cambiado el concepto de videojuego, hemos pasado de una experiencia individual a una experiencia colectiva.

Aprovechando el mercado emergente de los videojuegos desarrollados en Flash gracias a Facebook y mi interés por el proceso de creación de un videojuego he decidido desarrollar mi propio juego como proyecto final de carrera.

4.1.3. Objetivos del proyecto

O1. Desarrollo de un videojuego pasando por todas sus fases (diseño, programación, control de errores).

O2. Desarrollo de una inteligencia artificial dinámica.

O3. Creación de personajes y escenarios con una estética atractiva y original.

O4. Programación del juego óptima en cuanto a estructura y estilo.

O5. Gestión correcta del videojuego para que esté libre de errores.

O6. Creación de un sistema de puntuación.

Objetivo	Crítico	Prioritario	Secundario
1	X		
2		X	
3		X	
4		X	
5	X		
6			X

Tabla 1: Criticidad de los objetivos

4.1.4. Partes interesadas (Stakeholders)

Stakeholders		
Nombre	Descripción	Responsabilidad
P	Proyectista	Desarrollar un videojuego digno de un proyecto final de carrera.
D	Director del proyecto	Supervisar el trabajo del alumno
Perfiles de usuario		
J	Jugador	Divertirse

Tabla 2: Stakeholders

4.1.5. Producto y documentación del proyecto

- Se entregará una aplicación informática.

- La aplicación dispondrá de instrucciones para el jugador.

- Se elaborará una memoria del proyecto.

4.2. Estudio de la situación actual

4.2.1. Contexto

Actualmente resulta inimaginable pensar en un videojuego que no disponga de modalidad online. Internet alberga miles de videojuegos de los cuales un amplio porcentaje está desarrollado en Flash.

Por ejemplo, la red social Facebook utiliza esta tecnología en todos sus videojuegos, que son utilizados por millones de jugadores. Hace tan sólo unos meses Farm Ville era el videojuego con mayor número de usuarios.

Pese a que poco antes de morir, el gurúde Apple Steve Jobs predijo la desaparición de Flash en poco tiempo, esta tecnología está lejos de su extinción, es más, el nuevo CEO de Apple anunció recientemente que el Iphone 5 volverá a dar soporte a Flash, con lo que el futuro de esta tecnología está mas que asegurado.

4.2.2. Usuarios y/o personal del sistema

El único usuario del software será el jugador.

Yo seré el encargado de cubrir los perfiles de trabajo de diseño, programación y testeo.

4.2.3. Diagnóstico del sistema

Pese a que existen miles de videojuegos desarrollados en Flash pocos son del género de lucha y la mayoría de estos utilizan “sprites” de videojuegos míticos como Street Fighter

o King of Fighters en vez de diseños originales.

Por su parte el videojuego que propongo está diseñado íntegramente por mí ya que mi interés reside en aprender todas las fases de creación de un videojuego.

4.3. Requisitos del sistema

4.3.1. Requisitos funcionales

RFA1. Interacción entre el jugador y la aplicación correcta

RFA2. Funcionamiento de la creación de combates así como de la gestión de vidas libre de errores

RFA3. Desarrollo de una I.A dinámica

RFA4. Gestión de les puntuaciones de los jugadores

4.3.2. Requisitos no funcionales

RNF1. Lograr una estética original y propia tanto en los personajes como en el estilo de los escenarios

RNF2. Desarrollar la aplicación siguiendo unas reglas de estilo definidas en la ingeniería del software

RNF3. Programación del videojuego óptima siendo su código fácilmente detectado, corregido y ampliado

4.3.3. Restricciones del sistema

Para poder desarrollar la aplicación hay que obtener una licencia de Flash CS4 o CS5.

Para poder ejecutar la aplicación es necesario tener conexión a Internet y el plugin de Adobe Flash instalado en un navegador.

4.4.4. Catalogación y priorización de los requisitos

A continuación se muestra el orden y la preferencia de los requisitos.

Requisitos funcionales

Tipo / Requisito	1	2	3	4
Esencial	X	X	X	
Condicional				X
Opcional				

Tabla 3: Prioridad requisitos funcionales

Requisitos no funcionales:

Tipo / Requisito	1	2	3
Esencial			X
Condicional	X		
Opcional		X	

Tabla 4: Prioridad requisitos no funcionales

Tabla de relaciones entre los Objetivos y los requisitos:

	RF1	RF2	RF3	RF4	RNF1	RNF2	RNF3
O1		X					
O2			X				
O3					X		
O4	X	X				X	X
O5	X	X					
O6				X			

Tabla 5: Relación entre los Objetivos y los requisitos

4.4. Planificación

4.4.1. Introducción

Este documento recoge el Plan del proyecto, es decir, el conjunto de actividades que permite desarrollar, ejecutar y controlar el proyecto.

Además, incluye las tareas y puntos de control, el calendario y la evaluación de riesgos del proyecto.

Se ha utilizado una metodología lineal, denominada también en cascada, para la elaboración de este plan.

4.4.2 WBS (Work Breakdown Structure)

Para poder planificar el trabajo, conocer con exactitud cuáles son las fases del proyecto, con qué concluye cada una de esas fases y la fecha prevista de su finalización se utilizará la herramienta WBS.

Se describirá cada una de las partes del proyecto y detallará cuáles son los puntos de control.

4.4.2.1. Fases y actividades del proyecto

Este proyecto se puede separar en las siguientes fases, cada una de ellas viene

descrita con las actividades a realizar.

Fases	Descripción
Inicio	Fase de iniciación. Incluye las actividades, definición del proyecto, asignación y matriculación
Planificación	Incluye el Estudio de Viabilidad y el Plan de Proyecto
Análisis	Análisis de los requisitos funcionales y no funcionales
Diseño	Incluye el diseño gráfico
Desarrollo	Fase de desarrollo de la aplicación
Pruebas	Fase de pruebas del sistema
Generación de documentos	Fase de redacción de la memoria
Cierre del proyecto	Fase de cierre. El director del proyecto firma la aceptación y cierre del proyecto
Defensa	Defensa del proyecto frente al tribunal

Tabla 6: Fases del proyecto

4.4.2.2. Diagrama WBS

La WBS o Estructura desglosada del trabajo, es una técnica de planificación mediante la cual podemos definir y cuantificar el trabajo a realizar en todo el proyecto.

Para poder visualizar de manera más clara cada una de estas fases se muestra el siguiente esquema. Los nodos intermedios nombran las fases del proyecto y los nodos terminales muestran los objetivos a conseguir.

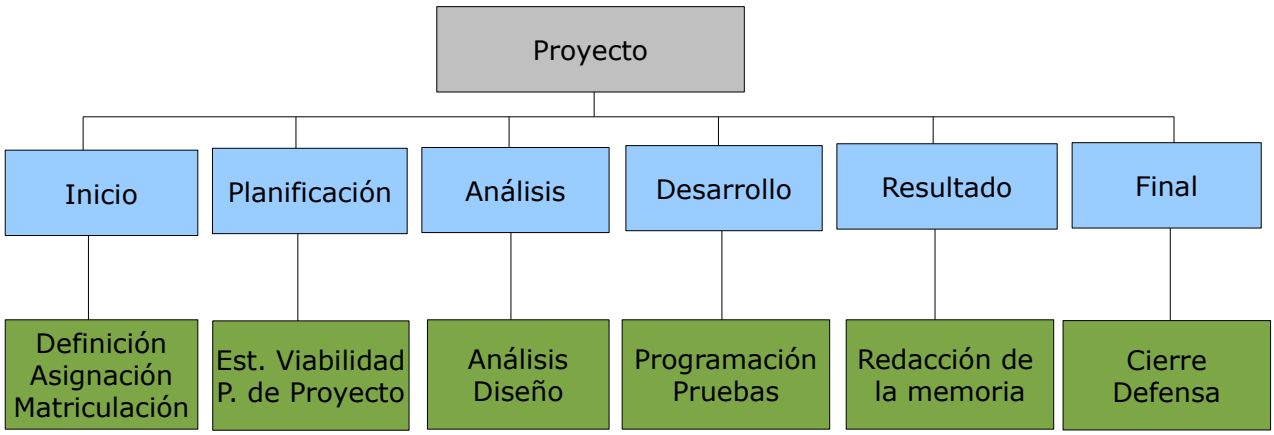

Ilustración 4.1. Work Breakdown Structure

4.4.2.3 Milestones

La siguiente tabla muestra las fechas en las que se hará un punto de control y que, además, tiene que estar finalizada cada fase.

Nombre	Descripción	Fecha
Inicio	Matriculación	15/09/11
Planificación	Aprobación	03/10/11
Análisis	Aprobación	18/10/11
Desarrollo	Aprobación	12/03/11
Cierre	Aceptación	23/04/12
Defensa	Evaluación	05/07/12

Tabla 7: Milestones

4.4.3. Planificación temporal

4.4.3.1. Calendario del proyecto

El proyecto se desarrollará desde Septiembre de 2011 hasta abril de 2012 con una dedicación de 8 horas semanales aproximadamente. El total de horas dedicadas al proyecto será de 250 horas.

- Fecha de inicio: 3 de octubre de 2011
- Fecha de finalización: 23 de abril de 2012
- Planificación: Para la planificación se ha utilizado la herramienta Microsoft Project

4.4.3.2. Dependencias

El proyecto se ha realizado utilizando un modelo lineal, es decir, cada fase no se inicia hasta que la fase anterior no está completada.

4.4.3.3. Diagrama de Gantt

En los siguiente cuadro se puede apreciar cuales son las tareas a desarrollar en cada fase del proyecto, su duración estimada, las fechas en las que se realizarán y de qué tareas dependen.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesora
1	**Proyecto**	**146 días**	**lun 03/10/11**	**lun 23/04/12**	
2	Inicio del proyecto: asignación y matriculación del proyecto	0 días	jue 15/09/11	jue 15/09/11	
3	**Análisis**	**12 días**	**lun 03/10/11**	**mar 18/10/11**	2
4	Toma de requerimientos	5 días	lun 03/10/11	vie 07/10/11	
5	Estudio de la lógica de la progrmación de videojuegos	3 días	lun 10/10/11	mié 12/10/11	4
6	Investigación entrono Flash CS4 y AS3	4 días	jue 13/10/11	mar 18/10/11	5
7	**Diseño**	**27 días**	**mié 19/10/11**	**jue 24/11/11**	6
8	Diseño gráfico de los personajes	7 días	mié 19/10/11	jue 27/10/11	
9	Diseño de los escenarios y demás componentes	7 días	vie 28/10/11	lun 07/11/11	8
10	Creación de sprites de los movimientos	10 días	mar 08/11/11	lun 21/11/11	9
11	Diseño de las pantallas de Inicio, Game Over y Final	3 días	mar 22/11/11	jue 24/11/11	10
12	**Desarrollo**	**57 días**	**vie 25/11/11**	**lun 13/02/12**	7
13	Diseño de las clases	7 días	vie 25/11/11	lun 05/12/11	
14	Programación del movimiento del personaje	10 días	mar 06/12/11	lun 19/12/11	13
15	Desarrollo de la inteligencia artificial	10 días	mar 20/12/11	lun 02/01/12	14
16	Programación de la lógica de combate	15 días	mar 03/01/12	lun 23/01/12	15
17	Programación lógica de la partida	15 días	mar 24/01/12	lun 13/02/12	16
18	**Test y pruebas**	**20 días**	**mar 14/02/12**	**lun 12/03/12**	12
19	Pruebas de la aplicación	10 días	mar 14/02/12	lun 27/02/12	
20	Realización de cambios en la aplicación	10 días	mar 28/02/12	lun 12/03/12	19
21	Redacción de la memoria	30 días	mar 13/03/12	lun 23/04/12	20

Ilustración 4.2. Tabla de tareas

Vista gráfica

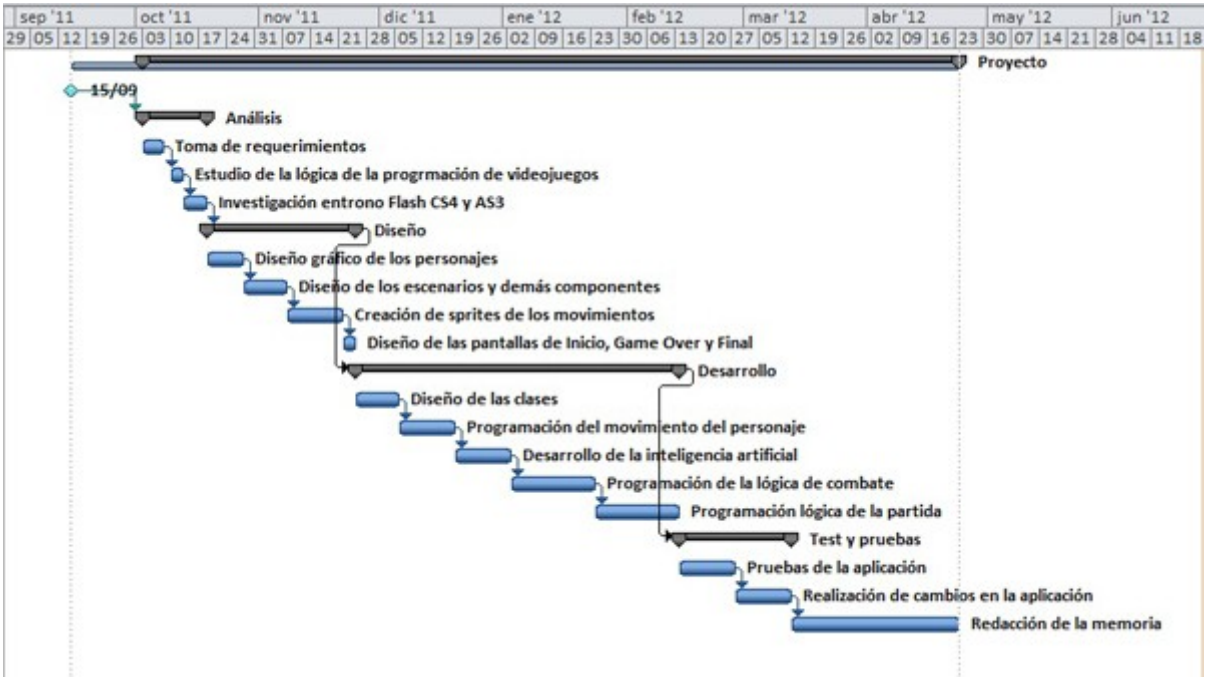

Ilustración 4.3. Representación gráfica

4.5. Evaluación de riesgos

Detectar las vulnerabilidades de un proyecto permite actuar con prevención sobre los posibles fallos, definir estos puntos así como un plan de contingencia para solucionarlos

permite un mayor control del proyecto.

