***Títol: Monturiol: El joc.***

***Volum: 1 de 1***

***Alumne: Alberto muñoz Rodríguez***

***Director/Ponent: Luis Solano Albajes***

***Departament: Llenguatges i Sistemes Informàtics***

**DADES DEL PROJECTE**

*Títol del Projecte:Monturiol: El joc*

*Nom de l'estudiant: Alberto Muñoz Rodríguez*

*Titulació:Enginyeria Técnica de Sistemes*

*Crèdits:22,5*

*Director/Ponent: Lluis Solano Albajes*

*Departament: Llenguatges i Sistemes Informàtics*

**MEMBRES DEL TRIBUNAL** *(nom i signatura)*

*President:Daniela Tost Pardell*

*Vocal:Victor Rotger Cerdà*

*Secretari: Lluis Solano Albajes*

**QUALIFICACIÓ**

*Qualificació numèrica:*

*Qualificació descriptiva:*

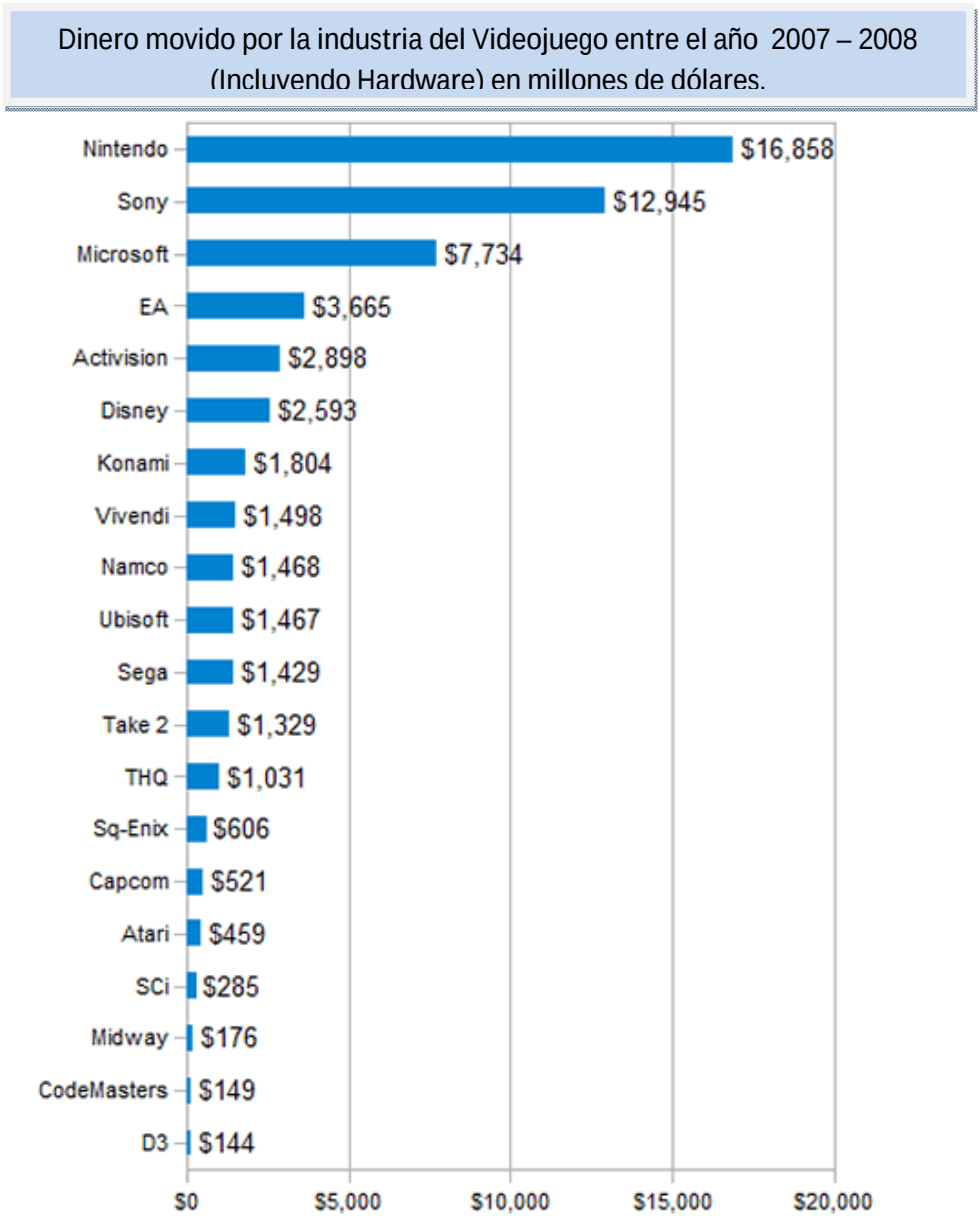
*Data:26/01/2010*

# ÍNDICE

# CAPÍTULO 1: INTRODUCCIÓN

El proyecto "*Monturiol: el Joc*" se presenta como un proyecto final de carrera de dos alumnos de Ingeniería técnica informática de sistemas de la Facultad de Informática de Barcelona, perteneciente a la Universidad Politécnica de Cataluña, realizando el proyecto para otra facultad de la UPC, concretamente para la ETSEIB y financiado por la Generalitat de Cataluña.

Hoy en día la industria de los videojuegos es uno de los sectores que mueve más dinero en el mundo, y no sólo tiene como componente entretenimiento y diversión, también se mueve en el ámbito de la educación. Con esta idea la Generalitat de Catalunya propuso un videojuego hecho en Flash para celebrar el 150 aniversario de la prueba del primer submarino creado por Narcís Monturiol y así, de este modo, los niños conocieran quien fue y todo lo que hizo.



Tal propuesta llegó a Lluís Solano, el cual la propuso como proyecto final de carrera y a nosotros nos resultó totalmente atrayente.

Dentro de lo que engloba este gran proyecto, la parte que conforma nuestro PFC es el apartado multijugador, con la finalidad de que los niños y niñas puedan jugar entre ellos por internet, siendo dos partes tan importantes tanto la creación del videojuego como la creación del servidor con quien se establecen las conexiones entre los diferentes clientes.

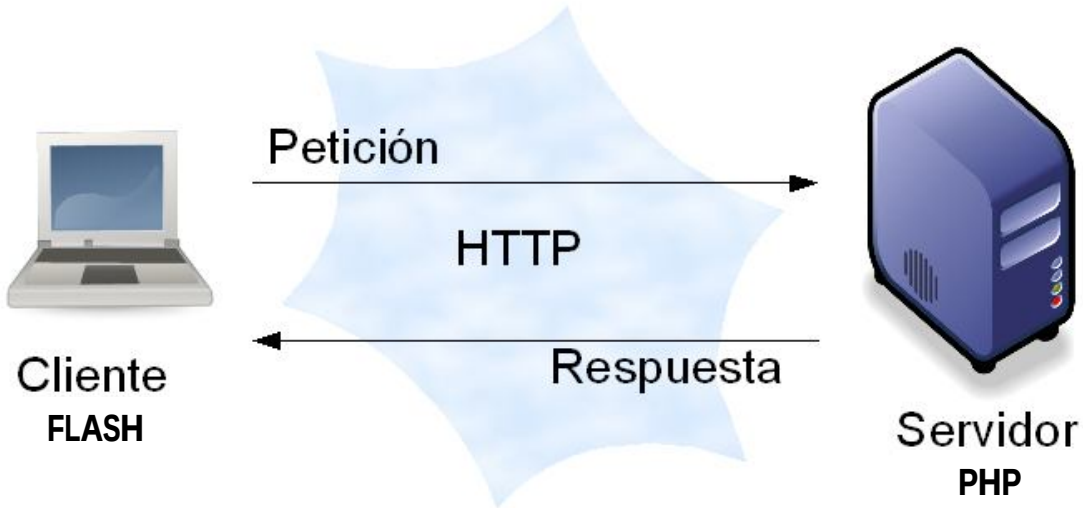
Diferentes imágenes del extenso mercado de videojuegos multijugador en la actualidad. Empezando por la izquierda Call of duty, Pro evolution Soccer 2009, Ogame, World of warCraft, Age of empires, Need for speed representativas la mayoría de géneros.

## 1.1 Descripción general del proyecto

Este proyecto consta de la elaboración de un videojuego en Flash, que mediante un servidor PHP nos permite ofrecer un servicio de multijugador al juego y, de este modo, poder jugar vía internet con otras personas en diferentes máquinas. El proyecto se divide en dos ramas, cada una de ellas desarrolladas por cada uno de los miembros que forman este proyecto.

Una parte es el videojuego, diseñado en ActionScript 3, que consta de una sala donde están todos los usuarios que han ingresado y desde donde pueden hablar entre ellos usando el chat incorporado, escoger el submarino con el que participarán en las carreras y editarlo, tanto gráfica como técnicamente, crear partidas, cerrarlas y, obviamente, jugar. Dichas partidas se podrán jugar individualmente, pudiendo de este modo probar la carrera, o contra otros jugadores, estableciendo las conexiones con el servidor. El tipo de juego que se desarrollará se basa en carreras submarinas en las que el primero en llegar a la meta será el ganador.

La otra parte es un servidor multijugador masivo programado en PHP utilizando sockets tcp/IP y un sistema de comunicación basado en XML diseñado, especificado e implementado para servir de anfitrión genérico para juegos Flash multijugador implementando todas las funcionalidades necesarias para integrar todo un sistema para conectar jugadores entre sí mediante una sala o chat, y la creación de servidores de comunicación entre jugadores para poder realizar partidas online sin que ningún usuario tenga que hacer de host para disfrutar de la partida.

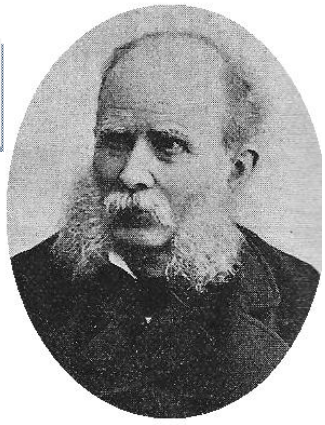


## 1.2 *Monturiol: el joc*

*Monturiol: el juego*, nos muestra los inventos más importantes que en su día creó Narcís Monturiol, y lo hace adentrándose dentro de sus sueños. Este juego se divide en dos vertientes, la faceta de un sólo jugador y la de multijugador.

Por un lado, encontramos un apartado de un jugador da a elegir entre diferentes submarinos para sumergirnos en el mundo de Monturiol, donde desde un menú principal se puede acceder a todo tipo de juego donde controlar el submarino, desde navegar para recoger monedas, explorar cuevas, carreras de submarinos y mucho más. Y de este modo, con libros explicativos de la vida de Monturiol y con preguntas tras finalizar cada juego, los más pequeños de la casa, y no tan pequeños, consiguen disfrutar un buen rato con el juego y además aprender todo lo que Monturiol llegó a ser y hacer.

Por otro lado, está el modo multijugador, en el cual, dos jugadores compiten entre ellos en una carrera con sus submarinos para ver quién es el más rápido y llega antes a la meta.



Narcís Monturiol y la réplica de su submarino Ictíneo tomada en el puerto de BCN en el año 2003

## 1.3 Objetivos

### 1.3.1 Objetivos del juego

- Documentar-se en ActionScript 3.
- Decidir el medio de comunicación.
- Especificar Diagrama de casos
- Especificar Diagrama de clases
- Especificar comunicación XML
- Implementar videojuego Offline.
- Implementar clase para comunicaciones.
- Implementar entorno sala.
- Añadir modo Online al Juego.
- Adherir del juego a la sala.

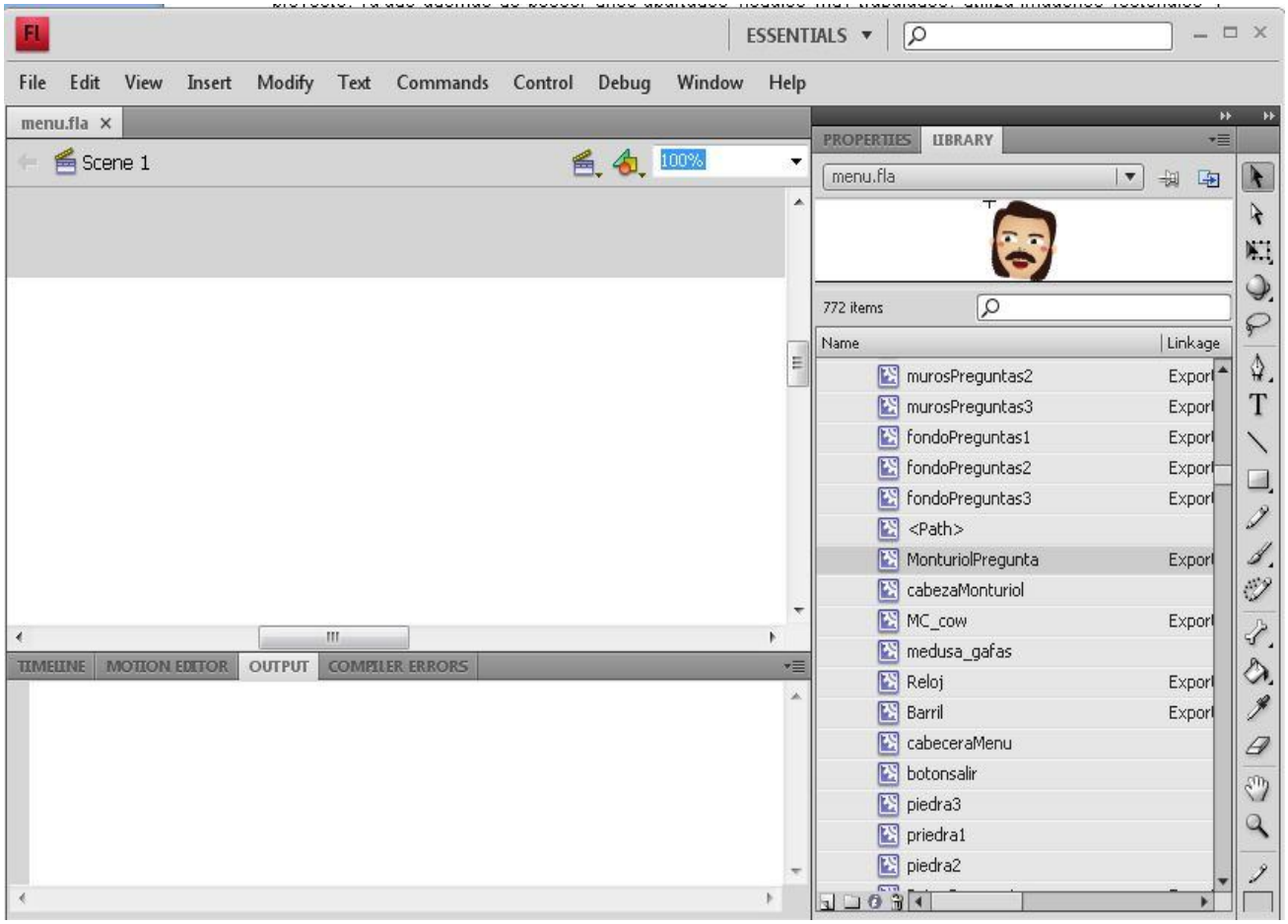
### 1.3.2 Objetivos del servidor

- Documentación y aprendizaje de PHP
- Documentación y aprendizaje de XML
- Documentación y aprendizaje de estructuras Cliente/servidor para juegos multijugador
- Diseñar e implementar varios servidores específicos en PHP
    - o Login.php
    - o Logout.php
    - o Com.php
    - o Sala.php
    - o Mapas.php
- Especificación y creación de un protocolo para la comunicación Servidor/cliente basado en XML
- Integración y demostración del servidor PHP para un juego FLASH AS3

# CAPÍTULO 2: ENTORNO DE DESARROLLO Y LENGUAJE

El entorno de desarrollo destinado para la creación del juego es Flash. Esta tecnología nos permite crear animaciones gráficas vectoriales independientemente del navegador, ya que únicamente hace falta un programa común en el ordenador donde queremos visualizarlo para que funcione. Así pues, se puede generar un videojuego accesible desde internet visualmente atractivo y con poco peso.

Con respecto al lenguaje de programación Flash consta de tres diferentes versiones de su lenguaje, ActionScript(AS). Hasta la fecha tenemos AS1, AS2 y AS3. Cabe destacar que anteriormente Flash era un entorno más pensado para diseñadores gráficos y la programación del lenguaje de los dos primeros así lo reflejaba. Sin embargo, con la aparición de AS3, que cambió considerablemente, se le dio una nueva visión más pensada para programadores.




Entorno de desarrollo Adobe Flash CS4

## 2.1 Flash

Adobe Flash, antes Macromedia Flash, es un entorno muy extendido por Internet, y en cualquier sitio se pueden encontrar todo tipo de aplicaciones hechas en Flash que nosotros tan siquiera sabemos; desde juegos, como es nuestro caso, hasta visores de video, o visores de imágenes.

El hecho de que se trata de un medio que no requiere grandes máquinas para su visualización, juntamente con que ha sido sobradamente probado en internet, lo convierten en un programa muy útil para nuestro proyecto, ya que además de poseer unos apartados visuales muy trabajados, utiliza imágenes vectoriales y tiene un gran potencial en la creación de animaciones y videojuegos.

Finalmente, el gran punto a tener en cuenta es la conexión y la seguridad por internet. Puesto que nuestro objetivo es brindar la posibilidad de que jueguen más de un jugador a la vez comunicándose con el servidor, resulta muy importante tanto el poder realizar dicha conexión, algo que Flash solventa con bajo coste, como el hacerlo de forma segura, ya que quiere evitar ataques malintencionados a nuestro servidor y cliente. En este aspecto, Flash se refuerza cada vez más, puesto que el programa encargado de visualizar los resultados de Flash es periódicamente actualizado, arreglando los agujeros de seguridad por internet y haciendo más robusta nuestra aplicación.

## 2.2 ActionScript 3

ActionScript es el lenguaje de programación que utiliza Flash. Como ya he comentado antes hay 3 versiones diferentes: AS 1.0, 2.0 y 3.0. Obviamente, la última versión es la que se utiliza, pero cabe destacar un montón de ventajas que nos reporta esta última con respecto a las dos anteriores.

Para comenzar, y uno de los aspectos más importantes, es que AS3 es un lenguaje orientado a objetos mucho más eficiente que sus dos predecesores, llegando a conseguir una velocidad de ejecución hasta diez veces más rápida. Esto se debe al hecho de que AS3 usa una nueva máquina virtual, diferente a la de las otras dos. Además, ahora son detectadas más excepciones en tiempo de ejecución, y de este modo conseguimos depurar más fácilmente los errores que con AS1 y AS2. A pesar esta ventaja, cabe destacar un inconveniente en común, pues en ningún caso se posee un depurador de errores en condiciones.

La sintaxis es el otro punto a favor. El hecho de decir que los anteriores lenguajes eran más destinados a diseñadores es principalmente a causa de esto. En ActionScript 2 las declaraciones de tipo era una opción que tenía el desarrollador para anotaciones suyas, pero la asignación de la clase se daba dinámicamente. En cambio, en ActionScript 3 se tienen que declarar, y de este modo se evitan posibles errores, haciendo más sólidos los programas y además consumiendo menos memoria y mejorando el rendimiento. En la sintaxis también han aparecido nuevos tipos, ya que al ser dados dinámicamente eran más genéricos. Por ejemplo, antes todos los números eran de tipo coma flotante, pero ahora han aparecido int, uint, haciendo así que las operaciones sean más simples y cuesten menos recursos.

Estas premisas dejan claro que de los tres lenguajes que pertenecen a Flash es esta tercera versión la más adecuada para nuestros fines.

## 2.3 Alternativa

Nuestro proyecto estaba pensado desde un principio para Flash, pero cabe destacar que no es la única opción viable con la que llevarlo a cabo. La alternativa en cuestión es Java. Java es un lenguaje muy usado también en el ámbito de internet, entre muchas otras, y puede aportar cosas que Flash no puede. Java es un lenguaje muy potente, con el que se puede llegar a hacer lo mismo que con Flash, siendo este primero multiplataforma, cosa que Flash no es. Este inconveniente viene dado por el hecho de que Flash pertenece a una empresa, en estos momentos Adobe, y que de momento sólo tiene versiones para Windows y Mac. Además, tenemos que añadir una diferencia muy significativa, ya que mientras Java es un lenguaje de código abierto, Flash no cumple esta característica, aunque con la cercana llegada de HTML5 Adobe ha comenzado a moverse y ha puesto algunos de sus componentes en código abierto y seguramente continuará haciéndolo temiendo una pérdida de mercado.

Como he explicado, Java es una potente alternativa y que no se puede desestimar en ningún caso. Sin embargo, a pesar de que parezca que Flash tiene las de perder, hay que destacar que para desarrollar un mismo producto en Java es más complicado y además Flash aporta el trabajo con gráficos vectoriales, puesto que si queremos escalar una imagen la mejor opción es que sea una imagen vectorial para no perder el objeto inicial y sus perspectivas. También en este ámbito encontramos que la manipulación de dichas imágenes y su importación es perfecta desde el entorno de trabajo que usan los diseñadores, parte indispensable para un videojuego.

# CAPÍTULO 3: REQUISITOS

## 3.1 Requisitos de desarrollo

Para poder desarrollar este juego el primer paso que se necesitaba era un ordenador y el programa Adobe Flash CS4. El ordenador en cuestión no debe ser muy potente, con una máquina normal se puede trabajar sin problemas, pero debido a que Flash consume muchos recursos para trabajar en el apartado más gráfico, animaciones y edición de imágenes vectoriales, los aspectos a tener más en cuenta son la memoria RAM, para poder moverse por el entorno lo más rápido y fluido posible y también un procesador actual, para que la compilación no tome demasiado tiempo.

El sistema operativo puede ser cualquiera, pero hay que tener en cuenta que Adobe Flash tiene versiones para Windows y Mac. Por lo tanto, son los dos sistemas operativos más adecuados, ya que si se quiere utilizar en Linux habrá que hacer una emulación de uno de estos SO o bien utilizar uno de los editores para Linux, que no son de Adobe, aunque el inconveniente de estos es que no tienen las mismas capacidades que Flash. En nuestro caso usaremos Adobe Flash sobre Windows.

Para hacer las pruebas y finalmente conseguir el producto del juego multijugador, también hace falta acceso a Internet, y de este modo conectarse a nuestro servidor para todas las pruebas necesarias.

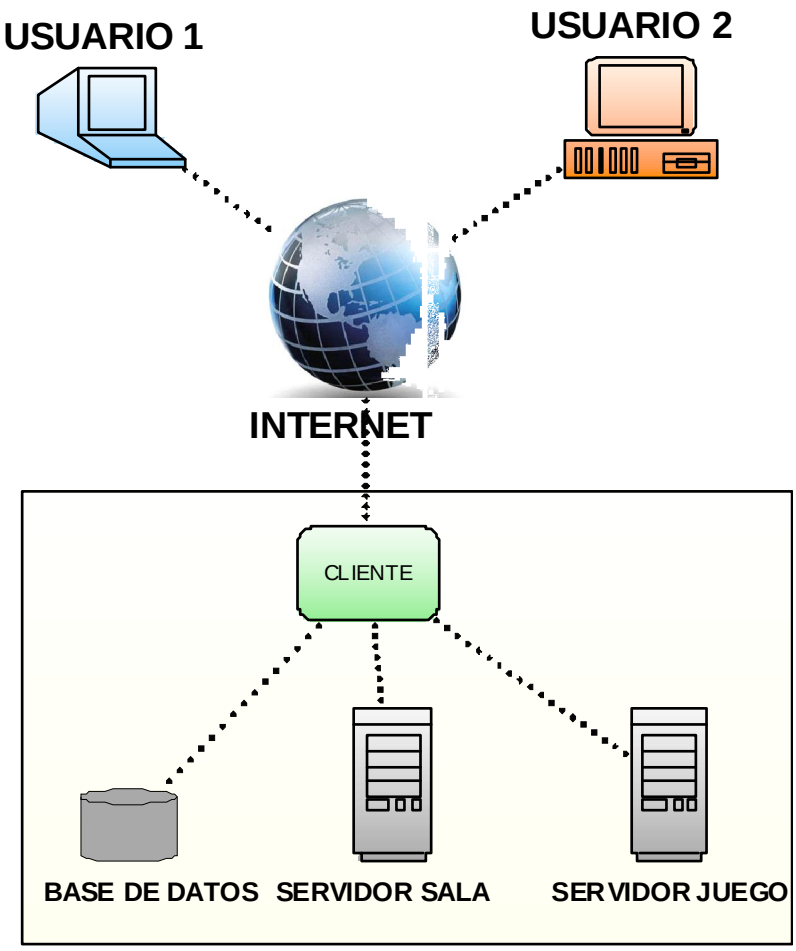
## 3.2 Requisitos de usuario

La visión que tendrá el usuario será única y exclusivamente de un portal web donde accederá al juego multijugador desde el navegador y donde podrá jugar y comunicarse con otros jugadores sin hacer ningún tipo de acción. Por lo tanto los requisitos que tienen los usuarios son mínimos. El primer paso que hay es un ordenador con acceso a Internet, para acceder a la web y poder jugar en modo multijugador. Finalmente también es necesario que tenga instalado el programa visualizador de Flash, que se trata de un *Plug-in* oficial subministrado por Adobe.