En este caso, se detallarán los riesgos que se pueden encontrar durante el transcurso del trabajo necesario para conseguir los objetivos marcados en los puntos anteriores, sus consecuencias y una solución para cada uno de ellos.

4.5.1. Lista de riesgos

Dentro de este proyecto se pueden encontrar diferentes situaciones que harán que no salga todo según lo planificado. Estos casos son:

- R1. Planificación temporal optimista: Plan del proyecto. No se acaba en la fecha prevista, aumentan los recursos.
- R2. Retraso de alguna tarea necesaria: Plan de proyecto. No se cumplen los objetivos del proyecto.
- R3. Cambios de requisitos: Estudio de viabilidad, análisis. Retraso en la finalización del proyecto, no se cumplen los objetivos del proyecto.
- R4. Tareas de desarrollo inadecuadas: Desarrollo. Retraso en la finalización del proyecto, menos calidad en la aplicación.
- R5. No se hace correctamente la fase de test: Desarrollo, implantación. Falta de calidad, deficiencias en el videojuego, insatisfacción usuarios.
- R6. Abandono del proyecto antes de la finalización: en cualquier fase. Suspenso del proyecto, frustración.

4.5.2. Catalogación de riesgos

Según el impacto que tengan sobre el proyecto, los riesgos definidos anteriormente se pueden catalogar de la siguiente forma:

Riesgo	Probabilidad	Impacto
R1	Media	Crítico
R2	Media	Crítico
R3	Baja	Crítico
R4	Media	Crítico
R5	Media	Medio
R6	Media	Catastrófico

Tabla 8: Tabla de riesgos

4.5.3. Plan de contingencia

Una vez establecidos cuáles son los posibles riesgos que se pueden encontrar en el transcurso del proyecto se trazará un plan que ayude a resolver cada una de las vulnerabilidades.

Riesgo	Solución
R1	Aplazar o descartar alguna funcionalidad secundaria si el proyecto se encuentra lo suficientemente avanzado para la entrega
R2	Aplazar o descartar alguna funcionalidad secundaria
R3	Buscar alternativas
R4	Buscar alternativas
R5	Realizar pruebas según se avanza el desarrollo e ir validando partes de código con el fin de encontrar el mínimo de errores al finalizar la aplicación
R6	Sin solución

Tabla 9: Planes de contingencia

4.6. Alternativas y selección de la solución

4.6.1. Alternativa 1

Desarrollar el videojuego en Java y no en Flash CS4

Problemas

- Sería necesario obtener una licencia de Adobe Photoshop o de un programa similar para diseñar a los personajes y los escenarios.
- Requeriría una mayor inversión de tiempo por mi parte para dominar el lenguaje de programación Java ya que mis conocimientos de esta tecnología son menores que los de Flash.

Mejoras

- El juego sería mas estándar al ser Java un lenguaje multiplataforma
- Nos ahorraríamos el coste de la licencia de Flash

4.6.2. Alternativa 2

Utilizar sprites de algún videojuego antiguo en vez de diseñarlos yo mismo

Problemas

- Dejaría de estar involucrado en todas las fases de la creación de un videojuego con lo que ya no resultaría un proyecto tan personal.
- No tendría la propiedad de los personajes al no ser de mi creación.

Mejoras

- El videojuego sería más sencillo de desarrollar y se podría finalizar en menos tiempo.

4.6.3 Solución propuesta

La implementación del videojuego es mejor desarrollarla en Flash por las siguientes razones:

- Es el lenguaje de programación que más domino.
- Gracias a soluciones de Adobe como Adobe Air resulta sencillo implementar las aplicaciones desarrolladas en Flash en dispositivos móviles.
- Flash presenta numerosas ventajas frente a otros lenguajes de programación para el diseño y programación de videojuegos (control de colisiones, herramientas de dibujo, tratamiento de las fuentes, etc.)

5. Diseño

5.1. Gráficos

Los gráficos son uno de los aspectos mas importantes de un videojuego ya que de ellos depende en gran parte la diversión del jugador. Sin unos gráficos atractivos es muy complicado que un juego triunfe por muy buena jugabilidad o historia que tenga.

El diseño de los personajes y escenarios de The Condemned se ha realizado íntegramente mediante las herramientas de dibujo que incorpora Flash CS4. Pese a que la mayoría de estas herramientas son las típicas de cualquier software de diseño como el lápiz o el bote de pintura, Adobe Flash ofrece numerosas ventajas para la realización de este estilo de dibujos en comparación a otros programas. Por ejemplo en la herramienta lápiz tenemos la posibilidad de redondear o convertir automáticamente los trazos que hagamos.

Como se muestra a continuación, se han separado las partes del cuerpo en capas [9] para facilitar el diseño, de este modo no es necesario dibujar todo el personaje en cada movimiento ya que se pueden aprovechar muchos componentes.

El movimiento se ha creado dibujando al personaje fotograma a fotograma en la línea de tiempo al estilo de las antiguas películas de Disney, una vez dibujado todo el movimiento se consigue la animación al mover la cabeza lectora de la línea tiempo como se explicará mas adelante.

A continuación se puede apreciar en una captura de pantalla el diseño de los movimientos de los personajes.

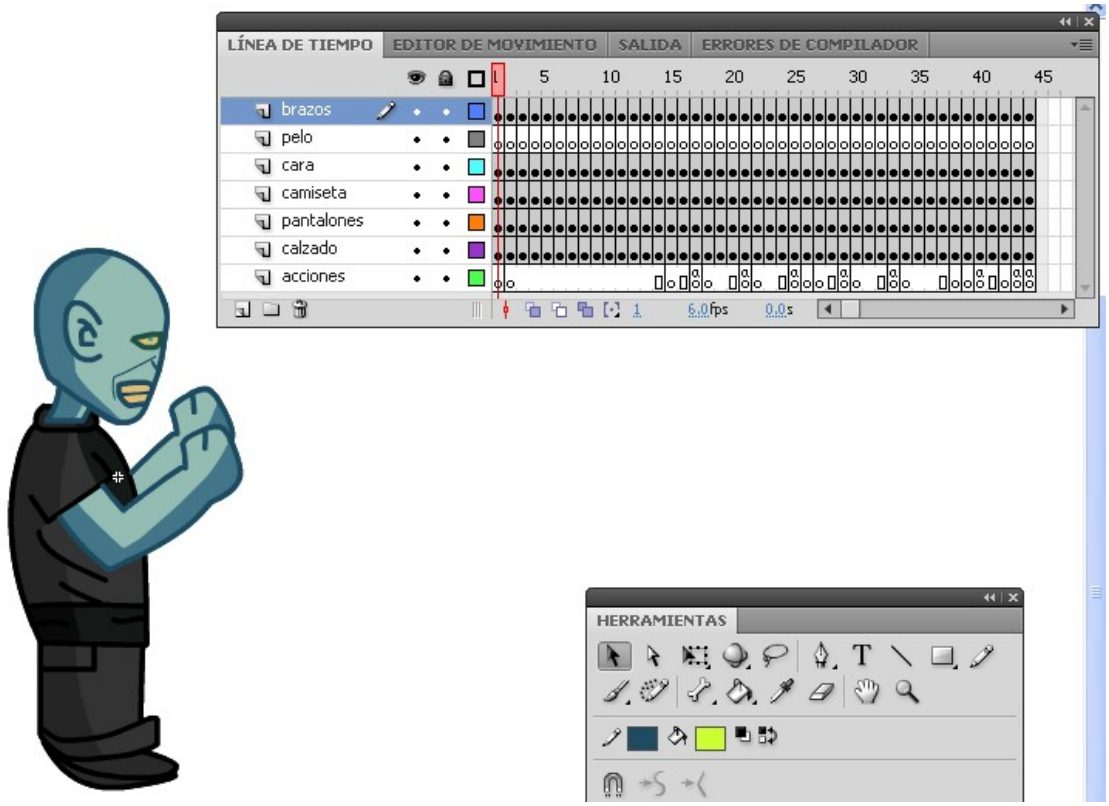

Ilustración 5.1: Capas en el diseño

Además gracias a la herramienta papel de cebolla podemos visualizar mas de un fotograma a la vez, lo que facilita la tarea de dibujar los movimientos.

En la siguiente ilustración se puede ver como seleccionando del fotograma dos al ocho en la herramienta papel de cebolla se puede ver el movimiento de caminar hacia la derecha completo.

También se puede observar que solo las capas correspondientes a los brazos y a las piernas se modifican en este movimiento con lo que se ha reutilizado gran parte del diseño.

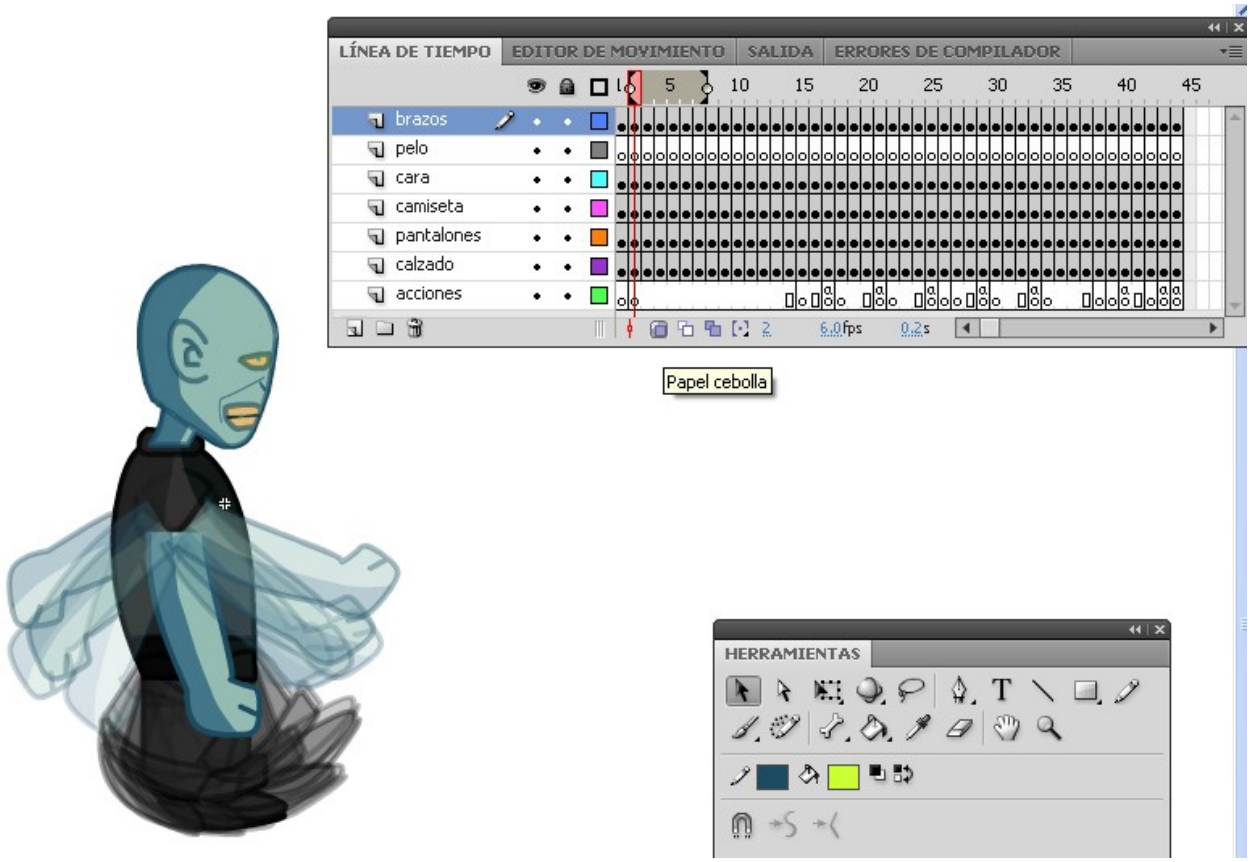

Ilustración 5.2: Herramienta papel de cebolla

5.2. Tecnología Utilizada

Para entender correctamente una aplicación diseñada en **Flash CS4** y programada en **ActionScript 3** es necesario explicar una serie de componentes y características particulares que hacen de esta tecnología diferente del resto de soluciones para crear software. Brevemente se señalaran las mas importantes así como un repaso a la historia de esta tecnología.

5.2.1. Historia de Flash y ActionScript

ActionScript es el lenguaje de programación de la plataforma **Adoble Flash**. En sus

orígenes se desarrolló para que los programadores realizaran sus trabajos de forma más interactiva. Actualmente se encuentra en la versión 3 y en un breve resumen veremos las características de esta versión y de sus predecesoras.

Este lenguaje empezó siendo un mecanismo sencillo de creación de scripts para las primeras versiones de la herramienta de edición **Flash**. Con el tiempo, los programadores empezaron a crear aplicaciones cada vez más complejas y en respuesta a estas necesidades cada nueva versión fue añadiendo características para facilitar este desarrollo.

En los inicios de Internet no había ningún software que mezclara animación con gráficos, reproducir animaciones en la web requería imágenes GIF o Applets de Java, tecnología difícil de manejar. Pero este problema lo solucionó un hombre: **Jonathan Gay**, padre de Flash.

En 1992, Jonathan Gay desarrolló una herramienta llamada **SuperPaint**. Poco más tarde se asoció con Charly Jackson y fundaron Silicon Beach Software. Posteriormente fundó Dark Castle y lanzó SuperPaint II donde adoptó el estándar de dibujo **PostScript**.

Este software evolucionó en un programa de dibujo llamado **Intellidraw**, que compitió en el mercado con **Illustrator** y **Freehand** con modestos resultados.

En 1993, Gay lanzó junto a Charly Jackson **FutureSplash**, un software de dibujo para computadoras con pluma de dibujo. Con la desaparición de las pen computers a mediado del 95 buscan hacerse un hueco en el mercado de algo llamado "Internet". Con poca aceptación en el medio cambian a **FutureSplash Animator**, un programa de animación para Internet basado en linea de tiempo y que pretendía hacer competencia con lo que entonces era el producto estrella de una joven compañía: el **Shockwave** de **Macromedia**.

En Octubre de 1995 tratan de vender su software a Adobe y a Microsoft pero estas no se interesaron por el producto, aunque Microsoft utilizó el software en sus primeras versiones de MSN.

A mediados de 1996 la página de **Disney** utiliza a FutureSplash, el primer gran éxito de la compañía.

En Noviembre de 1996 **Macromedia** decide eliminar la competencia y compra FutureSplash Animator, lanzándolo posteriormente como **Macromedia Flash 1.0**.

1996 es un año importante en la historia de la informática y tecnología ya que se inventó el **DVD**, se creó **Google**, **Flash** e Internet pasó de un millón de usuarios a 10 millones.

En Junio de 1997 aparece la segunda versión de Flash, que incluye una biblioteca de objetos.

En mayo de 1998 aparece la versión 3, con manejo de "**MovieClips**" y generador de archivos exe (projector), en esta versión ya se podían incluir scripts lo que supuso un verdadero avance. Aunque la sintaxis era complicada y poco intuitiva, los que trabajaban con Flash eran capaces de generar contenidos con una interactividad como no se había visto en la Web hasta la fecha.

En Junio de 1999 aparece la versión 4 que incluía manejo de variables y comandos llamados **ActionSCript**, con esta versión el reproductor Flash ya estaba embebido en las instalaciones de IE, Nestscape y AOL. De pronto, el 92% de los ordenadores conectados a Internet eran capaces de reproducir contenido basado en Flash.

En Agosto del 2000 aparece **Macromedia Flash 5**, con un lenguaje basado en ECMA con el nombre de ActionScript (AS), con manejo de XML y SmartClips. De repente, los desarrolladores de Flash se atrevían a hablar de "aplicaciones", "interfaces ricos" y términos como programación orientada a objetos, herencia e incluso patrones de diseño empezaron a ser motivo de discusión en los foros de esta herramienta, algo que sus competidores nunca hubieran creído posible.

Lo que sus competidores ignoraban era que ni siquiera el propio Flash estaba seguro de cuál sería su futuro. Si bien era cierto que había un gran abanico de desarrollos que Flash se sentía capaz de solventar con rapidez y elegancia, la realidad es que no disponía de un arsenal de herramientas que le permitiera cumplir con las expectativas, y ni siquiera las salidas de las versiones **5** y **MX** lograron solventar este problema.

Pero en Septiembre de 2003, Macromedia lanzó la release de Flash que marcó un punto de inflexión en su historia: **Macromedia Flash MX 2004**. De repente, Flash tenía un lenguaje orientado a objetos de verdad, un reproductor potente y eficiente y una serie de herramientas que le permitían cumplir con las expectativas creadas. Ahora sí se podía hablar de patrones de diseño, de extreme programming aplicado al desarrollo Flash, de UML y de otros tantos aspectos que solo estaban a disposición de programadores de otros lenguajes como Java.

El 5 de Mayo de 2005 **Adobe** absorbe a Macromedia y con ella a Flash, meses más tarde, en septiembre de ese año aparece la versión **Flash 8**, con un nuevo motor de renderizado de fuentes y unos nuevos códecs de video que abrieron muchas puertas a la implantación de la plataforma, además, unos meses después del lanzamiento de Flash 8, se liberó Flash Lite 2, de manera que ya no sólo se podía desarrollar aplicaciones para escritorio en AS2 sino que también se podían seguir las mismas metodologías para desarrollar aplicaciones para móviles.

En Abril de 2007 sale la versión **Adobe Flash CS3** que presenta una mayor integración con Illustrator y FireWorks y con **ActionScript 3,** esta versión del lenguaje ya no es un mero recubrimiento estilístico de AS1 como AS2 sino que es un lenguaje que compila un nuevo bytecode. AS3 proporciona una significativa mejora de rendimiento y sobre todo que facilita un modelo de desarrollo mucho más robusto, con revisiones en tiempo de compilación y de ejecución, un sistema de gestión de eventos basado en el estándar DOM, E4X y clases gráficas racionalizadas.

En Octubre de 2008 aparece **Adobe Flash CS4**, que contiene cinemática inversa, manejo básico de 3D, animación de propiedades de objetos y mejoras en ActionScript 3.

En Abril de 2010 aparece la última versión hasta la fecha, **Adobe Flash CS5**, con mejoras para creación de animaciones y funcionalidades para convertir juegos desarrollados en Flash en aplicaciones aptas para iPhone.

5.2.2. Cronología de Flash y sus componentes

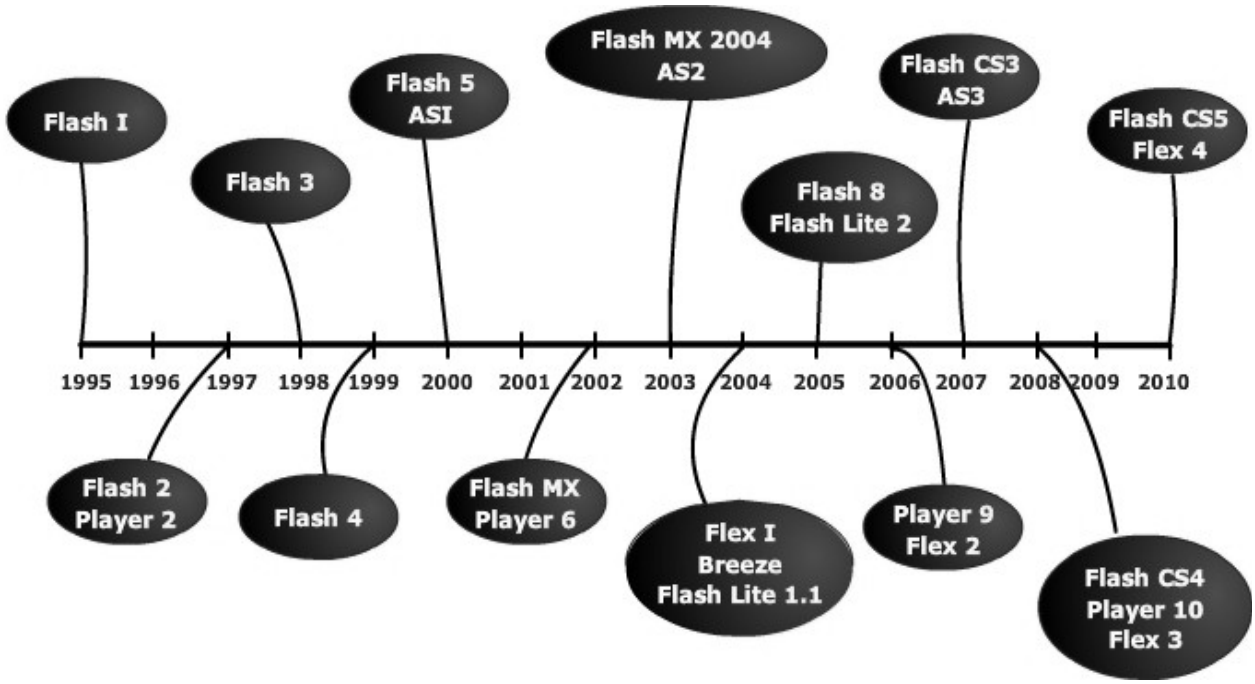

Ilustración 5.3: Cronología de Flash

5.2.3. Tipos de archivo

En Flash existen diferentes tipos de archivos, a continuación se explican sus extensiones así como su función.

- .fla: Es el archivo editable que forma el proyecto sería el equivalente al .psd de Photoshop.
- .swf: Es el archivo compilado preparado para embeber en una página web o visualizar a través de un reproductor.
- .as: En este tipo de archivo se escribe código ActionScript.

5.2.4. La línea de tiempo

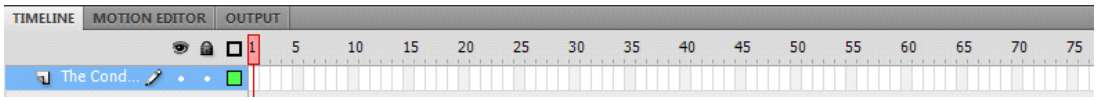

Imagen 5.4 Linea de tiempo

La función de la línea de tiempo es organizar y controlar el contenido de una película a través del tiempo en fotogramas. Los componentes principales de la Línea del tiempo son:

- **Los fotogramas:** forman la película, fotograma a fotograma se crea la animación que queramos realizar, ya sea destinada para web, un videojuego o cualquier otro tipo de aplicación.

- **Las capas:** Igual que otros programas como por ejemplo Adobe Photoshop en Flash CS4 se pueden separar por capas distintos elementos, siendo mucho más sencillo organizar la animación. Las capas de una película aparecen en una columna situada a la izquierda de la Línea de tiempo y podemos asignar un nombre a cada una de ellas.

- **La cabeza lectora:** indica el fotograma actual que se muestra en el escenario. En la parte inferior de la Línea de tiempo tenemos información donde se indica el fotograma actual, la velocidad de reproducción y el tiempo transcurrido de película.

En este videojuego la línea de tiempo principal se encuentra libre de código ya que aunque tenemos esa opción, resulta mucho más ordenado instanciar una clase al escenario principal, así como a cualquier MovieClip que creemos. Un ejemplo de utilización de la línea de tiempo en **The Condemned** es el siguiente:

Los **MovieClips** que forman las barras de vida de los luchadores tienen diversos fotogramas, en el fotograma uno la barra está completamente amarilla, cada vez que reciben un golpe la cabeza lectora se desplaza para mostrar un fotograma en el que la barra tienes menos vida.

En las siguientes imágenes podemos ver gráficamente el ejemplo anterior, en la primera imagen la cabeza lectora se encuentra en el fotograma número uno, con lo que la vida del personaje esta al 100%. En la segunda imagen el cabezal se desplaza al fotograma número 2 después de haber recibido un golpe y en la tercera y última imagen de la serie el cabezal está en el fotograma 12 ya que el personaje ha recibido once golpes y ha perdido el combate.

También podemos apreciar en esta serie de imágenes como tenemos dos capas en este MovieClip, la **Capa 1** tiene una imagen distinta en cada fotograma ya que es la encargada de realizar la animación que acabamos de explicar. Por su parte, la **Capa 2** mantiene la misma imagen durante todo el MovieClip porqué contiene el dibujo del personaje y su nombre.

Imagen 5.5 Ejemplo línea de tiempo estado 1

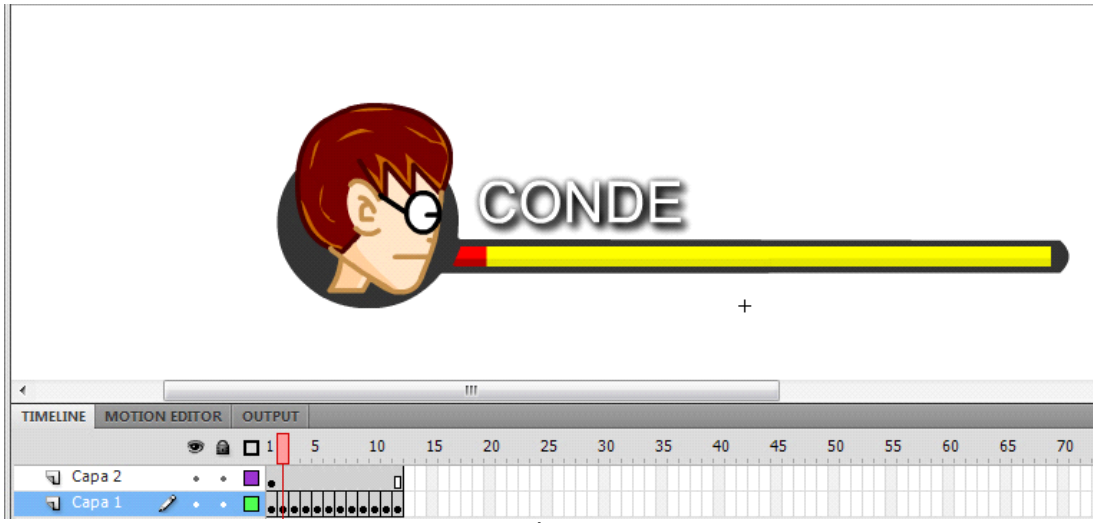

Imagen 5.6 Ejemplo línea de tiempo estado 2

Imagen 5.7 Ejemplo línea de tiempo estado 3

5.2.5. Los MovieClips

Los **MovieClips**, o como los llamaremos desde ahora para abreviar **MC** son uno los componentes más importantes en el diseño y desarrollo de aplicaciones con Flash.