# CAPÍTULO 4: ESPECIFICACIÓN

## 4.1 Esquema del sistema

El siguiente esquema muestra todas las comunicaciones que habrán, siendo nosotros el jugador 1:

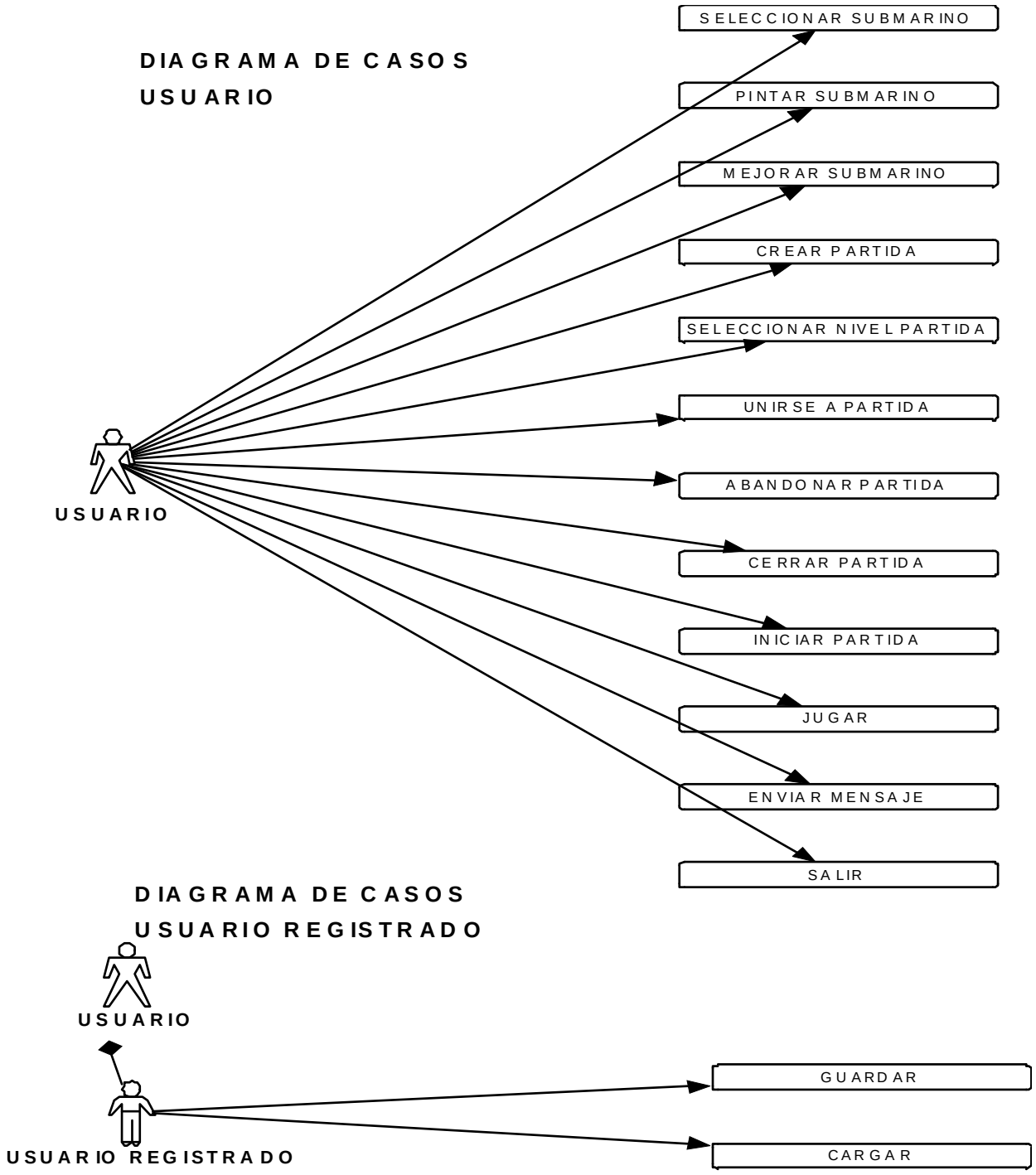



El funcionamiento es el mismo para todos los usuarios. Se conectan al cliente a través de internet. En caso de tener un usuario registrado consulta la base de datos. Tanto si ha entrado como usuario registrado, como si no se conecta al servidor de la sala, donde están todos los otros jugadores. Desde aquí puede crear las partidas y comenzar a jugar.

Una vez empiece a jugar, el cliente se conectará al servidor de juego, donde se participara en una carrera, contra otro usuario que también estará conectado a este servidor.

## 4.2 Diagrama de casos

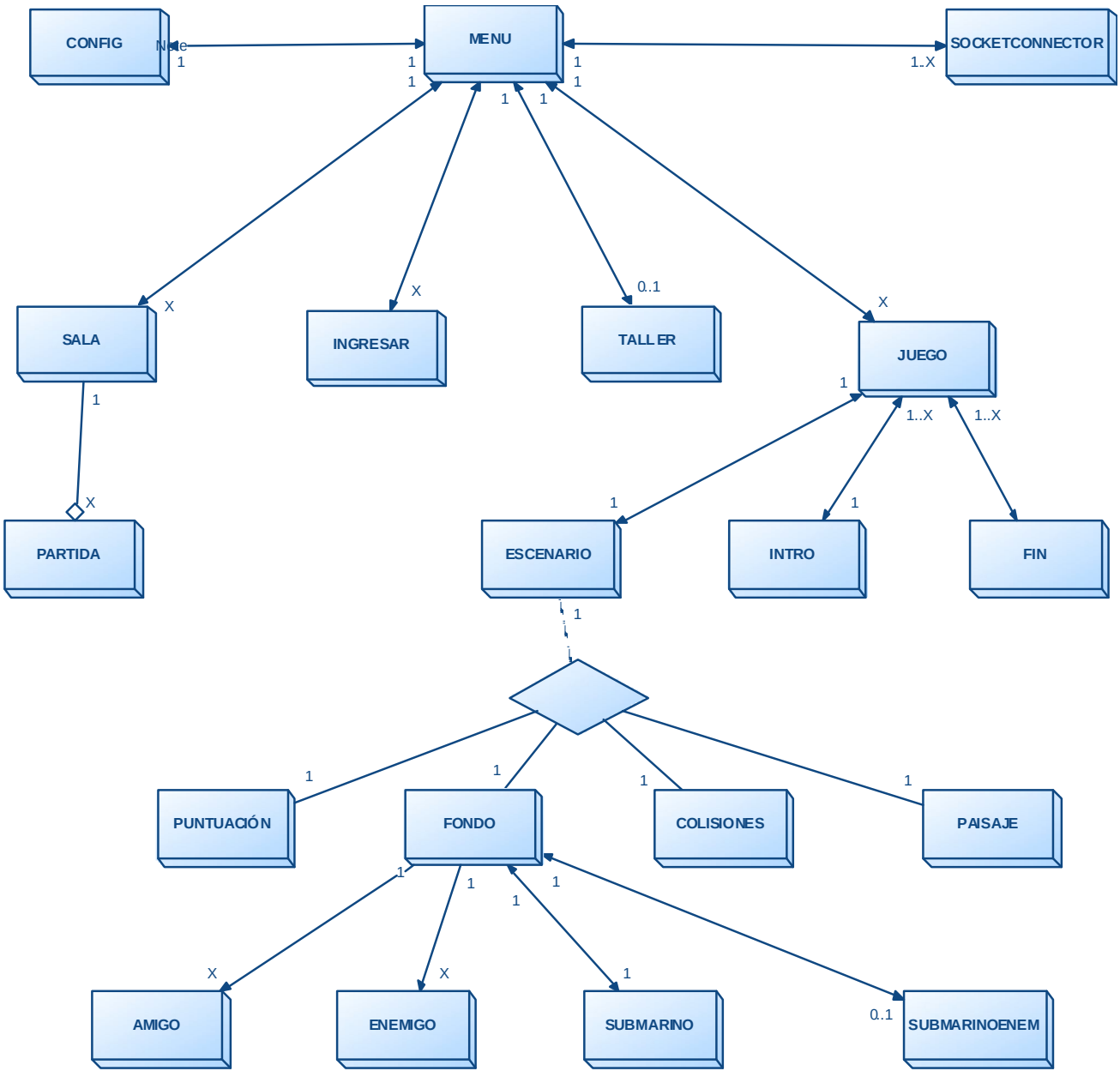
El siguiente diagrama visualiza los casos en los que se puede encontrar un usuario, tanto registrado, como no registrado. Esto ayuda a ver cómo empezar a crear el juego y la sala.




Como se puede observar, el usuario registrado tiene todas las mismas opciones que el usuario sin registrar. Pues como se indica hereda todos sus casos, pero además incluye guardar y cargar, puesto que tendrá el usuario en la base de datos.

## 4.3 Diagrama de clases

Antes de pasar a hacer el juego, tenemos que pensar cómo queremos que funcione todo en conjunto. Por este motivo se tiene que crear un diagrama con las clases que aparecerán. Gracias a este diagrama se hace más fácil la creación del juego, la sala y sus elementos.




Tras ver este esquema, queda claro que todo el control del juego pasará por la clase *Menú.* Separando de este modo en dos grupos importantes nuestro esquema. Por un lado estará la *Sala* y por el otro el *Juego*.

# CAPÍTULO 5: DISEÑO

La siguiente parte expone el diseño que tendrá el juego, tanto los aspectos visuales como el diseño de las clases.

## 5.1 Vistas

En el apartado gráfico tenemos que tener en cuenta las vistas que tendrán nuestras clases y el modo de interaccionar e introducir datos en éstas.
Cuando se usa Flash, que será el entorno que hemos elegido, las imágenes que se usan toman el nombre de *MovieClips* y se trata de imágenes vectoriales que podemos crear o bien con el programa "*Adobe Ilustrator*" o bien con el propio Flash.

### 5.1.2 Entrar usuario

Esta vista es muy simple; se trata de la primera pantalla del juego, y tenemos dos opciones: o entrar como usuario registrado o bien como invitado, por lo tanto, únicamente necesita de dos campos de texto donde introducir usuario y contraseña y dos botones, uno para aceptar datos y el otro para entrar como invitado.

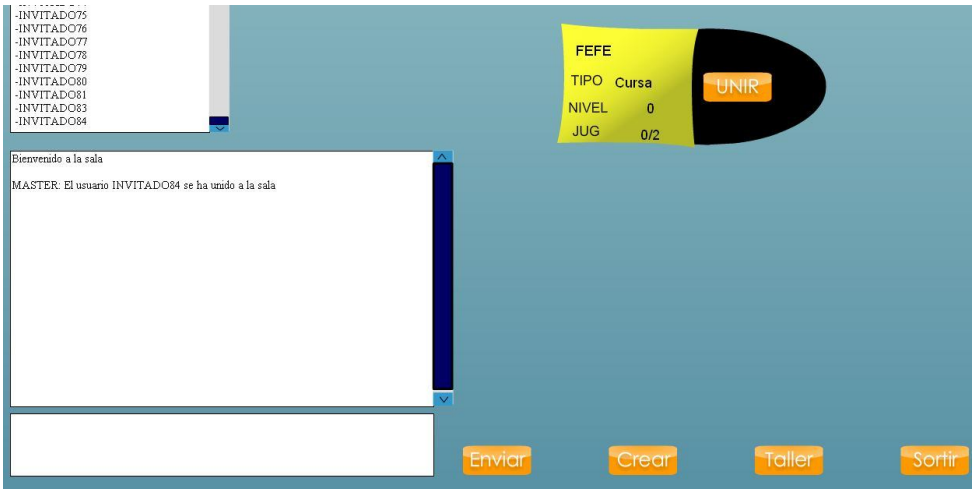

Vista del registro de usuario

### 5.1.2 Sala

La sala es donde el usuario se desenvolverá para llevar a cabo acciones relacionadas con la comunicación con otros jugadores, interacción con las partidas, edición de submarino y salir.
Para poder visualizar toda esta cantidad de elementos es necesario plantear una buena distribución y, de este modo, no dar sensación de estar demasiado lleno y, que a la vez, sea sencillo desenvolverse.

Por tanto, se tienen en cuenta tres campos de texto: para la lista de usuarios, el texto donde escribir los comentarios de todos los usuarios y del servidor y, el texto que pretendemos enviar con su botón de

*enviar*. Luego necesitamos un espacio donde poner la lista de partidas creadas desplazable con barra deslizante. De este modo, pueden haber cuantas queramos sin ocupar más espacio de lo deseado. Finalmente, se requieren botones para todas las opciones que hay.




Vista de la sala con una partida creada

### 5.1.3 Crear partida

Para crear partida sólo se necesita un campo para poner el nombre y un *combo box* para seleccionar el nivel.

### 5.1.4 Taller

Cuando entremos al taller únicamente podremos seleccionar el submarino que queramos y, si queremos mejorar, pintar el submarino o salir del taller; todo esto será controlado mediante botones.