Los MC son objetos para Action script pero se tratan como símbolos [10] durante el manejo de la película. Esto significa que pese a que tienen líneas temporales propias y pueden actuar independientemente de los fotogramas de la película principal, cuando están integrados en un escenario únicamente se muestra como un símbolo, sin línea de tiempo aparente.

Esto hace que al realizar animaciones y programas muy complejos podamos trabajar con orden ya que de otro modo nos encontraríamos con un número de capas y fotogramas totalmente imposible de manejar.

Cuando se crea un MC se le asigna un nombre único para la **librería de objetos** y un nombre de **instancia**, de este modo asociamos el objeto con una **clase** y con unos métodos y atributos propios.

En **The Condemned** hay diversos MC, por ejemplo el personaje principal **Conde** es un MC de la clase **avatar**. Cada enemigo tiene un nombre de instancia propio, con lo que cada uno posee su propia clase aunque todas deriven de la clase madre **Enemigo**. Esto se debe a que AS3 hace una diferenciación entre los nombres de clase y los de instancia, a diferencia de otros lenguajes de programación donde podemos asignar la misma clase a dos objetos distintos en AS3 el nombre de instancia debe ser único, ya que es el nombre que se asocia para poder mostrarlo en el escenario.

Otros elementos como los **escenarios de lucha**, el **reloj**, las **barras de vida** de los contrincantes, los **símbolos de victoria** y los de **vida**, también son MC, cada uno con su clase asignada, sus atributos y métodos correspondientes. De este modo controlamos todos los elementos del escenario, haciendo que aparezcan, desaparezcan o realicen animaciones cuando nos convenga.

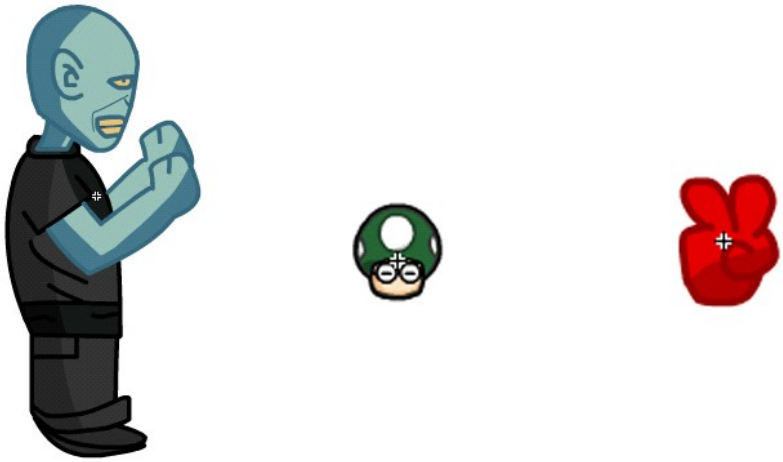

Imagen 5.8 Distintos MC que aparecen en el videojuego

Para añadir al escenario un MC debemos llamar a la función **addChild**(nombre de MC) y para eliminarlo debemos llamar a la función **removeChild**(nombre de MC).

5.2.6. Los listeners

Para la programación orientada a eventos de as3 es clave la figura de los **listeners**, un listener es un objeto creado dinámicamente por código utilizado para detectar eventos.

Por ejemplo podemos detectar si el jugador **ha presionado una tecla** del teclado o a **movido el mouse**. De esta forma los listeners son clave para lograr la interacción entre el usuario y la aplicación además de optimizar recursos, ya que podemos controlar que los eventos no se ejecuten hasta que no sea necesario, además de poder eliminarlos cuando ya no los necesitemos.

Por ejemplo si queremos detectar los eventos del teclado (si se presiona una tecla) lo asignaremos a la clase **Key**.

En esta aplicación tenemos un listener de la clase key para controlar que tecla esta presionando el jugador, si por ejemplo presiona la tecla izquierda el listener llamará a una función que se encargará de desplazar al MC horizontalmente además de mover la cabeza lectora en su línea de tiempo para realizar la animación de desplazarse hacia la izquierda.

De este modo los movimientos del jugador están sujetos a un listener, cuando termina un round y queremos controlar que el usuario no pueda seguir moviendo al personaje simplemente eliminamos el listener de teclado, así conseguimos que no se ejecuten los métodos de movimiento.

El código de este ejemplo es el siguiente:

```
stage.addEventListener(KeyboardEvent.KEY_DOWN,presionaTecla);

stage.addEventListener(KeyboardEvent.KEY_UP,liberaTecla);


        function presionaTecla(e:KeyboardEvent) {
            switch (e.keyCode) {
                case 37 :
                    pressIzquierda=true;
                    break;
                case 38 :
                    pressArriba=true;
                    break;
                case 39 :
                    pressDerecha=true;
```

```
                    break;
                case 40 :
                    pressAbajo=true;
                    break;
                case 49 :
                    pressPuñetazo=true;
                    break;
                case 50 :
                    pressPatada=true;
                    break;
                case 51 :
                    pressBloqueo=true;
                    break;
                case 52 :
                    pressKame=true;
                    break;
            }
        }
```

Como podemos observar inicialmente creamos un listener del tipo teclado **KeyboardEvent.KEY_DOWN** y lo asociamos al método presionaTecla, en este método únicamente tenemos un switch donde según que tecla presionemos (el número que acompaña a cada case es el código ASCII de una tecla del teclado) pondremos a true un determinado movimiento que se realizará acto seguido.

Del mismo modo funciona el método **liberaTecla**, con la particularidad de que al estar asociado a un listener **KeyboardEvent.KEY_UP** lo que controlamos es cuando se deja de presionar una tecla para así dejar de realizar los movimientos asociados a dichas teclas.

A continuación se muestran los sprites que forman todos los movimientos del personaje principal.

Ilustración 5.9 Movimiento de Conde a la derecha

Ilustración 5.10 Movimiento de Conde a la izquierda

Ilustración 5.11 Movimiento de Conde saltando

Ilustración 5.12 Movimiento de Conde pegando puñetazo

Ilustración 5.13 Movimiento de Conde pegando patada

Ilustración 5.14 Movimiento de Conde bloqueando

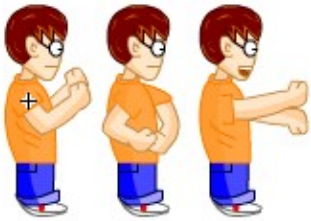

Ilustración 5.15 Movimiento de Conde atacando a distancia

5.2.7. Los timers

La clase **Timer** es una novedad de AS3, esta clase nos permite crear un objeto que cada cierto tiempo llama a una función. Este objeto no esta ligado a la velocidad de la película, sino que lo hace definiendo un intervalo de tiempo. De esta manera podemos indicar que un objeto se mueva cada 5 milésimas, 5 segundos o 5 minutos.

A un objeto Timer, junto con la frecuencia, se le puede indicar cuantas veces se debe ejecutar (por ejemplo, llamar a una función 5 veces cada medio segundo)

Además los timers cuentan con métodos (***start***, ***stop*** y ***reset)*** para controlarlos y con eventos que indican cada vez que se dispara la función o cuándo se ha completado un ciclo de llamadas.

En este videojuego los movimientos de los enemigos están sujetos a un timer, también esta sujeto a un timer el movimiento horizontal de los "kames" [12], es decir de los ataques a distancia.

A continuación explicamos toda la lógica de un kame, desde la creación del timer hasta la eliminación del MC kame.

```
gameTimer=new Timer(60);

gameTimer.addEventListener(TimerEvent.TIMER, mueveAvatar);

gameTimer.addEventListener(TimerEvent.TIMER, combat);

gameTimer.start();
```

Primero creamos un objeto de la clase **Timer** llamado **gameTimer**, y le asignamos que se ejecute cada 60 milésimas de segundo.

Este objeto tiene asignadas dos funciones, una llamada **mueveAvatar**, encargada de

iniciar y parar los movimientos del personaje principal y otra llamada **combat** que alberga toda la lógica del combate.

Para iniciar el timer debemos siempre inicializarlo con un gameTimer.start();

```
function InstanciarKame() {
                    gameTimer.addEventListener(TimerEvent.TIMER,moverKame);
                    kame=new Kame  ;
                    stage.addChild(kame);
                    kame.scaleY=0.5;
                    kame.y=avatar.y;
                    kameEnEscenario=true;
                    kameLibre=false;
                        if (escalaMC==0.5) {
                        kame.x=avatar.x+100;
                        kame.scaleX=0.5;
                    }
                    if (escalaMC==-0.5) {
                        kame.x=avatar.x-100;
                        kame.scaleX=-0.5;
                    }
                }
```

Esta function es la encargada de crear el objeto **kame**, así como de añadir un listener para llamar a otra función llamada **moverKame**.

Una vez creado el objeto kame y de añadirlo al escenario, lo posicionamos correctamente comprobando la escala del personaje que realiza el ataque, esto lo hacemos gracias a la característica **scaleX**, si esta es igual a 0,5, el ataque se realizará hacia la derecha y si es igual a -0,5 se realizará hacia la izquierda.

De este modo se controla en el juego todo el tiempo hacia que lado deben mirar los contrincantes, así si un combatiente salta por encima de otro automáticamente los dos cambian su orientación y vuelven a posicionarse frente a frente.

```
function moverKame(event:TimerEvent) {

                    if (kameEnEscenario==true) {
                        if (kame.x>LIMITE_DERECHA) {
                            eliminarKame();
```

```
                    }
                    if (kame.x<LIMITE_IZQUIERDA) {
                        eliminarKame();
                    }
                    if (escalaMC==0.5) {
                        kame.x+=velocidad;
                    }
                    if (escalaMC==-0.5) {
                        kame.x-=velocidad;
                    }
                    if (kame.hitTestObject(malo)) {
                        puntuacionAvatar=puntuacionAvatar+500
                        nivelBarraEnemigo++;
                        barraMalo.gotoAndStop(nivelBarraEnemigo);
                        avatar.gotoAndStop(1);
                        malo.gotoAndStop(41);
                        eliminarKame();
                    }
                }
            }
```

En esta función controlamos que si el kame sobrepasa los limites del escenario este desaparezca llamando a la función **eliminarKame();** controlamos nuevamente la orientación del personaje para que el movimiento del kame sea acorde a esta. Por último detectamos si el enemigo recibe el ataque mediante el método **hitTestObject** (que explicaremos a continuación) y si esto sucede realizamos una serie de acciones lógicas en el juego (aumento de puntuación, descenso en la barra de vida del enemigo, inicio de animación en la que el enemigo sufre daño y eliminación del MC kame).

```
function eliminarKame() {
gameTimer.removeEventListener(TimerEvent.TIMER, moverKame);
                    stage.removeChild(kame);
                    kameLibre=true;
                    kameEnEscenario=false;
                    }
```

En esta última función eliminamos tanto el listener moverKame como el propio MC kame,

de esto modo como hemos indicado anteriormente se liberan recursos y se puede lograr una programación óptima.

5.2.8. Control de colisiones

En un videojuego es fundamental el control de colisiones tanto entre los contrincantes como entre los personajes y el entorno para poder ofrecer una experiencia realista al jugador.

No tendría sentido por ejemplo que encontrándose en puntas contrarias del escenario, el jugador lanzara un puñetazo y el enemigo respondiera con la animación de recibir un golpe y su barra de vida disminuyera.

En **AS3** la función **hitTestObject(obj:DisplayObject)** evalúa un objeto para comprobar si choca con otro objeto (obj). Por ejemplo para controlar si el jugador y un enemigo chocan escribimos:

```
Avatar.hitTestObject(enemigo)
```

Cuando el jugador y el enemigo colisionan se encuentran en rango, esto significa que pueden golpearse, también controlamos así que los personajes no puedan traspasarse unos a otros, cuando se detecta una colisión entre dos **MC** impedimos que puedan seguir avanzando.

Para que los personajes no puedan pasar los límites del escenario únicamente se debe controlar su coordenada x, que define su posición horizontal. Se han asignado dos variables numéricas llamadas **limiteIzquierdo** y **limiteDerecho**, la variable limiteIzquierdo tiene valor 0 y a la variable limiteDerecho se le ha asignado un valor de 800px.

De este modo, cuando uno de los personajes tiene una posición menor que el limiteIzquierdo o mayor que el limiteDerecho evitamos que siga avanzando.

5.2.9. Los Imports

Si deseamos utlizar una clase que está dentro de un paquete en AS3, debemos importar el paquete o la clase específica, esta es una de las diferencias con respecto a ActionScript 2, donde la importación de clases era opcional.

Por ejemplo pensemos en importar la clase timer, esta clase reside en un paquete denomidado utils, hay que utilizar una de las siguientes sentencias de importación antes de utilizar la clase Timer:

```
import utils.*;
```

o

```
import utils.Timer;
```

En general, las sentencias import deben ser lo más específicas posible. Si se pretende utilizar la clase Timer desde el paquete utils, hay que importar únicamente la clase Timer, no todo el paquete al que pertenece. La importación de paquetes completos puede producir conflictos de nombre inesperados.

Tras importar correctamente la clase o el paquete, se puede utilizar el nombre completo de la clase (utils.Timer) o simplemente el nombre de la clase (Timer).

Los nombres completos son útiles cuando hay ambigüedad en el código a causa de clases, métodos o propiedades con nombres idénticos, pero pueden ser difíciles de administrar si se usan para todos los identificadores. Por ejemplo, la utilización del nombre completo produce código demasiado extenso al crear una instancia de la clase Timer:

```
var myTimer:utils.Timer = new utils.Timer();
```

En vez de esto resulta mucho más sencillo y provoca muchos menos errores declarar las instancias de esta manera:

```
var myTimer:Timer = new Timer();
```

En The Condemned se ha utilizado esta opción para declarar todas las instancias ya que todo el código está estructurado para minimizar los errores y problemas de interpretación.

5.2.10. Los sonidos

Dentro de las aplicaciones multimedia, la utilización de medios como el sonido es de vital importancia, ya que brinda un gran atractivo a la aplicación, además, permite ir más allá del texto y las imágenes al momento de presentar información.

En The Condemend hay dos tipos de sonidos, la música de fondo que acompaña cada combate y los sonidos que se reproducen cuando uno de los contrincantes ataca o recibe un golpe. Para poder diferenciar estos dos tipos de sonidos se han declarado dos SoundChannel distintos, de este modo se pueden reproducir a la vez.

En Flash se pueden utilizar sonidos dentro de la línea de tiempo o de forma dinámica. Todos los sonidos que se utilicen en la línea de tiempo deben también estar en la biblioteca, mientras los que se trabajan de forma dinámica pueden estar en la biblioteca o cargarse de archivos externos.

En este videojuego se ha optado por la utilización de sonidos dinámicos externos, ya que es la forma más eficiente de trabajar, debido que disminuye el tamaño del swf final y así permite tener menor transferencia cuando subimos la aplicación a Internet. También permite modificar los sonidos sin necesidad de compilar la aplicación, los sonidos externos deben estar codificados en mp3.

5.3. Conceptos de programación

The Condemned es un videojuego que se ha desarrollado siguiendo las buenas prácticas que recomienda la **ingeniería del software**. Cualquier programa creado puede realizarse de muchas maneras distintas y ofrecer a simple vista el mismo resultado, aunque no todas sean igual de eficientes.

Uno de los puntos importantes es la **optimización de recursos**, en el desarrollo de aplicaciones se debe controlar siempre los recursos que consume nuestra máquina para así obtener mejores resultados. Este punto se ha dejado un poco de lado en la actualidad ya que las máquinas cada vez son mejores y acabar con sus recursos parece algo imposible.

De todas formas nunca se debe hacer un programa más pesado de lo necesario y esto se ha cuidado como ya se ha explicado antes mediante la gestión correcta de eventos y listeners.

Por otro lado una aplicación se encuentra bien diseñada si es **fácil de interpretar**, es decir si una persona (con conocimientos suficientes de programación) ajena a su desarrollo puede identificar rápidamente la lógica del programa.

Por último el concepto que considero clave para cualquier diseño de software, **la modularidad**, un programa debe ser fácil de ampliar y manipular. Cuanto más separadas se encuentren las partes de un programa mas fácil de ampliarlo o modificarlo resultará.

Para lograr un diseño óptimo como el que hemos descrito, existen dos conceptos indispensables: la **herencia** y los **patrones de diseño**, además de explicar estos dos conceptos se explicarán las técnicas que se han utilizado para lograr un estilo estandar y entendible.

5.3.1. Herencia

La herencia es un concepto clave en la programación orientada a objetos, se utiliza para crear clases a través de otras clases existentes para que compartan funciones y atributos.

Fuentes como la Wikipedia definen la herencia como una relación entre una clase general y otra clase más especifica. Por ejemplo si declaramos una clase párrafo derivada de una clase texto, todos los métodos y variables asociadas con la clase texto son automáticamente heredados por la subclase párrafo.

En **The Condemned** la herencia tiene un papel fundamental ya que gracias a ella se ha podido reutilizar mucho código, sobretodo en el caso de los enemigos, pues cada uno de ellos tiene una clase propia pero todas heredan de la clase principal "**Enemigo**". Del mismo modo los MC de las barras de vida, de los rounds ganados y de los escenarios también utilizan esta estructura.

Utilizando esta estructura nos ahorramos reescribir gran cantidad de código ya que tanto los métodos como los atributos están definidos en las clases generales y en las clases que heredan únicamente debemos referenciarlos.

Si por ejemplo queremos crear un nuevo enemigo ya no será necesario escribir todas sus funciones, sólamente será necesario definir su clase como hija de Enemigo y tendremos acceso a todas las propiedades de la clase madre.

5.3.2. Patrones de diseño

Cuando nos iniciamos en la programación, independientemente del lenguaje que utilicemos, el término **patrones de diseño** nos suena algo extraño y distante. Es cuando empezamos a profundizar en la algoritmia cuando acabamos empleándolos para poder solucionar algún problema de diseño que se nos haya planteado.

¿Qué son los patrones de diseño? Son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Los patrones de diseño aportan **flexibilidad** y **modularidad** a los diseños y son **reutilizables**, lo que significa que son aplicables a diferentes problemas de diseño en distintas circunstancias.

Existen tres tipos de patrones de diseño:

- **Patrones de Creación:** Resuelven problemas sobre inicialización y configuración de objetos.

- **Patrones Estructurales:** Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan para formar estructuras más grandes.

- **Patrones de Comportamiento:** Más que describir objetos o clases, describen la comunicación entre ellos.

En este videojuego se ha utilizado el patrón de diseño **Factory Method**, que pasamos a explicar a continuación.

5.3.3. Factory Method

El patrón de diseño **Factory Method** nos permite crear objetos de distintos subtipos según los parámetros que le demos al método creador, el cual pertenece a una clase fábrica (encargada de crear los objetos).

Este patrón resulta útil cuando es necesario acceder a algún recurso para la creación y configuración de un objeto o cuando no conocemos hasta el momento preciso de la instanciación qué tipo concreto de objeto se va a instanciar.

Un ejemplo de implementación de este patrón en el videojuego es el relacionado con los enemigos. Como hemos explicado anteriormente, tenemos la clase madre **Enemigo** que posee diferentes atributos y métodos. Tenemos cuatro enemigos distintos, llamados **Cell**, **Maul**, **Condemor** y **Sauron**, cada uno de estos enemigos tiene su propia clase homónima que hereda métodos y atributos de la clase madre, además de poseer los suyos propios.

¿Dónde entra en juego la clase Factory Method? Pues bien, el videojuego tiene cuatro pantallas, en cada una de ellas deberemos enfrentarnos a un enemigo distinto, así pues nos surge la siguiente duda ¿como controlamos que enemigo asignar en cada pantalla?

La respuesta es simple si utilizamos una clase Factoria, en este caso llamada **Enemigo_Factory**, donde pondremos el siguiente código:

```
public class EnemigoFactory {
public var enemigoDarthMaul:EnemigoDarthMaul;
    {
            public var enemigoCell:EnemigoCell;
            public var enemigoSauron:EnemigoSauron;
            public var enemigoVoldemor:EnemigoVoldemor;

            public function _CreateEnemigo(numero:int):Enemigo {
                switch (numero) {
                    case 1 :
                        trace("he creado un cell");
                        enemigoCell=new EnemigoCell  ;
                        return enemigoCell;
                    case 2 :
                        trace("he creado un maul");
                        enemigoDarthMaul=new EnemigoDarthMaul  ;
                        return enemigoDarthMaul;
                    case 4 :
                        trace("he creado un sauron");
                        enemigoSauron=new EnemigoSauron  ;
                        return enemigoSauron;
                    case 3 :
                        trace("he creado un voldemor");
                        enemigoVoldemor=new EnemigoVoldemor  ;
                        return enemigoVoldemor;
                    default :
                        return null;
```

```
                    }
                }
```

Con el método **_CreateEnemigo** podemos controlar que clase de enemigo creamos según un valor numérico, este valor está relacionado con la variable **numeroCombate.** Con esta variable integer que inicializamos en 1 asignamos el primer enemigo a la primera pantalla.

Empleando esta técnica, únicamente se debe controlar que cada vez que el jugador gane un combate la variable se incremente en uno, de este modo se construyen correctamente todos los combates.

Los escenarios y las barras de vida se cargan en las partidas de la misma forma, así es seguro que el enemigo estará con su barra de vida y escenario correspondiente.

Si quisieramos añadir un nuevo enemigo al juego únicamente se debe crear un nuevo case en el switch (en este caso sería case 5) y crear una instancia del tipo de **enemigo5**.

Con los patrones de diseño podemos simplificar código que de otra manera hubiera resultado tedioso y poco optimizado, cuanto mejor estructurado este nuestro código mas fácilmente podremos corregirlo, ampliarlo y reutilizarlo.

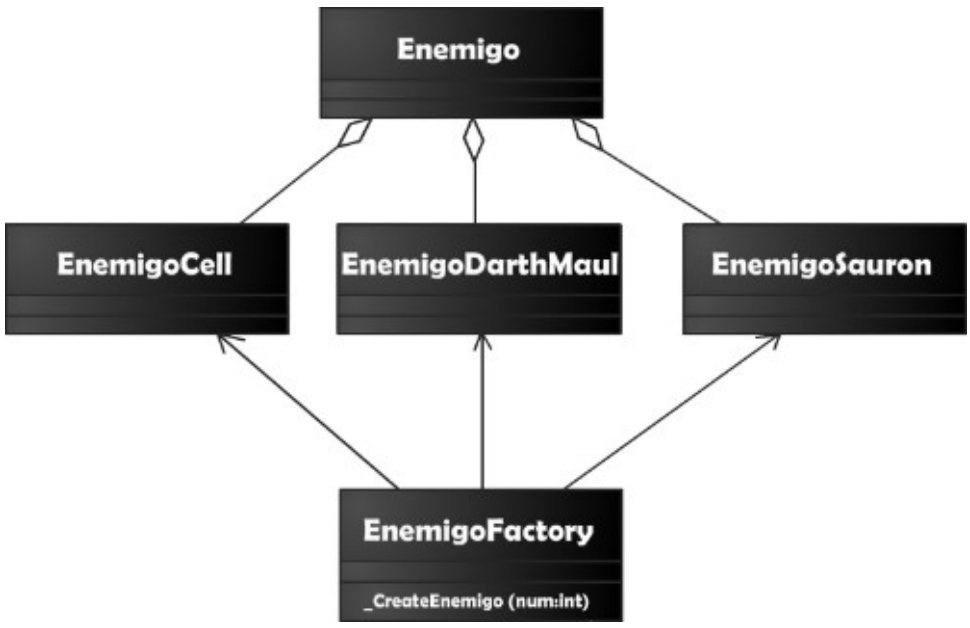

Ilustración 5.16: Esquema de Factory Method

5.4. Inteligencia artificial

5.4.1. Introducción

La inteligencia artificial o I.A es la ciencia de recrear comportamientos humanos en computadoras. En el sector de los videojuegos la I.A es fundamental para dotar realismo a las situaciones en que se encuentra el jugador, ya que cuanto mas trabajada esté más impredecible será el comportamiento de los enemigos y demás personajes.

Repasando brevemente la historia de la I.A podremos valorar el salto cualitativo que ha experimentado en un lapso relativamente corto de tiempo.

5.4.2. Historia de la I.A

Si queremos repasar la historia de la I.A es imperativo volver a nombrar a uno de los padres de la computación, **Alan Turing**. Turing diseñó en 1936 una **Máquina universal,** que demostraba formalmente que un dispositivo físico era capaz de implementar cualquier cómputo formalmente definido. Esto además de revolucionar la teoría de autómatas supuso el primer paso para la creación de comportamientos complejos en entes artificiales.

En el campo de los videojuegos el primer ejemplo tangible es también de Turing, cuando desarrolló el primer programa de ajedrez creó a su vez el primer ente computacional que era capaz de comportarse como un ser humano.

Años mas tarde, en los inicios de los 70, los movimientos de los enemigos estaban aún almacenados memoria, con lo que sus comportamientos seguían siempre las mismas pautas.

Después con la aparición de los primeros microprocesadores se añadió cierta aleatoriedad a los movimientos, por ejemplo **Space Invaders** utilizaba funciones **hash** (basadas en acciones de jugador) para añadir complejidad.

Otra mejora que se realizó en esa época fue en el juego **Pac-Man**, donde se añadió cierta "personalidad" a los enemigos para lograr un mayor realismo.

Los años 90 supusieron un boom en cuanto a nuevos géneros de videojuegos y con ellos a nuevas técnicas de I.A. Se empezaron a implementar **máquinas finitas de estados**, **búsqueda de caminos**, **decisiones en tiempo real**, **planificación**, etc. Esto supuso una mejora notable en el realismo de los comportamientos de los enemigos, aunque aún resultaban un tanto predecibles ya que estos tenían un número finito de movimientos.

Juegos como **Battlecruiser 300AD** añadieron la aplicación de redes neuronales. **Creatures** o **Black & Withe** son ejemplos de comportamientos emergentes donde la I.A ya era una mente que podía aprender de sus errores.

El primer videojuego en aplicar estas técnicas con verdadero éxito fue **GoldenEye 007**, que utilizaba una IA que reaccionaba a los movimientos y acciones de los jugadores.