En la edición de submarino aparecerá una pantalla con todos los elementos que se pueden mejorar y para seleccionarlos se tendrá que clicar encima. A la vez, se visualizará la cantidad de puntos que disponemos, el coste de las mejoras que queremos realizar y el saldo que nos quedará tras esta compra. Para confirmar la compra, cancelarla o salir se destinarán distintos botones, uno para cada acción.

Por último, para pintar el submarino veremos una pantalla simple con el submarino que tenemos seleccionado y unas flechas para que, al clicar encima vaya cambiando el color del submarino. Cada par de flechas hará referencia a una parte del submarino diferente (cabeza, cola y cuerpo).

## Juego

En el juego cabe diferenciar dos partes importantes. Una, es la parte de la cabecera, donde alojar la vida, oxigeno, puntuación, tipo de partida y tiempo.

Luego tenemos la pantalla del juego, donde transcurrirá la carrera. En este sitio tendremos el submarino controlado y el del contrario, en el mapa de la pantalla.


Vista de una partida en juego,

## 5.2 Clases

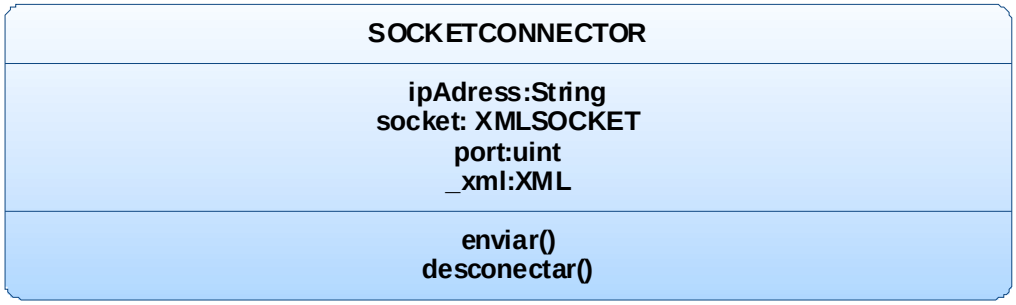
### 5.2.1 Menú


Esta es la clase principal, la que controla todo el contenido y gestiona su aparición, capas y conexiones.
Lo primero que tiene que hacer es crear un objeto de configuración que contenga todos los atributos necesarios, tanto a nivel de juego, como de idioma y estados del jugador. Acto seguido, establece la conexión con el servidor para acceder como jugador registrado o invitado a la sala y muestra al usuario el contenido donde acceder.

En el momento en que se ha ingresado como usuario correcto, Menú es informado, quita la pantalla de petición de usuario y desconecta la anterior conexión establecida y se dirige al servidor de la sala, creando un objeto *Sala* que retoma el control.

Cuando el jugador decida iniciar una partida, *Menú* será informado desde *Sala* y, de este modo, cerrará la *Sala* y su conexión, conectará con el puerto en el que esté el servidor de la partida e iniciará el Juego. Una vez finalizada la partida volverá a cambiar la conexión a la anterior de la *Sala* y volverá a mostrarla manteniendo el nombre de usuario.

### 5.2.2 SocketConnector



Las conexiones al servidor tienen que ser totalmente transparentes para cada una de las otras clases. Por eso, esta clase, establece la conexión correspondiente con la entrada de una dirección IP y un puerto.

Ofrece el método para poder enviar texto al servidor, un texto que ya vendrá con el formato dado y avisará de la llegada de datos desde el servidor.

### 5.2.3 Config

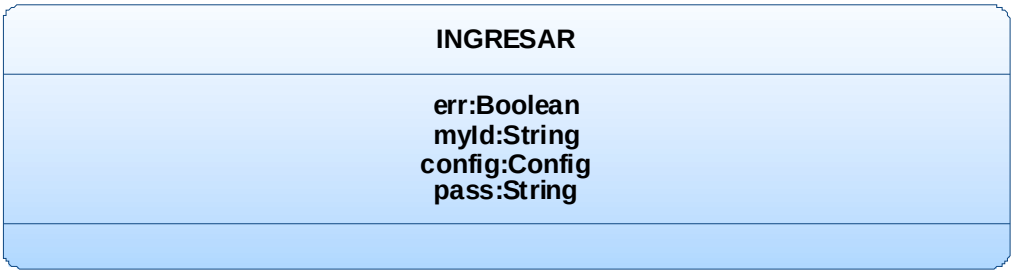

En *Config* es donde se almacenan todos los atributos del submarino, los del jugador al ser identificado y el estado en que se encuentre, así como el nivel que quiera jugar y el idioma que se usará juntamente con los textos que se mostrarán.

También se encarga de generar una cadena de texto a partir de sí mismo para poder almacenarlo en la base de datos en el caso de ser un jugador registrado. Del mismo modo, tiene que poder leer esta cadena de texto procedente de la base de datos para recuperar todos los datos almacenados.

### 5.2.4 Taller


El taller gestiona que queramos cambiar el submarino, cambiando el atributo en la configuración, tras detectar que se ha tocado el botón correspondiente.
También se encarga de cambiar las vistas para pintar y mejorar el submarino. En ambas, detectará las acciones del usuario, mejorando el submarino y pintando, y lo actualizará en la configuración.

### 5.2.5 Ingresar




El juego contempla dos tipos de usuarios, los registrados y los invitados. Con esta clase se gestiona que tipo de usuario se pretende usar. Esta clase establece la comunicación con el servidor y carga de partida salvada en caso de querer entrar como usuario registrado. Si por contra lo que se pretende es entrar como invitado, directamente envía a la sala sin datos previamente salvados.

### 5.2.6 Sala


Hacer carreras de submarinos contra otros jugadores es el objetivo final, y esto no sería posible sin un medio donde gestionar todo; aquí radica la importancia de esta clase. Por esta clase es donde se enviarán y recibirán los mensajes entre servidor y cliente, y ésta es la clase indicada para atender los mensajes recibidos y actuar en consecuencia.

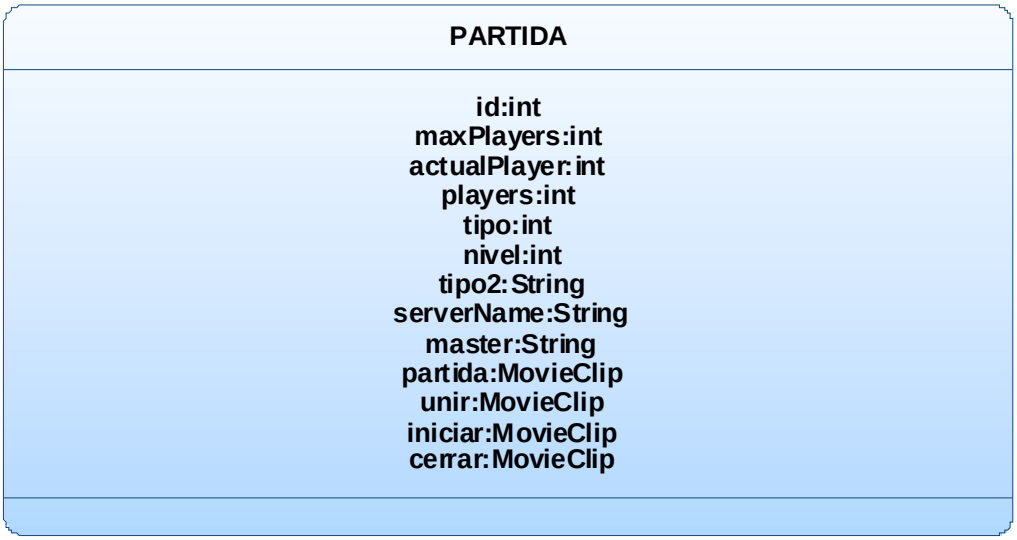
El primer punto será establecer la sala en un estado inicial, sabiendo todos los jugadores que hay y las partidas disponibles. Seguidamente, tiene que tomar control de el chat, tanto a la hora de enviar un mensaje nuestro, como de recibir y mostrar los de los otros jugadores o bien el servidor. Las partidas tienen muchos estados diferentes y alteraciones, es por esto, que tiene que estar en todo momento visible. Las acciones que se podrán hacer con las partidas serán las de crear, cerrar, iniciar y unirse a una partida.

Nosotros no seremos los únicos que podemos interactuar, por lo que se recibirán modificaciones de otros usuarios también.

Lo último que también se tendrá que informar al servidor es si queremos salir de la sala y, de este modo, salir de la partida en que podamos estar unidos.

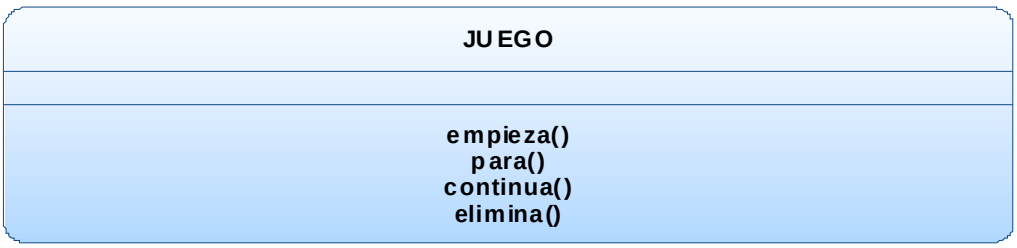
A parte de esto también se tiene que poder acceder a todas las opciones de las que se dispone como crear partidas, editar el submarino o salir desde un simple clic.

### 5.2.7 Partida




Cada partida que se cree será almacenada como una nueva clase de este tipo. Aquí tendremos toda la información necesaria de dicha clase, para poder gestionarse posteriormente desde la sala y así todos los usuarios tener los mismos datos.

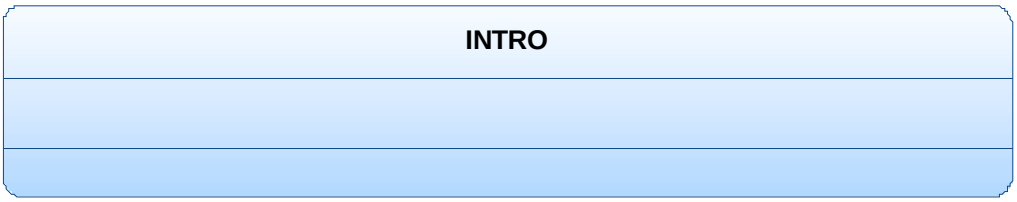
### 5.2.8 Juego



Ésta es la clase contendora de todo el juego, e independiente de la *Sala*. Desde aquí es desde donde controlaremos los estados básicos del juego tales como mostrar el juego, poner el control de la entrada al juego, y el fin de este, iniciar el juego y finalizarlo. Éstas son las labores que tiene el juego a nivel de un jugador. Sin embargo, al agregar la vertiente multijugador hay que añadir en esta clase toda la interpretación para luego actuar en consecuencia.

En esta faceta hemos de controlar todas las cadenas que lleguen del otro jugador, para saber que éste está listo, controlar el inicio de la carrera y el fin en caso de que alguno llegue a la meta o muera. Obviamente deberemos tener constancia de la posición del otro jugador en cada momento y en caso de ser anfitrión, el otro jugador también nos dirá el tiempo. Todos estos mensajes serán los que darán lugar a mostrar la entrada o el fin, iniciar la carrera o la cuenta atrás, y mover el submarino enemigo.

### 5.2.9 Intro

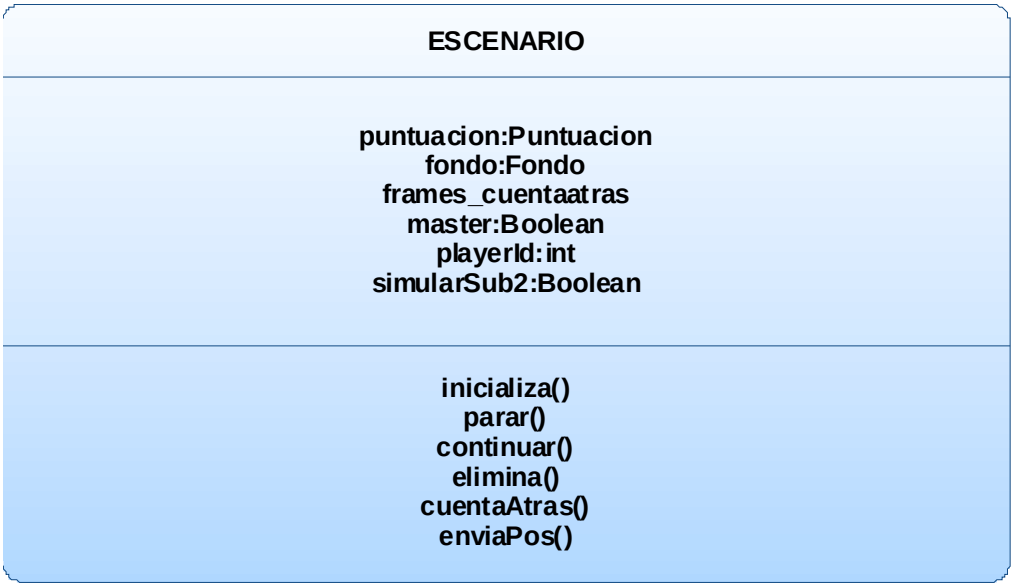



Controlado desde Juego muestra la pantalla de entrada para la espera de todos los jugadores de la partida. Desde aquí se mostrará un texto explicativo dependiendo de qué juego se trate, pues en esta versión multijugador únicamente encontramos el tipo de carreras, pero hay opción para más. En el momento que se le dé al botón para iniciar la partida se devolverá el control a Juego para proceder.