En **Half-Life** (1998) los enemigos tenían la capacidad de trabajar juntos para buscar al jugador, se cubrían entre ellos, etc.
Halo (2001) fue un avance al permitir a a su I.A utilizar vehículos y otras acciones tácticas, la I.A era capaz de reconocer amenazas y actuar en consecuencia.

En **Far Cry** (2004) los enemigos reaccionaban al modo de juego del jugador, actuaban en consecuencia empleando tácticas militares. La I.A no hacía "trampas" para conocer la posición real del jugador sino que almacenaba la última posición conocida.

F.E.A.R (2005) fue el primer juego en emplear planificación en tiempo real para controlar la IA.

Left 4 Dead (2008) generaba de forma procedural diferentes experiencias para los usuarios cada vez que jugaban. La I.A era capaz de detectar cómo había jugado el usuario y trataba de añadir "eventos" que les ofrecían cierta sensación de "narrativa", logrando un juego mucho más dinámico.

5.4.3. Esquema de I.A en el videojuego The Condemned

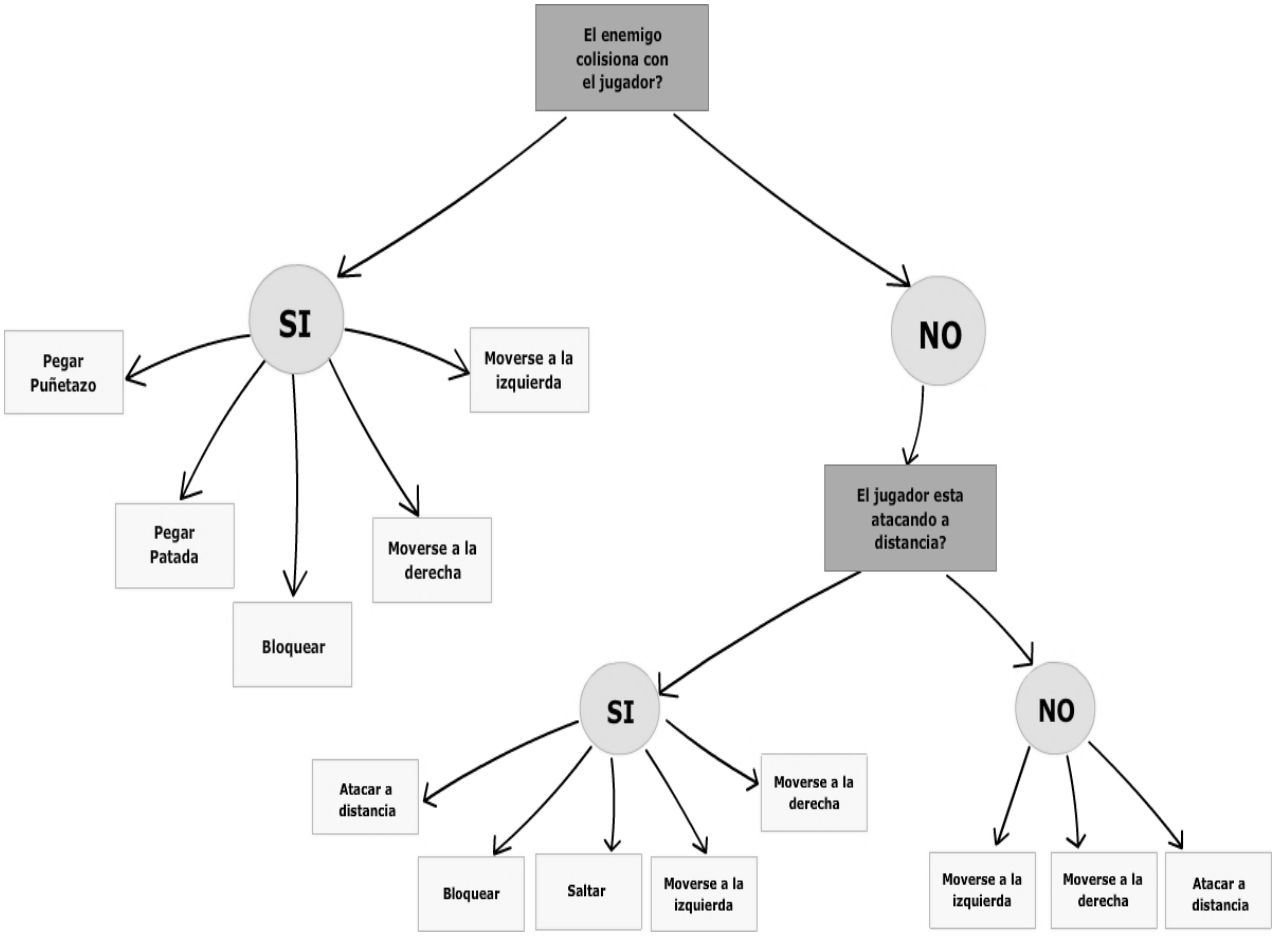

Ilustración 5.17 Esquema de la I.A en The Condemned

Este es el esquema de la I.A en el videojuego The Condemned. Una inteligencia artificial como se puede apreciar basada en **arboles de decisión**.

Los árboles de decisión es una técnica en la que las acciones de los personajes controlados por la I.A. responden a pautas predefinidas mediante estructuras en forma de

árbol con condiciones.

El enemigo tiene tres opciones a realizar si no se encuentra en contacto con el personaje jugador:

- Desplazarse a la izquierda
- Desplazarse a la derecha
- Realizar un ataque a distancia

Para controlar el comportamiento de la I.A. se ha creado la variable **Aprendizaje**, este parámetro indica lo inteligente que es el enemigo. Cuanto mayor sea el aprendizaje más rápido aprenderá el enemigo y más ataques realizará.

Según si el enemigo se encuentra en rango, es decir si está en el radio de choque con el jugador, realizará unos movimientos u otros. De esta forma, cuando el enemigo este colisionando con el jugador, podrá realizar cinco acciones:

- Pegar puñetazo
- Pegar patada
- Bloquear
- Avanzar
- Retroceder

Cada una de estas acciones empieza con un 20% de posibilidades de realizarse, pero si la acción del enemigo es considerada un "acierto", se incrementa la posibilidad de este tipo de acción y si es un "fracaso", se disminuye.

Supongamos que nos encontramos en el estado inicial en el que las probabilidades son 20, 20, 20, 20, 20, el enemigo se acerca al jugador, y en cuanto colisiona el jugador lanza un puñetazo, el enemigo realiza el movimiento de bloqueo y para el golpe, esto se considera un acierto, con lo que el sistema de aprendizaje aumenta el peso de que este movimiento vuelva a realizarse cuando se cumplan las mismas condiciones, siendo las probabilidades ahora, 16, 16, 24, 16, 16 y así sucesivamente.

Ahora bien, para que no todos los enemigos sean igual de inteligentes utilizamos la variable aprendizaje. El número que se suma o se resta a las probabilidades no es cuatro, es el valor de esa variable. Si se quiere incrementar la dificultad de un enemigo lo único que se debe hacer es ponerle más valor al atributo aprendizaje.

En resumen, los enemigos en The Condemned tienen un comportamiento que depende de las acciones del jugador y son capaces de aprender de sus errores.

Con el atributo aprendizaje y el juego de probabilidades logramos que los enemigos tengan un comportamiento realista y así la experiencia del jugador sea lo más entretenida posible y no resulte el juego repetitivo.

6. Pruebas

Es necesario en toda aplicación realizar una serie de pruebas para corroborar que su funcionamiento es el correcto en todo momento y que está carente de errores. Un error es un comportamiento distinto del que espera un usuario razonable.

La etapa de pruebas es una de las fases del ciclo de vida de los proyectos. Se la podría ubicar después del análisis, el diseño y la programación, pero dependiendo del tipo de proyecto su realización podría ser de forma paralela a las fases citadas.

La importancia de esta fase depende de las características del sistema desarrollado, llegando a ser vital en sistemas de tiempo real u otros en los que los errores sean irrecuperables.

Las pruebas no tienen el objeto de prevenir errores sino de detectarlos. Se efectúan sobre el trabajo realizado y se deben encarar con la intención de descubrir la mayor cantidad de errores posible.

6.1. Tipos de pruebas

- **Pruebas unitarias**: Las pruebas unitarias se realizan para controlar el funcionamiento de pequeñas partes de código como subprogramas en la programación estructurada o métodos en la programación orientada a objetos

- **Pruebas de integración**: Las pruebas de integración tienen como base las pruebas unitarias y consisten en una progresión ordenada de pruebas para las cuales los distintos módulos van siendo ensamblados y probados hasta haber integrado el sistema completo. Si bien se realizan sobre módulos ya probados en forma individual, no es necesario que se terminen todas las pruebas unitarias para comenzar con las de integración.

Con la finalidad de encontrar los máximos errores en la aplicación se han realizado multitud de pruebas, a continuación se explican detalladamente las más importantes.

6.2. Pruebas realizadas

6.2.1. Pruebas unitarias

- **Control del personaje**: Cada vez que se implementaba un nuevo movimiento en el personaje se han realizado las pruebas oportunas para comprobar que la acción era la esperada.

  En todos los casos menos en el movimiento de salto no se detectó ningún error. El salto presentaba un problema, el personaje saltaba pero se quedaba en el aire y no volvía a la posición inicial.

  Este error se solucionó mediante la creación de una variable llamada **gravedad.** Esta variable condiciona la propiedad **y** del personaje, es decir, su altura, de tal forma que cuando llega a la altura máxima del salto toma un valor negativo y el personaje vuelve a su altura inicial.

- **Puntuación**: En el momento en que se programó la lógica que controla las

puntuaciones se hicieron pruebas para corroborar que estaba todo correcto. Asimismo se comprobó no se reiniciara su valor cuando se cambia de pantalla y que se mostrara correctamente en la pantalla final y en la de game over.

- **Colisiones**: Sobre este factor se han realizado diversas pruebas ya que en un principio no se utilizaba la función **hitTestObject** sino que se comprobaban la colisiones mediante una diferencia de posiciones.

  Si esta diferencia era menor a un valor determinado se declaraba la colisión. Esto presentaba problemas ya que las posiciones de los personajes cambian constantemente y cuando uno saltaba por encima del otro el valor de sus posiciones cambiaba de sentido (de positivo a negativo y a la inversa) y la función dejaba de funcionar.

  Para solucionar este error se optó por utilizar la técnica explicada anteriormente ya que ofrece mejores resultados.

  Por otro lado, controlar que los personajes no traspasaran los límites del escenario también presentó un problema. En un principio la velocidad de salto en horizontal era de 20px y la del desplazamiento horizontal normal de 10px.

  Se controlaba que cuando se llegara a un límite se sumara o restara (dependiendo si era el límite izquierdo o el derecho) 10px para que no pudiera superar la posición del escenario. No se tuvo en cuenta que como la velocidad del salto era mayor era posible salir de los límites saltando. Esto se descubrió en una prueba y se arregló modificando la velocidad de salto a 10px.

- **Ataques a distancia**: Una vez se programó este movimiento se realizaron muchas pruebas ya que a diferencia de los otros movimientos este crea un nuevo MC que se dirige hacia el adversario. Esto aumenta la complejidad ya que se debe controlar su creación, su dirección, su movimiento y su eliminación.

  A priori no se encontró ningún problema pero realizando una prueba en la que se planteaban distintas situaciones se comprobó que si había un kame en el escenario cuando un personaje saltaba encima del otro el kame cambiaba su dirección y seguía persiguiendo al contrincante en vez de continuar su movimiento hasta el final del escenario.

  Esto se arregló mediante la creación de una variable llamada **direccionKame**, a esta variable se le da el valor Izquierda o Derecha en el momento que se inicia el ataque. De esta forma el movimiento del kame siempre irá en la misma dirección por mucho que se cambien las posiciones de los personajes.

- **Inteligencia artificial**: Para comprobar que la I.A. Estaba correctamente diseñada se asignaron diversos valores a la variable **aprendizaje** y se comprobó como modificaban la probabilidad de detener un golpe. Con esta prueba se pudo observar que el enemigo aprendía con mayor rapidez conforme mayor era su aprendizaje.

6.2.2. Pruebas de integración

- **Control de vidas**: Se realizó la misma prueba en todas las pantallas para confirmar que el control de las vidas era el adecuado. Dejando que la I.A.

venciera el combate se comprobaban los diversos factores:

- La vida debía restarse del número de vidas total
- El MC asociado a esa vida debía desaparecer
- El combate se tenía que reiniciar en la pantalla donde se encontraba el jugador en ese momento a no ser que no le quedaran vidas, lo que le llevaría a la pantalla de Game Over.

- **Finalización de Rounds y combates**: Hay una serie de condiciones que se deben cumplir cuando se finaliza un Round o un Combate, se han realizado diversas pruebas para comprobar que el comportamiento de la aplicación en todos los casos es el esperado.

  Cuando finaliza un Round

  - Lo primero que se debe controlar es si el Round que termina es el último del combate, es decir si es el segundo que gana uno de los dos contrincantes. Si es así dependiendo del ganador se realizarán una serie de acciones u otras, estas se explican con detalle más adelante.
  - Si el Round no finaliza el combate se deben controlar las siguientes acciones:
    - El tiempo tiene que volver a marcar 60 segundos.
    - Los MC que forman las barras de vida del jugador y de su adversario deben volver a estar en la situación inicial.
    - Tiene que aparecer la señal de Round ganado al lado del nombre del ganador del Round que acaba de finalizar.
    - Tiene que aparecer un MC anunciando correctamente el número de Round que se va a disputar a continuación.
    - Los contrincantes deben volver a su posición inicial.

  Cuando finaliza un combate

  Se ha realizado una prueba tomando como condición que el jugador ha salido victorioso del combate para controlar lo siguiente:

  - Si se encuentra en la última pantalla debe saltar a la pantalla donde aparece el símbolo de victoria y su puntuación.
  - Si no es así debe pasar a la pantalla que le corresponda, controlando lo siguiente:
    - Debe mantener su puntuación.
    - El número de vidas ha de estar actualizado.

- Los MC que señalan los Rounds ganados deben desaparecer.
- Su barra de vida debe volver al estado inicial.

La prueba que toma como condición que el jugador pierde el combate ha servido para comprobar lo siguiente:

- Si el jugador se ha quedado sin vidas debe aparecer la pantalla de Game Over junto a su puntuación.
- Si el jugador aún dispone de alguna vida se debe controlar lo siguiente:
  - El número de vidas debe disminuir
  - Se debe reiniciar el combate en la pantalla actual.
  - Los personajes deben volver a su posición inicial.
  - Los MC de las barras de vida deben volver a su estado inicial.
  - Los MC de los rounds ganados deben desaparecer.

- **Finalización de la cuenta atrás**: Se han realizado una serie de pruebas para determinar si la lógica respecto a la finalización del tiempo de combate era la correcta.

  Cuando se acaba el tiempo el Round debe finalizar, esto implica que los contrincantes deben dejar de luchar así que tanto el jugador como la I.A. Deben detenerse en ese instante. Según el nivel de vida de cada luchador estaremos en una de estas situaciones:

  - El jugador tiene mas vida que el enemigo: En esta situación el jugador es nombrado ganador del Round.
  - El enemigo tiene más vida que el jugador: En esta situación se declara al enemigo ganador del combate.
  - Los dos luchadores tienen la misma cantidad de vida: En este caso el Round queda anulado y se reinicia con el mismo número. Ninguno de los personajes sumará este Round como vencido.

7. GUIA DEL JUEGO

A continuación se explicará brevemente las distintas opciones que tiene el jugador cuando se dispone a jugar a "The Condemned", así como un resumen de las diferentes pantallas.

Para jugar a The Condemned hay que dirigirse a la página www.thecondemned.idomyweb.com, idomyweb proporciona un hosting gratuito a los desarrolladores que quieran subir su página a Internet sin coste alguno.

Ilustración 7.1: Pantalla de inicio del juego

Esta es la pantalla de inicio del videojuego **The Condemned,** en ella el jugador podrá iniciar el juego o acceder a una pantalla en la que podrá consultar los controles del juego.

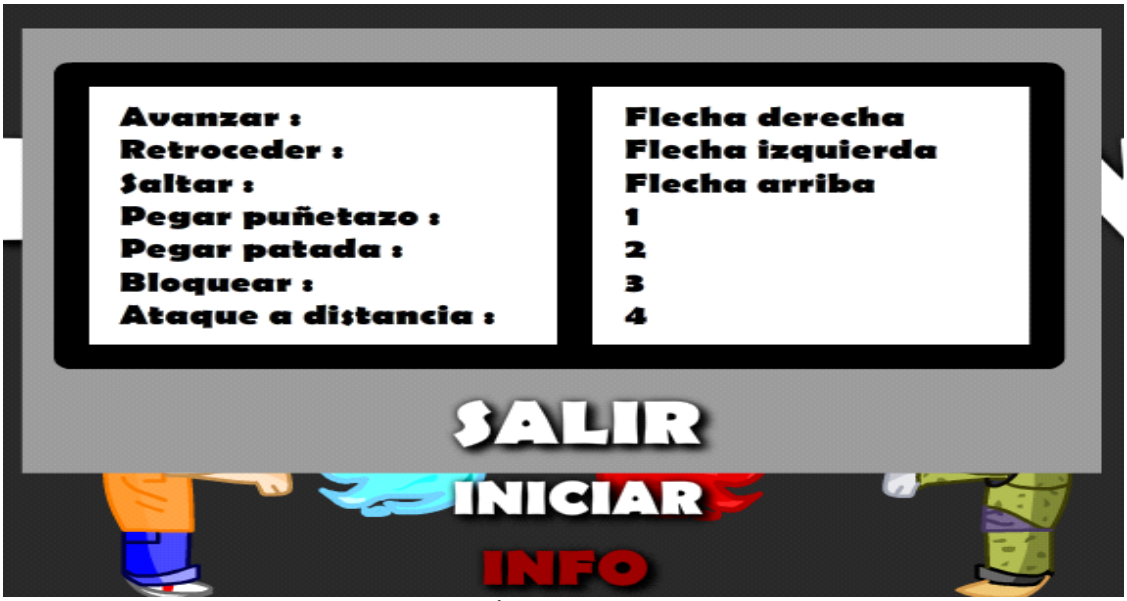

Ilustración 7.2: Pantalla INFO

De este modo el jugador conoce antes de empezar a jugar que teclas debe presionar para realizar los movimientos con el protagonista Conde.

Una vez el jugador conozca la relación entre teclas y movimientos puede empezar a jugar a THE CONDEMNED, el juego consta de cuatro pantallas, en cada una de ellas nos enfrentaremos a un oponente distinto.

Para ganar un combate es necesario vencer dos Rounds [11] , si un mismo contrincante vence en los dos primeros Rounds será el ganador del combate, en caso de que cada combatiente salga vencedor de un Round se hará un tercero de desempate.

En este videojuego los enemigos están inspirados en famosos malvados de la televisión y del cine, esta es la relación entre las distintas fases y los enemigos que nos encontramos:

- En la primera pantalla nos enfrentaremos a **Cell** de **Dragon Ball**
- En la segunda a **Darth Maul** de **Star Wars Episodio I**
- En la tercera pantalla nos econtraremos a **Voldermor** de **Harry Potter**
- Como enemigo final tendremos a **Sauron** de **El señor de los anillos**

El escenario de cada pantalla va acorde con el enemigo ambientando el mundo al que pertenece.

Esta es una captura de pantalla de la primera fase:

Ilustración 7.3: Primera pantalla del videojuego

Se puede apreciar que la estética del juego es de dibujo animado, los videojuegos Flash acostumbran a ser muy coloridos y se ha optado por seguir esa línea en la realización de este proyecto.

Si vemos detenidamente esta captura de pantalla podemos ver unos cuantos elementos que se

deben comentar:

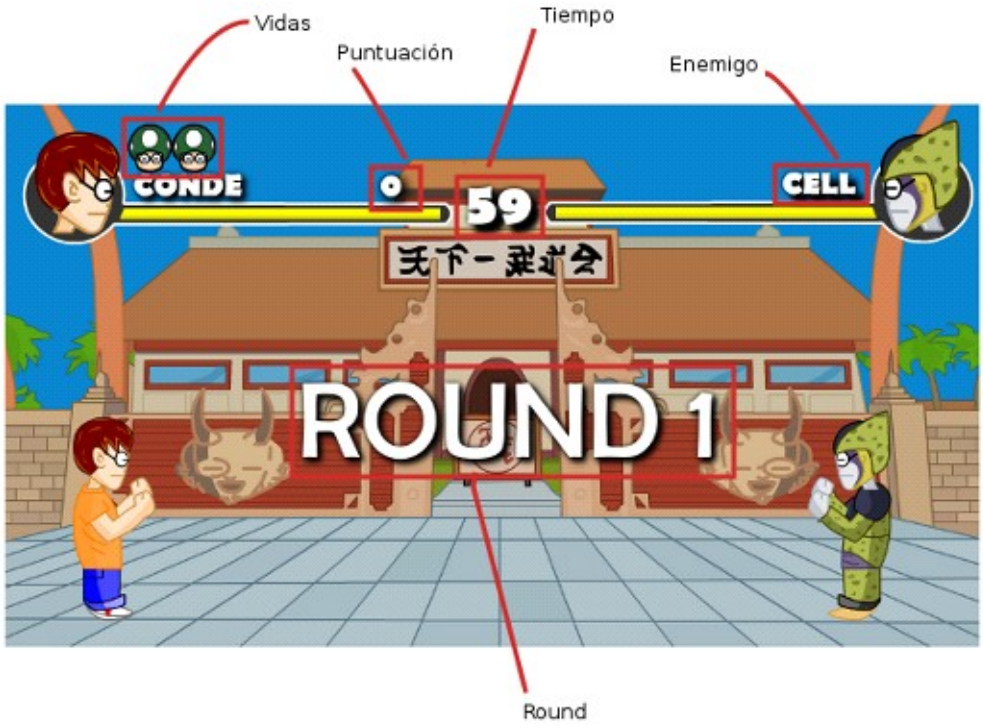

Ilustración 7.4: Elementos de una pantalla

- **Vidas:** En el inicio de la partida el jugador tiene dos vidas, si un enemigo nos gana un combate se restará una vida. En el momento en que se agoten las vidas el jugador perderá el juego y deberá empezar desde el principio.

  En un principio se optó por la posibilidad de poder guardar la partida para poder continuar en una determinada pantalla pero al final se rechazó la idea ya que en prácticamente todos los juegos de este tipo no existe esta opción.

- **Puntuación:** Cada vez que el jugador logra golpear al enemigo su puntuación aumenta y cada vez que el enemigo golpea al jugador esta disminuye. De esta manera se consigue que sea prácticamente imposible repetir una misma puntuación por mucho que se superen todas las fases del juego.

  La puntuación es un valor numérico que empezará de cero cada vez que se inicie el juego. Tanto en la pantalla final, una vez hayamos vencido a todos los enemigos, como en la pantalla de Game Over, se podrán consultar las puntuaciones.

- **Tiempo:** Cada Round tiene un tiempo de 60 segundos, en cuanto se inicia el combate empieza la cuenta atrás y en el momento en que llega a 0 termina el combate. Si en ese momento aún están los dos contrincantes con vida el ganador del Round será el que más vida tenga.

- **Enemigo:** En The Condemned hay cuatro enemigos diferentes, cada vez que ganemos un combate aparecerá un nuevo escenario con un nuevo contrincante hasta que lleguemos al enemigo final.

- **Round:** El oponente que gane dos Rounds será el ganador del combate, así pues el combate constará como máximo de tres Rounds.

Cuando uno de los dos luchadores gana un Round aparece en la pantalla el siguiente símbolo:

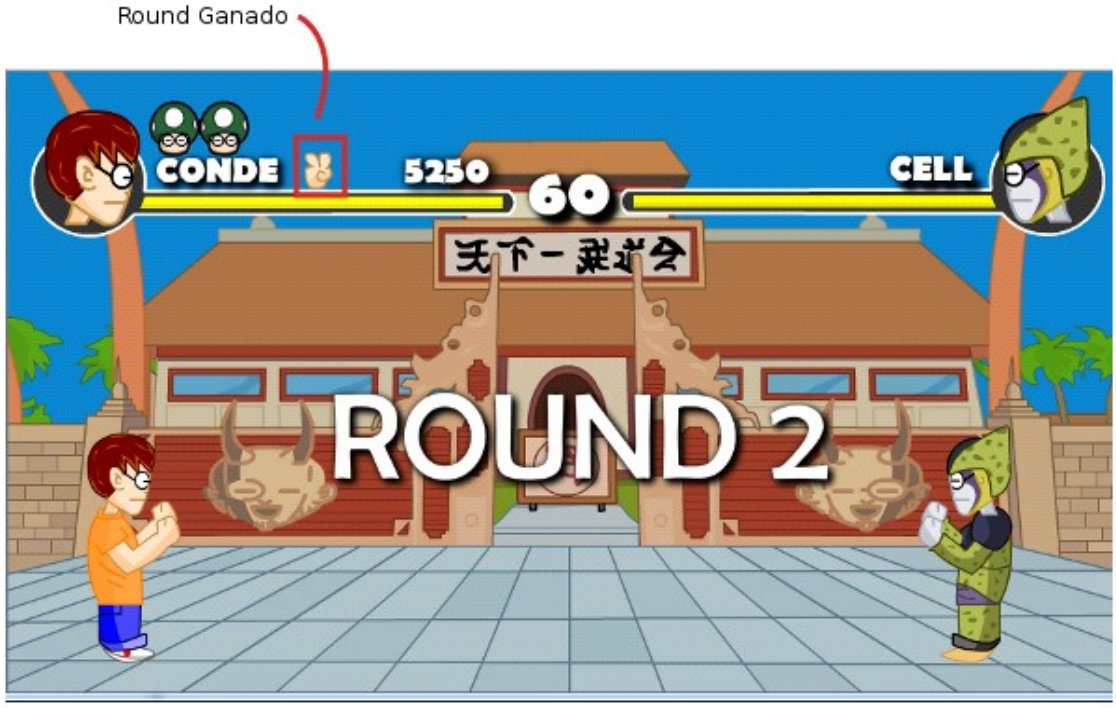

Ilustración 7.5: Round 2

Si logramos vencer a los enemigos que van apareciendo en las distintas fases llegaremos a la cuarta y última pantalla. A continuación mostramos una serie de capturas de las tres fases restantes así como la pantalla de victoria y la de Game Over.

Ilustración 7.6: Fase 2

Ilustración 7.7: Fase 3

Ilustración 7.8: Fase 4

Ilustración 7.9: Pantalla final del juego

Ilustración 7.10: Pantalla de Game Over

8. Conclusiones

8.1. Cumplimiento de objetivos

A continuación se hace una reflexión sobre si se ha cumplido cada uno de los objetivos del proyecto.