### 5.2.10 Fin


Una vez ha finalizado la carrera por cualquiera de los eventos posibles *Juego* mostrara la pastilla de final de partida, en esta se visualizará el estado finalizado, indicando si se ha ganado o se ha perdido. En caso de haber finalizado la carrera se indicará el tiempo conseguido, tanto por sectores, como la carrera en general y un ranking.

### 5.2.11 Escenario




Una vez se inicia la carrera, *Juego* cede el control a esta clase. Es desde *Escenario* donde se crean todos los contenidos del juego necesarios y se mueve el bucle principal del juego. Lo primero que debe hacer es crear todas las vistas necesarias, y añadirlas en orden para de este modo dividirlo bien por capas. De estas capas la primera que se creará es el paisaje, encima suyo se pondrá el fondo, donde ya se encuentran almacenados los submarinos y finalmente la cabecera donde se encuentran la puntuación y tiempos. La otra clase a crear pero sin parte grafica es la de colisiones donde se controlarán, como el nombre, indica las colisiones.

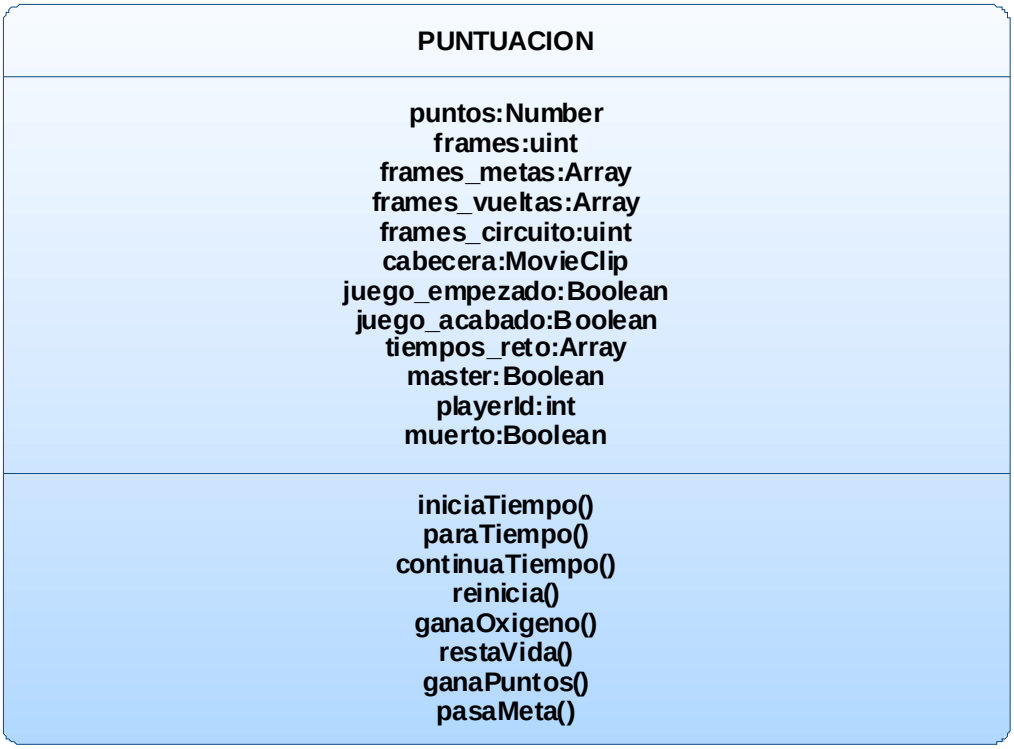
Cuando hemos añadido todo debe servir unas funciones públicas básicas para el control de estados. El motivo es que el encargado de interpretar los mensajes del otro jugador es *Juego*. Por lo tanto, tenemos que dejar que pueda iniciar y parar la partida, así como otras funciones para actualizar tiempo y submarino.

Esta clase también debe controlar la música principal del juego para iniciarla y siempre que termine volver a iniciar.

El bucle principal controlará todo los movimientos, así pues, desde aquí llamaremos a todas las funciones necesarias para el movimiento del escenario y detectar colisiones. Para el correcto funcionamiento, tendremos que hacer avanzar tanto al juego como al fondo y seguidamente mirar las colisiones pertinentes.

La última acción a llevar a cabo es la de enviar nuestra posición al servidor.

### 5.2.12 Puntuación

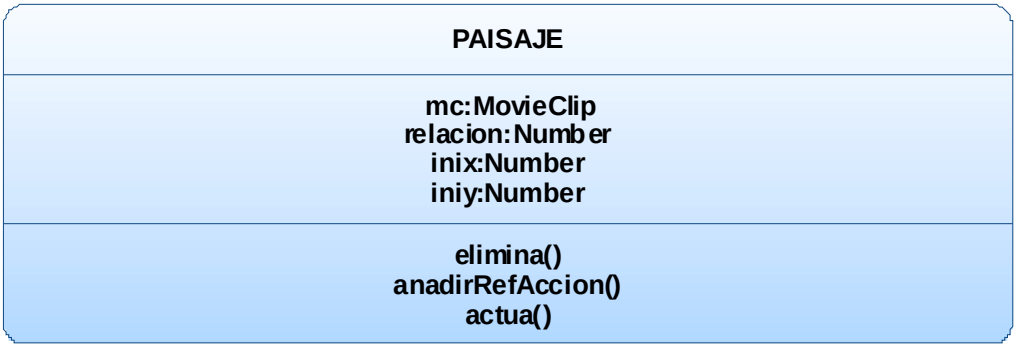



Desde esta clase se añadirá la cabecera, donde están todos los campos técnicos del juego, entendiendo como técnicos, la vida, oxigeno y tiempo.

También se controlarán los segundos que vayan pasando y actualizando ese tiempo. Aunque iniciar y parar el tiempo será una elección de *Escenario* y por lo tanto puntuación brinda sendas funciones para iniciar y parar la cuenta del tiempo.

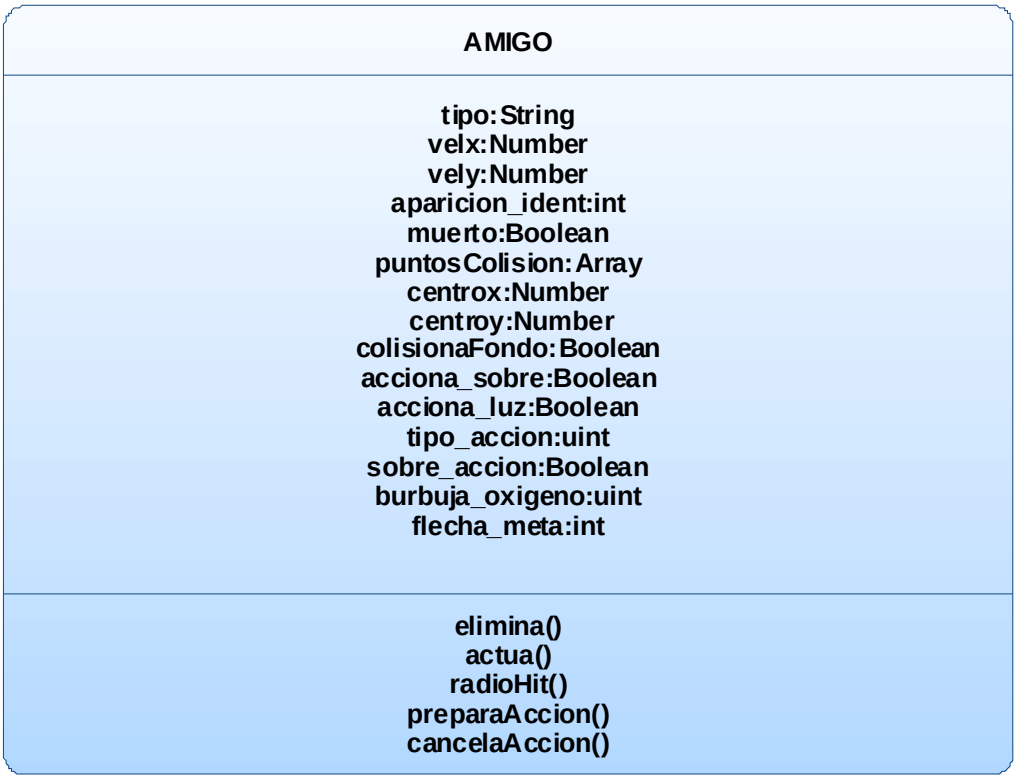
El último cometido de esta clase es llevar el control de todos los eventos generados durante la carrera desde diferentes puntos, bien sea perdida de vida al chocar, ganar oxigeno al salir al exterior, ganar puntos, o pasar las diferentes metas o puntos de control que se encuentran en la carrera.

### 5.2.13 Paisaje



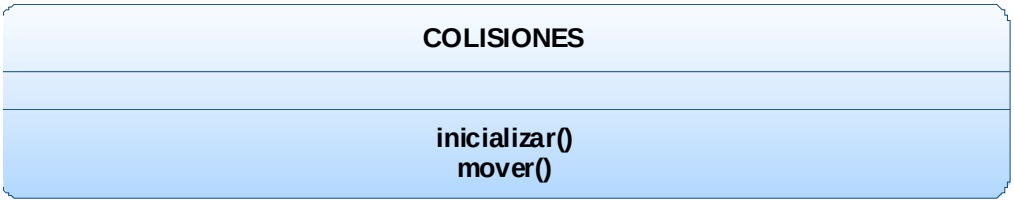
Durante la carrera hay un recorrido a seguir, pero tras este recorrido se encuentra un paisaje con el que no se entra en contacto y es meramente visual. Con el objetivo de dar más vivacidad al juego dicho paisaje se moverá también. Pero este movimiento lo controlará, como todos, colisiones y será dado en relación al movimiento con el fondo.

### 5.2.14 Amigo




En el juego no sólo encontramos submarinos, también hay otros elementos con los que interaccionar. En esta clase se encuentran únicamente que son beneficiosos o de ayuda. Hasta este punto en este juego únicamente encontramos unas flechas guías. La idea es que con esta sola clase se puedan implementar todo tipo de elementos en un futuro, o en diferentes juegos. La única acción que tiene que desenvolver esta clase es informar del comportamiento que tiene cada *Amigo*. Ya sea mediante una animación o con movimiento por medio de la clase *Colisiones* todos los amigos tendrán movimiento.

### 5.2.15 Colisiones



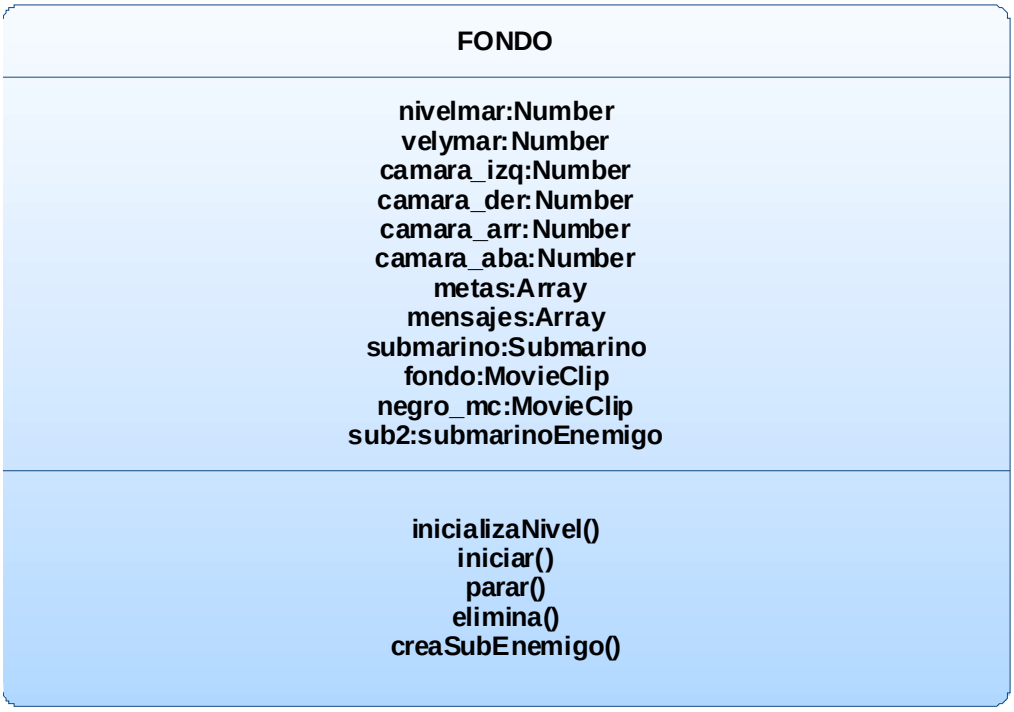
Las físicas son una de las partes más importantes de un juego, es precisamente esta clase la que se encarga tanto de moverlo todo, como realizar las interacciones de todos los elementos o colisiones.