- O1. Creación de un videojuego pasando por todas las fases de desarrollo.

  Este objetivo se ha cumplido ya que se ha realizado el diseño gráfico, la programación y la fase de pruebas.

- O2. Programación óptima de la aplicación para poder ampliarla fácilmente en el futuro.

  Este objetivo se ha cumplido ya que se ha estructurado todo el programa mediante las buenas prácticas relacionadas con la ingeniería del software.

- O3. Desarrollo de una inteligencia artificial dinámica.

  Objetivo cumplido ya que los enemigos son más difíciles de vencer conforme se va avanzando de pantalla.

- O4. Ampliación de conocimientos de Flash y ActionScript 3.

  Objetivo logrado ya que mientras realizaba el proyecto he adquirido nuevos conocimientos en la materia y he consolidado los que ya tenía.

8.2. Desviaciones sobre la planificación

Varias de las tareas han tenido una duración mas larga de lo planificado lo que ha provocado que el tiempo total del proyecto aumentara, los cambios han sido los siguientes:

- La programación del movimiento del personaje ha durado 3 días más de lo previsto.
- La creación de la I.A. se ha realizado en 14 días en vez de en 10 días.
- La redacción de la memoria ha tenido una duración final de 35 días.

Por otra parte se hizo un cambio en la programación ya que en el inicio no se usaban las funciones a través de timers sino a través de funciones enter frame, es decir, la velocidad de los movimientos estaba relacionada con la velocidad de reproducción de los fotogramas, lo que resultaba un problema porque si aumentamos mucho esta velocidad el movimiento de los personajes dejaba de ser fluido y se movían a "saltos". Esta modificación atrasó el proyecto cinco días.

La duración final del proyecto ha sido de 282 horas en vez de las 250 horas previstas.

En la siguiente imagen se muestra el diagrama de Gantt actualizado del proyecto.

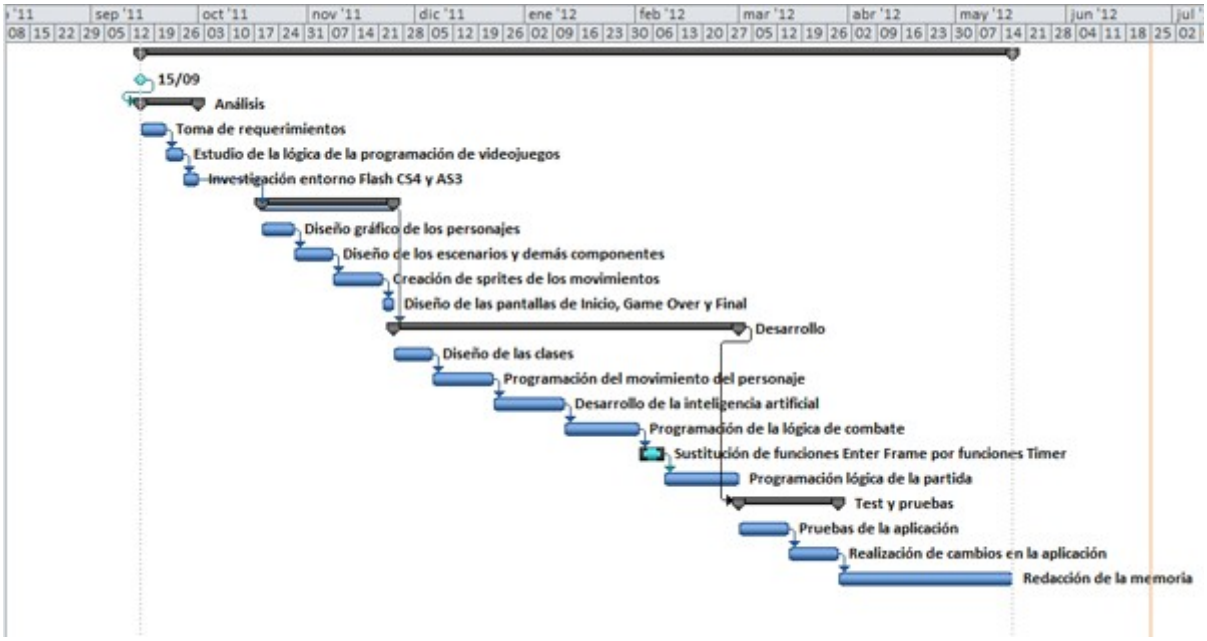

Ilustración 8.1. Diagrama de Gantt final

8.3. Líneas futuras

Las ampliaciones de este proyecto de cara al futuro son las siguientes:

- Añadir la posibilidad de caracterizar el personaje pudiendo escoger diferentes rasgos físicos.
- Crear más pantallas con nuevos enemigos.
- Añadir la opción de jugador contra jugador.
- Crear un sistema de registro de usuario y login para consultar el ranking de puntuaciones.
- Integrar el videojuego en facebook para que tenga una mayor difusión.

8.4. Valoración personal

Soy consciente de que la creación de un videojuego es algo atípico para el proyecto final de carrera de una ingeniería informática y en ocasiones me he planteado si mi elección era la correcta o si debería haber optado por un tipo de trabajo más "práctico" o con mas "futuro".

Cuando estamos acabando una carrera aún no tenemos claro que perfil profesional tenemos, la elección de un proyecto suele estar supeditada a lograr adquirir las aptitudes necesarias para dominar un lenguaje de programación o un determinado sistema.

En mi caso la elección ha sido condicionada por mis deseos de crear algo desde cero, tenía muy claro que mi proyecto debía ser 100% desarrollado por mi. Otra de las condiciones que

me auto impuse es que debía tratarse de un proyecto que me divirtiera mientras lo realizaba, no quería terminar el trabajo con hastío y enterrar la memoria en un cajón para no volver a verla, mi intención siempre ha sido terminar este trabajo como lo empecé, con orgullo e ilusión.

Se puede creer que el no realizar un trabajo tipo tiene sus inconvenientes ya que cuando aparece un problema no se dispone de ninguna documentación para consultarla. En vez de tomar esto como un problema me he tomado las dificultades como retos en los que he tenido que utilizar todos mis conocimientos y mi creatividad para superarlos.

Diseñar mis propios personajes me ha producido una gran satisfacción, la sensación que se experimenta cuando la pantalla en blanco se transforma en un universo creado por uno mismo es increíble. Más aún cuando empezamos a controlar ese universo mediante líneas y líneas de código.

La realización de este proyecto me ha servido para afianzar mis conocimientos no solo de Flash y ActionScript sino del desarrollo de software en general. No me he contentado con que la aplicación funcionase sino que he intentado en todo momento optimizar al máximo la programación, por esta razón he utilizado clases y patrones de diseño.

Antes de empezar este proyecto no era consciente de la importancia de factores como los comentarios del código o la realización de pruebas. Cuando una aplicación empieza a tener un tamaño considerable es imprescindible comentar todo el código porque resulta sencillo perderse aunque sea tu propio programa.

La realización de pruebas y su documentación escrita me ha servido para detectar y resolver numerosos errores que de otra manera hubiera tardado mucho en solventar. Es de vital importancia presentar cualquier aplicación libre de fallos.

En resumen, este proyecto ha cumplido mis expectativas, he ampliado conocimientos mientras me divertía y he podido experimentar lo que se siente al crear un videojuego desde cero pasando por todas las fases de su desarrollo.

Tipología y palabras clave

Adobe Flash CS4: Aplicación de creación y manipulación de gráficos vectoriales con posibilidad de manipulación de código mediante un lenguaje llamado ActionScript. [1]

ActionScript 3: Lenguaje de programación de la Plataforma Adobe Flash. [2]

Programación: Procedimiento informático basado en algoritmos (listas de instrucciones para resolver problemas abstractos) que trata de crear, probar y depurar el código fuente de programas computacionales. [3]

Ingeniería del software: Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y el estudio de estos enfoques, es decir, la aplicación de la ingeniería al software. [4]

Inteligencia artificial: Ciencia de recrear comportamientos humanos en computadoras. [5]

Display: Dispositivo de ciertos aparatos electrónicos que permite mostrar información al usuario de manera visual. [6]

Osciloscopio: Instrumento de medición electrónico para la representación gráfica de señales eléctricas que pueden variar en el tiempo. [7]

Periférico: Aparatos y/o dispositivos auxiliares e independientes conectados a la unidad de procesamiento de una computadora. [8]

Capa: Grupo de elementos que comparten un mismo nivel en la estructura de un archivo. [9]

Símbolo: Gráfico, botón o MovieClip creado por el usuario y que puede reutilizar en el transcurso de la película o en otras películas. [10]

Round: Asalto, el ganador de dos round será el ganador del combate. [11]

Kame: Ataque a distancia similar al que se realiza en otros videojuegos como Street Fighter II. [12]

Bibliografía

Libros

Colin Moock, ActionScript 3.0. ANAYA 2008

Joey Lott / Danny Patterson, ActionScript 3, Patrones de diseño. ANAYA 2007

Stuart Rusell, Inteligencia artificial. Pearson 2012

Recursos web

http://livedocs.adobe.com/flash/9.0_es/UsingFlash/flash_cs3_help.pdf

http://livedocs.adobe.com/flash/9.0_es/main/flash_as3_programming.pdf

http://livedocs.adobe.com/flash/9.0_es/main/flash_as3_components_help.pdf

www.cristalab.com

Indice de ilustraciones

Ilustración 2.1. Lanzamiento de misiles, 5
Ilustración 2.2. OXO, 5
Ilustración 2.3. Tennis for Two, 6
Ilustración 2.4. Spacewar, 6
Ilustración 2.5. Pong, 7
Ilustración 2.6. Space Invaders, 7
Ilustración 2.7. Primera generació (Tennis for two), 8
Ilustración 2.8. Segunda generación (Pacman), 8
Ilustración 2.9. Tercera generación (Mario Bros), 8
Ilustración 2.10. Cuarta generación (S.Fighter II), 8
Ilustración 2.11. Quinta generación (Metal Gear), 9
Ilustración 2.12. Sexta generación (FIFA), 9
Ilustración 2.13. Séptima generación, 9
Ilustración 2.14. Octava generación, 9
Ilustración 2.15. WOW, 10
Ilustración 2.16. Angry Birds, 10
Ilustración 3.1. Capcom vs Marvel, 11
Ilustración 3.2. Final Fight 3, 11
Ilustración 3.3. Quake 2, 12
Ilustración 3.4. Grand Theft Auto IV, 12
Ilustración 3.5. Metal Slug, 13
Ilustración 3.6. Super Mario Wii, 13
Ilustración 3.7. Pro evolution soccer 5, 14
Ilustración 3.8. Gran turismo 5, 14
Ilustración 3.9. Flight Simulator X, 14
Ilustración 3.10. Sim City 4, 15
Ilustración 3.11. The sims 3, 15
Ilustración 3.12. Guitar Hero 5, 16
Ilustración 3.13. Monkey Island, 16
Ilustración 3.14. Guild Wars, 17
Ilustración 3.15. Command & Conquer: Red Alert 3, 17
Ilustración 3.16. Brain Training, 17
Ilustración 3.17. Tetris, 18
Ilustración 4.1. Work Breakdown Structure, 23
Ilustración 4.2. Tabla de tareas, 25
Ilustración 4.3. Representación gráfica, 25
Ilustración 5.1. Capas en el diseño, 29
Ilustración 5.2. Herramienta papel de cebolla, 30
Ilustración 5.3. Cronología de Flash, 33
Ilustración 5.4. Línea de tiempo, 33
Ilustración 5.5. Ejemplo línea de tiempo estado 1, 35
Ilustración 5.6. Ejemplo línea de tiempo estado 2, 35
Ilustración 5.7. Ejemplo línea de tiempo estado 3, 35
Ilustración 5.8. Distintos MC que aparecen en el videojuego, 36
Ilustración 5.9. Movimiento de Conde a la derecha, 38
Ilustración 5.10. Movimiento de Conde a la izquierda, 39
Ilustración 5.11. Movimiento de Conde saltando, 39
Ilustración 5.12. Movimiento de Conde pegando puñetazo, 39
Ilustración 5.13. Movimiento de Conde pegando patada, 39
Ilustración 5.14. Movimiento de Conde bloqueando, 40
Ilustración 5.15. Movimiento de Conde atacando a distancia, 40
Ilustración 5.16. Esquema de Factory Method, 48

Ilustración 5.17. Esquema de la I.A en The Condemned, 50
Ilustración 7.1. Pantalla de inicio del juego, 57
Ilustración 7.2. Pantalla INFO, 57
Ilustración 7.3. Primera pantalla del videojuego, 58
Ilustración 7.4. Elementos de una pantalla, 59
Ilustración 7.5. Round 2, 60
Ilustración 7.6. Fase 2, 60
Ilustración 7.7. Fase 3, 61
Ilustración 7.8. Fase 4, 61
Ilustración 7.9. Pantalla final del juego, 62
Ilustración 7.10. Pantalla de Game Over, 62
Ilustración 8.1. Diagrama de Gantt final, 64

Indice de tablas

Tabla 1: Criticidad de los objetivos, 19
Tabla 2: Stakeholders, 20
Tabla 3: Prioridad requisitos funcionales, 21
Tabla 4: Prioridad requisitos no funcionales, 22
Tabla 5: Relación entre los Objetivos y los requisitos, 22
Tabla 6: Fases del proyecto, 23
Tabla 7: Milestones, 24
Tabla 8: Tabla de riesgos, 26
Tabla 9: Planes de contingencia, 27

Sabadell 26 de junio de 2012

Daniel Barbadillo Dubon