Por lo tanto, lo primero que necesita al crearse es tener todos los elementos a mover, en este caso se trata del paisaje y el fondo, que contiene los dos submarinos. Pero estas funciones únicamente las llevará a cabo si desde *Escenario* es pedido, así pues, necesitará una función desde la cual se harán todos los movimientos para ese preciso momento y sus colisiones.

Todas las velocidades que se contemplan son contenidas y generadas en cada elemento, por lo que en cada petición, lo primero en hacer es mover el submarino, para ello, se consultan las velocidades en los ejes" x" e "y". Mediante una simple suma de posiciones se obtiene la posición temporal. Esta velocidad es temporal porque entonces se mira una posible de colisión y en caso de darse tal colisión se actúa en consecuencia. Hay que destacar que al tratarse de un mapa más grande que la pantalla para dar sensación de movimiento quien se moverá es el fondo, y el submarino estará en la misma posición. El submarino no se moverá excepto que lleve velocidades muy altas en las que se moverá acorde con su velocidad, perdiendo visibilidad, o si esta en un límite del mapa.
Acto seguido se mueve el paisaje en relación con el fondo.

Finalmente el submarino tiene un haz de luz para dar más sensación de profundidad, este haz se mueve desde aquí y será en relación a la posición del submarino y sus velocidades. Además cambiará su intensidad dependiendo la profundidad.

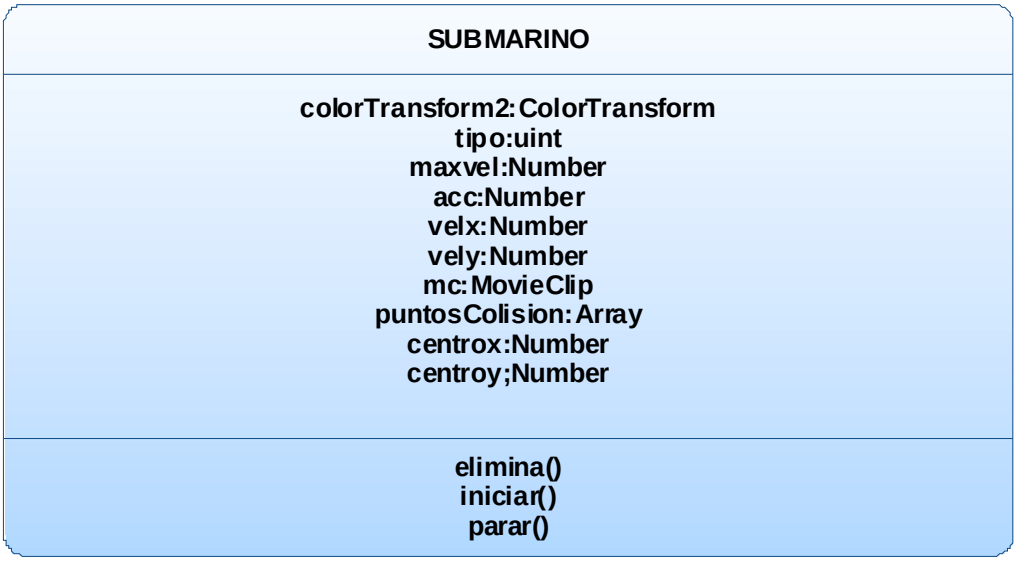
### 5.2.16 Fondo



En esta clase es donde se encuentra casi toda la parte visual del juego. Desde aquí se crea todo el fondo que conforma el circuito, para llevar esto a cabo, el circuito está dividido gráficamente en partes para de este modo ser más eficiente y poder reutilizaras en el diseño de diferentes circuitos. Por lo tanto dinámicamente se genera el circuito poniendo las partes en las posiciones ideadas, y dependiendo de qué carrera se quiere se genera el elegido.

Desde aquí también se crean tanto nuestro submarino como el submarino enemigo y se disponen para poder ser accedidos y modificar sus posiciones y la de dicho fondo desde el escenario.

### 5.2.17 Submarino

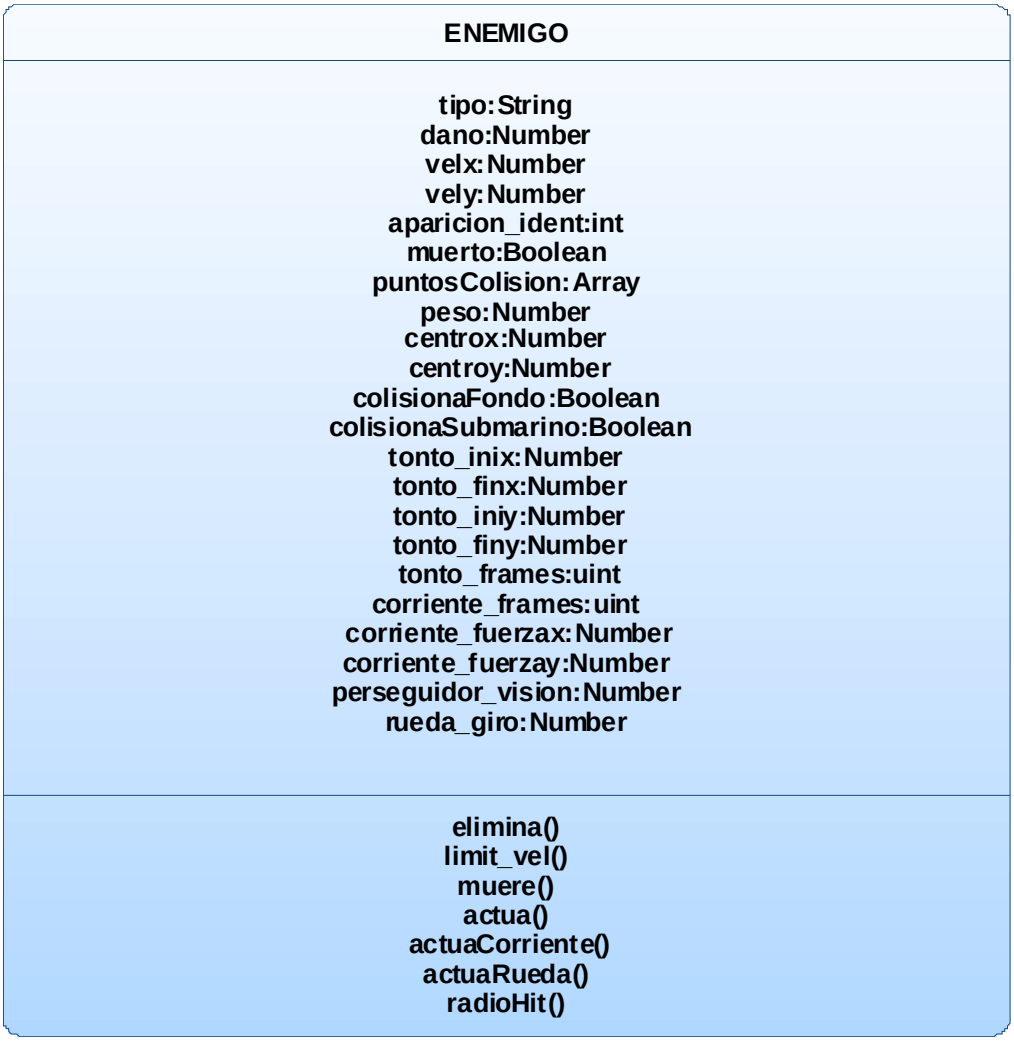



El submarino como clase tiene dos finalidades bien diferenciadas, por un lado la parte grafica del submarino y por otro lado la parte de control.

En cuanto a la parte grafica se contiene la imagen del submarino elegido y almacenado en la configuración y con esta todos los colores de las 3 diferentes partes del submarino elegidas en el taller.

Con referencia a la parte de control es en esta clase donde se detectan los eventos del teclado con los que se controla la velocidad que lleva el submarino, que aumenta con una progresión lineal. Todas las características están almacenada en la configuración, la aceleración y la velocidad máxima de la que no puede pasar.

### 5.2.18 Enemigo

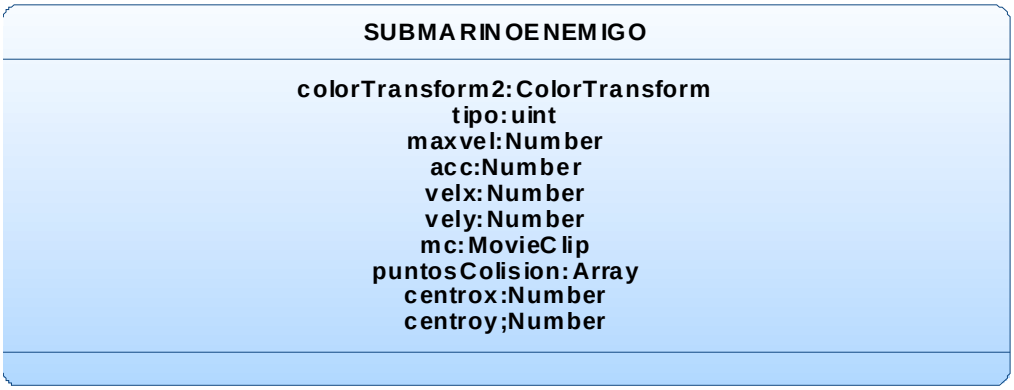



A diferencia de la clase *Amigo*, en esta clase encontramos los elementos que pueden ser negativos para nosotros. En este momento contiene las corrientes marinas, y las ruedas.

Aquí se contiene el tipo de acción que desenvuelve cada *Enemigo*. Para las ruedas generara el movimiento circular. En el caso de la corriente contiene las variaciones de velocidad que produce sobre el submarino. Dicho cambio será ejecutado en *Colisiones*

En esta clase se podrá añadir cualquier otro enemigo que queramos añadir, únicamente se tendrán que integrar sus métodos particulares.

### 5.2.19 SubmarinoEnemigo




A diferencia de la clase Submarino en este caso únicamente se contiene la parte grafica del submarino enemigo pues cuando se reciben el tipo de enemigo y su color se integran.

# CAPÍTULO 6: IMPLEMENTACIÓN

Tras diseñar todas las clases llega el momento de implementarlo todo. El primer paso es explicar qué arquitectura se va a usar, qué ventajas e inconvenientes tiene y qué posibles alternativas hay. Y después explicar cómo llevarlo a cabo.

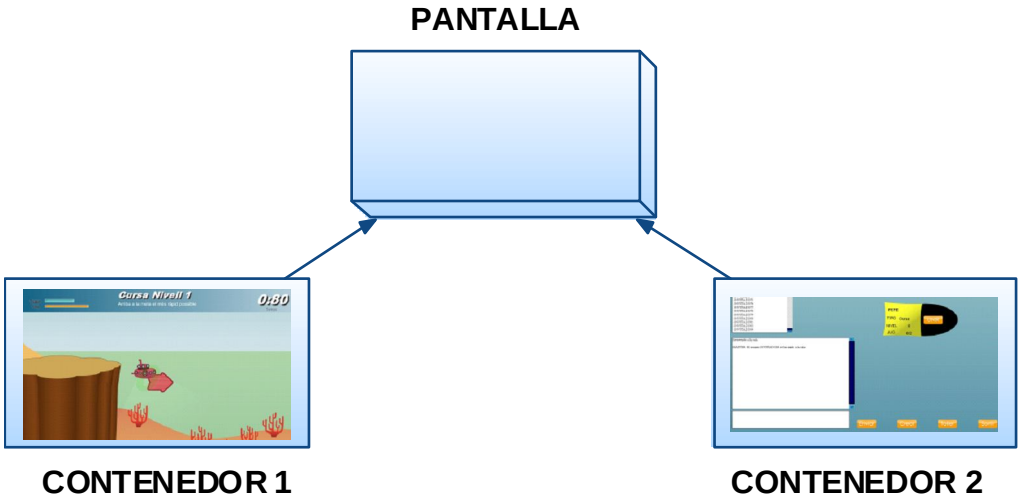
## 6.1 ActionScript 3

Este apartado explica cómo se han implementado algunos aspectos del juego y ciertas particularidades de ActionScript3 y Flash.

### 6.1.1 Capas

El apartado gráfico tiene que estar lo mejor estructurado posible. Para llevar a cabo este cometido, se ha seguido un una estructuración por capas. La primera idea es crear un contenedor que se añade sobre la pantalla raíz, y a partir de aquí, todo va sobre este contenedor. En caso de cambiar de entorno, como puede ser de la sala al juego, el cometido es quitar el primer contendor de pantalla y crear otro nuevo donde poner el juego. De este modo, siguen estando presentes ambos pero uno no necesita interacción gráfica.

El segundo concepto es crear cada clase como elemento gráfico y añadirlo a este contendor, por lo que si vamos siguiendo un orden, por niveles, visualmente conseguimos la misma finalidad, pero cada clase está en un nivel diferente y si se quiere eliminar uno de ellos es de fácil acceso.




Ejemplo dos contenedores, para poner en pantalla.

### 6.1.2 MovieClips

Una de las particularidades de Flash son las animaciones, no generadas por código. Estos elementos pertenecen a la clase MovieClip. Dado que pretendemos estructurarlo todo por capas, el contenedor antes citado será un MovieClip donde se agregará todo el resto.

Y todas las clases que aportan parte gráfica, necesitamos tratarlas como MovieClips. Por este motivo, las extendemos a MovieClip y así conseguimos heredar todos los atributos y métodos y poder tratarlo como si de un MovieClip se tratara.

Una vez queda claro que es un MovieClip, hay que saber que se puede hacer con él. Lo primero que se tiene que hacer es añadirlo a la pantalla, o bien sobre otro MovieClip. Esto lo conseguimos con el método *addChild*:

```
pantalla.addChild(objeto_a_agregar);
```

Con esto hemos conseguido añadirlo allí donde queramos, en este caso en la pantalla. Ahora ya podemos interactuar con él como queramos. De las acciones más comunes que se llevan son las de reproducir, para reproducir si tiene una animación y moverlo. Para moverlo, el MovieClip tiene unas posiciones X e Y, únicamente hay que modificarlas.

Finalmente hay que aclarar que Flash tiene una librería en la que se almacenan todos los gráficos que queremos usar en el juego. El modo de usarlos es como si de un MovieClip se tratará, simplemente se tiene que declarar con el nombre que tiene y ya podemos operar como con cualquier otro MovieClip.

De este modo, conseguimos crear un submarino, un muro, o cualquier cosa y añadirlo a nuestro juego.

### 6.1.3 Socket

Nuestro juego requiere una alta transferencia de datos y sincronización con el servidor, y esto sólo es posible con sockets. En nuestro caso particular, para la comunicación tanto con el servidor, cuando estamos en la sala, o con otro jugador, en una carrera, está hecha en XML. Gracias a esto, podemos usar la clase *XMLSocket* de Flash. La peculiaridad de este tipo de socket es que su mensaje siempre es un XML, por lo que nos va perfecto para nuestro propósito.

Una vez encontrado como conectar con el servidor, se intenta aislar lo máximo posible la conexión del resto del código. Por ese motivo, he creado la clase, conocida como *SocketConnector*; el cometido de esta clase es conectarse a una dirección IP y un puerto que le son asignados durante la creación. De esta manera, esta clase se encarga de establecer la conexión, brindar una función con la que enviar los mensajes desde otras clases y avisar siempre que se recibe un mensaje de entrada. La forma de avisar la miraremos más delante; de este modo siempre que se recibe un mensaje la clase encargada lo recoge.

Finalmente, con esta clase conseguimos hacer una conexión y poder enviar y recibir los datos desde nuestra sala o juego fácilmente.

### 6.1.4 Eventos

La forma de enterarse en Flash que se ha dado lugar un suceso es mediante los eventos, o también conocido en informática como interrupciones.

Los eventos de Flash son detectados en *frame* y son servidos en el mismo orden que se han recibido, gracias a lo que podemos llevar a cabo todas las siguientes necesidades.

### 6.1.5 Comunicación entre clases

Para comunicarnos entre las clases bien, podemos acceder a ellas, pero para esto, lo primero de todo, tenemos que tener privilegios y tampoco es del todo seguro. Por tanto, aprovechamos los eventos para esta comunicación.

A un objeto se le puede añadir una escucha de evento con el nombre que queramos. Tanto da si es uno de los propios de Flash como si la creamos nosotros.

```
objeto.addEventListener("Evento Especial" ,FuncionEspecial);
```

Tras esta sentencia, cuando se detecte el evento *"Evento Especial"*, ejecutará la *FuncionEspecial*. Para lanzar el evento simplemente se tiene que hacer.

```
objeto.dispatchEvent(new Event("Evento Especial"));
```

Con esto podemos comunicar sucesos entre clases, como en el caso del *socketConnector* avisar que está listo el mensaje recibido, o comunicar con el menú para cambio entre *Sala* y *Juego* o viceversa, entre más sitios donde se comunican. También se pueden enviar objetos de todo tipo en estos eventos.

### 6.1.6 Cargar y guardar

Aprovechando la integración que tiene XML en Flash el formato que tienen los datos de la partida guardada están en XML. Se trata de una cadena de caracteres, generado por la clase *Config* . Aquí es donde están almacenadas todas las características que tiene nuestro submarino, tales como, colores, puntuación, y valores técnicos. Esta cadena se genera siempre que vamos a guardar la partida, y se envía al servidor la tira XML. El servidor la almacena en la base de datos, y siempre que un jugador registrado se conecta le envía esta cadena tal como la dejó el servidor. Una vez recibido por el cliente lo único que tiene que hacer es recuperar la configuración tal como la tenía anteriormente. Para lograr esto la clase *Config* también dispone de la función de pasar esta cadena a las características del submarino.

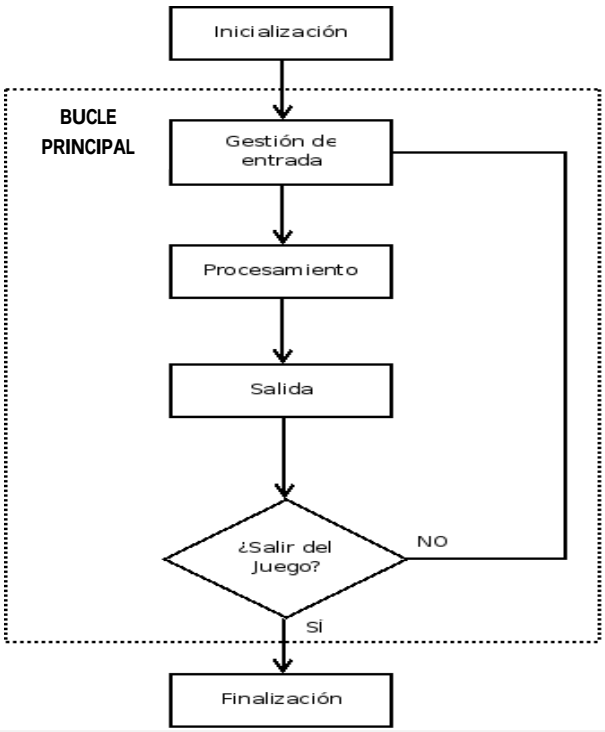
```
<config valido="s i" color="X" punt_total="punt_total" punt_actual=" punt_actual" valores="V" />
```

### 6.1.7 Bucle principal

El bucle principal del juego también es un evento. Concretamente, se trata de un evento de Flash, que se detecta cada vez que hay un nuevo *frame*, por lo que cada vez que entre en un *frame* se ejecutará la sentencia contenida en la función citada.

Escuchar este tipo de evento es idéntico que en el primer caso, únicamente cambia el nombre del evento pues ya está asignado por Flash. La diferencia radica en la forma de dispararse ya que al ser un evento propio de Flash es lanzado automáticamente en cuanto se entra en un nuevo *frame*.

El lugar donde se encuentra este bucle, es en la clase *Escenario.* De este modo, desde esta clase cada *frame* llamamos las funciones de todas las clases que contienen, tanto el *Fondo* y *Submarino*, para moverse como llamar a *Colisiones* para que detecte si hay colisiones.




Esquema de funcionamiento del bucle principal

### 6.1.8 Teclado y ratón

En cualquier juego, y de hecho, en cualquier aplicación lo más importante de todo es la interacción con el usuario, ya que es esta parte la que le da dinamismo. De lo contrario nos encontraríamos ante una película.

Para el caso de mover el submarino, al tener control por teclado tenemos que mirar los eventos de teclado. Las detecciones son como en el caso del bucle principal, con la diferencia del nombre del evento, y el evento también es disparado cuando sucede dicho evento sin necesidad de añadir código. Para los eventos de teclado podemos detectar tanto cuando se ha presionado una tecla, como cuando se suelta como si esta pulsada y qué tecla es la que está causando dicho evento. Gracias a esto, moveremos el submarino.

Obviamente, también necesitamos hacer clics con el ratón. Dichos clics son detectados también como eventos, estos se le asignarán al objeto donde queremos detectar el clic y, en cuanto se detecte dicho evento, se hará la función deseada.

### 6.1.9 Control del submarino

Una vez sabemos detectar la interacción con el usuario, tenemos que plantearnos como realizar los movimientos de los submarinos. Para este caso, podemos pensar en la física simple para llevar a cabo dichos movimientos, que son el movimiento rectilíneo uniforme y el movimiento rectilíneo uniformemente acelerado.

En el primer tipo de movimiento, cada vez que se detectase el evento de teclado se incrementaría la posición de un valor predeterminado, lo que equivaldría a su velocidad. En el segundo, cada vez que fuera detectado, sería la velocidad la que incrementaría dicho valor predeterminado. En este caso la aceleración, asignando una velocidad máxima, y acto seguido, se sumaría dicha velocidad a la posición que afecte.

Evidentemente la mejor solución que genera grandes resultados es el movimiento rectilíneo uniformemente acelerado, dando más realismo al movimiento, porque, mientras uno mueve siempre la misma distancia, el otro depende de su velocidad y se puede configurar perfectamente la aceleración y velocidad máxima para los diferentes submarinos, haciéndolos con diferentes características, tal vez uno muy rápido pero con poca aceleración, o viceversa. Como es de esperar, si se tuviera detección de presión en las teclas podríamos hacer una aceleración variable, lo que sería perfecto, pero no estamos en este caso pues únicamente se detecta si se ha pulsado o no la tecla.

Ahora que sabemos cómo actuaremos, tenemos que pensar de qué modo nos moveremos, ya que al ser un juego en dos dimensiones tendremos dos ejes sobre los que movernos, "eje x" y "eje y". En definitiva, si se detectan pulsaciones en las flechas de arriba o abajo se modificará la velocidad en el "eje y" y si se detecta a izquierda o derecha las modificaciones serán para el “eje x”, por lo que necesitaremos dos velocidades diferentes para cada uno de los ejes. Y siempre en caso de que no se detecte ninguna presión de tecla la velocidad se irá modificando hasta llegar a 0 para volver a la situación de reposo.

Una vez hemos conseguido las velocidades, simplemente hemos de aplicarlas a las posiciones para mover el submarino, puesto que nuestra velocidad siempre variará entre negativo para ir hacia atrás o positivo hacia adelante.

### 6.1.10 Movimiento del submarino

Si bien es cierto que las velocidades  son del submarino, las sumas a las posiciones no son asignadas a la del submarino, y esto tiene una fácil explicación. Al tratarse de un juego de carreras en el que el mapa es más grande que la pantalla, lo que se mueve no tiene que ser el submarino, pues en tal caso lo perderíamos de vista recorriendo el circuito. Por lo tanto, quien se mueve es el mundo.

Por ello, para dar realismo al movimiento lo que se hace es restar las velocidades del submarino al mundo. En tal caso, da la sensación de irse moviendo el submarino por el escenario.

A pesar de esta máxima, hay excepciones, pues cuando el submarino llega a un límite del escenario es el submarino el que toma el control y se mueve libremente o si el submarino va a velocidad máxima deja de estar centrado para desplazarse un poco dando sensación de velocidad.

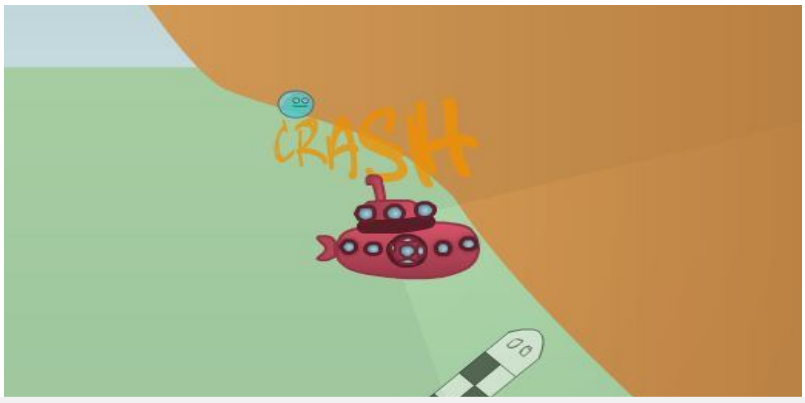
### 6.1.11 Colisiones

En todos los juegos en los que intervienen físicas las partes más complicadas son siempre las colisiones. Pues se tienen que conseguir dos objetivos simples de ver pero no tan evidentes en la programación. Primeramente, tenemos que detectar la colisión y, justamente en el momento, ya que más tarde ya tal vez sea demasiado tarde y además tenemos que actuar en consecuencia y que quede lo más real posible la colisión.

Para detectar la colisión, lo primero que observamos es que con la clase básica de Flash la detección no es del todo precisa pues no detecta sobre el contorno del objeto sino en un recuadro imaginario, por lo que al tener submarinos de formas irregulares esta detección no nos sirve.

Lo que sí que se puede encontrar es la colisión por puntos. Por lo tanto, la estrategia que se sigue es poner unos puntos invisibles en el contorno de los submarinos y se detecta la colisión en estos puntos. Sabiendo el punto donde colisiona, podemos conseguir la dirección del choque.

Una vez tenemos detectada la colisión, hemos de actuar en consecuencia. Esta es la parte donde se corre mayor peligro,  pues un fallo nos puede dejar el submarino en un punto de no retorno colisionando con todos los elementos.




Colisión del submarino con el fondo

Una vez detectado dónde es la colisión y la dirección de este choque, únicamente hemos de cambiar las velocidades en “x “e “y” en relación a la dirección antes mencionada, y actuar restando la vida necesaria.

En caso de no haber colisión se moverá sin ningún problema.

### 6.1.12 Multijugador

En la comunicación entre jugadores o cliente servidor, como ya he comentado, se envían mensajes XML, por lo tanto el orden de actuar para tratarlos es el siguiente tanto en la sala como en los juegos. Se recibe un evento del *socketConnector*, en ese momento se lee el archivo de llegada, que se trata de un XML como hemos comentado, y se procesa.

Para procesar dicho fichero, Flash ya tiene el objeto de tipo XML, con el que se puede acceder a los hijos y atributos de los campos, por lo tanto simplemente se tiene que mirar en unos condicionales qué tipo es el de la entrada y actuar.

No voy a explicar otra vez cada uno de los paquetes, simplemente cómo procede. Siempre que recibe un paquete tanto en *Sala* como en *Juego*, o bien actúa sobre la propia clase, o bien ejecuta la función adecuada en cada una de las clases que contienen, como por ejemplo en el caso del juego cuando recibe que los dos jugadores están listos, comienza la partida.

También hay pensado un método de anfitrión y esclavo. Esto se aplica tanto en *Sala* como en *Juego*, pero mientras en *Sala* el anfitrión es el servidor, y el que controla todos los estados, en *Juego* es diferente.

Cuando entramos al juego se nos son asignados unos nombres o identificaciones numéricos, comenzando en el 0, por lo que el número 0 es automáticamente el anfitrión. En este juego, su única función es la de controlar el tiempo. De este modo, sólo uno lleva el control de éste y se evitan conflictos, pero no únicamente controla esto, también es el encargado de controlar la cuenta atrás y la gestión de las esperas, ya que hasta que no estén todos los jugadores listos no empieza la carrera.

Otro motivo puede ser para controlar el escenario y sus enemigos, pero puesto que esto no atañe al tipo de carreras, no afecta, aunque se podría usar.

### 6.1.13 Control del submarino enemigo

Cada vez que se actualiza la posición del submarino, es enviada por el socket e inmediatamente actualizada en el contrario, pero puesto que se pueden producir errores, el juego contiene una predicción de movimiento.

Esta predicción es básicamente la aplicación del control de movimiento que tiene el submarino propio, sobre el del contrincante, ya que cuando actualiza posición no solamente envía la posición sino que también envía las velocidades. Por ello, se puede predecir con un menor margen de error la futura posición aunque a largo plazo no es una solución y sería totalmente erróneo, pero en el momento en que llegara el siguiente, automáticamente sería corregida la posición.

## 6.2 XML

Para podernos comunicar en la sala donde seleccionar el juego con el servidor o mientras estamos jugando la partida contra otro jugador, necesitamos establecer algún protocolo de comunicación entre ellos. Debido a esta necesidad, elegimos crear nuestro protocolo encapsulándolo en XML, pues de este modo tenemos todo estructurado con un lenguaje estándar.

Ya que tenemos dos medios diferentes en los que comunicarnos, uno es cliente-servidor y el otro es jugador-jugador, hemos de crear dos protocolos diferentes.

### 6.2.1 Cliente-Servidor

El propósito de este protocolo es proporcionar una comunicación basada en el formato XML bidireccional.

**Cliente**

Seguidamente prosigo a explicar los diferentes tipos de peticiones y mensajes encapsulados en XML que puede enviar el Cliente hacia el servidor, su composición y finalidad. Las respuestas que se reciban son explicadas en el apartado del servidor.

- Carga de datos:

```
<LOGIN USER="X" PASS="Y" />
```

Si el cliente quiere entrar como usuario registrado, y recuperar los datos guardados tiene que enviar este mensaje. Envía su nombre de usuario y contraseña, y confirma que existe en la base de datos.

-Guardar datos

```
<LOGOUT ID="X" PASS="Y" ESTADO="MSG  CONF" />
```

Como jugador registrado se permite guardar en la base de datos, para una futura recuperación de los datos. Con este propósito se envía el nombre de usuario *ID* y la contraseña *PASS.* La última parte del mensaje, *ESTADO* , contiene una cadena con los datos de la partida a guardar. Este mensaje es predefinido por el juego, independiente del servidor.

-Petición entrada a la sala:

```
<ENTRARSALA USER="X" />
```

El cliente hace una petición a la sala con el nombre de usuario X, el servidor nos contestará si está libre o no, y, en caso de serlo procederemos a entrar en la sala.

-Petición jugadores de la sala:

```
<LISTSALA ORDEN="0"/>
```

El cliente, al entrar en la sala, para saber los jugadores que hay en la sala en ese momento envía esta sentencia. El orden sirve para comunicar al servidor el tipo de orden con el que deseamos la respuesta.

-Petición listado de partidas:

```
<LISTPARTIDA ORDEN="0"/>
```

El cliente, al entrar en la sala, hace esta petición para saber la lista de partidas que hay, en ese momento, creadas. *ORDEN* es el orden en el que se espera la respuesta la partida, por defecto 0.

-Salir de la sala:

```
<SALIRSALA USER="X" IDPARTIDA="Y" />
```

Para que el servidor sea consciente de que queremos abandonar la sala hemos de enviarle este mensaje. Necesariamente debe contener el nombre de usuario, en este caso *X*, y para que pueda liberar la partida en la que estemos unidos para que otro jugador pueda ocupar nuestra partida, debemos enviar el campo *Y*, que variará de 0 al último número de partida en el caso de que estemos en una partida en ese momento, o -1 en caso de que no estemos unidos a ninguna partida.

-Enviar mensaje:

```
<MSG USER="X" CONTENIDO="Y"/>
```

MSG lo usa el usuario para enviar al servidor un mensaje que posteriormente aparezca en el chat de todos los jugadores. *X* pertenece al nombre del usuario que lo envía e *Y* es el contenido de dicho mensaje.

-Crear una partida:

```
<CREAR MAXPLAYERS="X" SERVERNAME="Y" MASTER="Z" TIPO="T"NIVEL="1"/>
```

Los usuarios tienen a su disposición crear partidas en el servidor. Para ello deben enviar el mensaje *CREAR*, seguido de los otros campos obligatorios. En el campo *MAXPLAYERS* tendremos que indicar el máximo de jugadores que puede haber en la partida en cuestión. En el campo *SERVERNAME* se espera el nombre que le damos a la partida, en este caso *Y*, en *MASTER* hace referencia al creador de la partida, y que evidentemente es el propio jugador por lo que envía su nombre. Finalmente los campos *TIPO* y *NIVEL* son para especificar qué tipo de partida pretendemos jugar, y el nivel de dicha partida.

-Cerrar una partida:

```
<CERRAR IDPARTIDA="Y" MASTER="X"/>
```

Las partidas únicamente pueden ser cerradas por el usuario que las creó, por lo que para cerrarla enviaremos el mensaje *CERRAR*, seguidos de los campos *IDPARTIDA* y *MASTER*, siendo el primero la partida y el segundo nuestro nombre. De este modo confirmará que sea el mismo que el que la creo.

-Unir a una partida:

```
<UNIR IDPARTIDA="1" IDPLAYER="Y"/>
```

Siempre que un usuario quiera unirse a una partida tendrá que enviar un mensaje como el anterior, siendo *IDPARTIDA* el nombre de dicha partida e *IDPLAYER* el nombre de este jugador.

-Dejar una partida:

```
<DESUNIR IDPARTIDA="1" IDPLAYER="Y"/>
```

Una vez un usuario esté unido a una partida, tiene la opción de abandonarla. Para ello tiene que enviar *DESUNIR*, cumplimentado por los campos *IDPARTIDA* e *IDPLAYER*, siendo el nombre de dicha partida el primero y el nombre del usuario el segundo, en el ejemplo *1* e *Y* respectivamente.

-Iniciar una partida:

```
<INICIAR IDPARTIDA="1" MASTER="Y"/>
```

En cualquier momento el cliente que creó una partida puede pedir iniciarla al servidor, enviando *INICIAR*. Para que el servidor sepa de qué partida se trata rellenaremos el campo *IDPARTIDA* con el nombre de dicha partida y el campo *MASTER* con nuestro nombre de usuario para que pueda confirmar que es quien la creo, pues es el único que la puede iniciar, en este caso *Y*.

**Servidor**

Una vez conocidas las peticiones que nos puede hacer el cliente y los mensajes que nos puede enviar, procedo a explicar las respuestas generadas por el servidor o mensajes que le envía al cliente.

-Respuesta cargar datos:

```
<LOGIN ERR="1" />
```

Siempre que un usuario intente cargar su partida de la base de datos, el servidor confirma si la identificación es correcta. En caso de no serlo responde con este mensaje de error. Si en cambio es correcta responde con los datos de la partida guardada. Estos datos fueron generados y enviados por el juego y el servidor no los modifica.

-Respuesta guardar datos:

```
<LOGOUT OK/>  <LOGOUT ERR="1"/>
```

Si se ha conseguido guardar correctamente en la base de datos, se envía el primer mensaje, en caso contrario se envía el mensaje de error.

-Respuesta para entrar en la sala:

```
<ENTRARSALA STATUS="X" />
```

Cuando el servidor recibe una petición para entrar en la sala, comprueba si el nombre enviado por el cliente está permitido o no y procede a contestar en consecuencia. El mensaje es el mismo, lo que se diferencia es en el estado, *X*, que puede variar entre *OK* en el caso de que se acepte la petición o *FAIL* en caso contrario.

-Respuesta a los jugadores de la sala:

```
<LISTSALA NUMERO="N" PLAYER1="X" PLAYER2="Y" .....PLAYERN="Z"/>
```

Cuando el servidor recibe una petición de los jugadores que hay una sala, consulta todos los jugadores activos, y genera una respuesta con el nombre *LISTSALA* con el campo *NUMERO* que contiene el número de jugadores, en este caso N, seguido de todos los nombres de dichos jugadores. Estos nombres vendrán dentro de un campo del estilo *PLAYER**K*** siendo K un valor comprendido entre 1-N, con N el máximo de jugadores, por lo que recibiremos *N* jugadores dentro de sus respectivos campos.

-Respuesta a partidas en sala:

```
<PARTIDAN ID="1" STATUS="0" MAXPLAYERS="16" ACTPLAYERS="3" SERVERNAME ="X"
MASTER="Y" SERVERHOST="Z" SERVERPORT="P" TIPO="T" NIVEL="0"/>
```

En el caso de que el cliente haya pedido la lista de las partidas de la sala, la respuesta del servidor serán tantos mensajes como partidas hayan creadas, cumplimentadas con la información de esta.

Este mensaje está compuesto por el nombre *PARTIDA* , con los campos *ID* para el identificador de la partida, *STATUS* el estado en el que se encuentra la partida, *MAXPLAYERS* el número máximo de dicha partida, *ACTPLAYERS* el número de jugadores que tiene en ese momento la partida, *SERVERNAME* el nombre que tiene esta partida, *MASTER* el creador de la partida, *SERVERHOST* y *SERVERPORT* son la dirección IP y el puerto en el que cuanto se inicie la partida se encontrará el servidor de esta, *TIPO* el tipo de partida que es y, finalmente, *NIVEL,* que es el nivel de dificultad de dicha partida.

-Respuesta de salir de la sala:

```
<SALIRSALA STATUS="X"/>
```

En el momento en que el cliente intenta abandonar la sala, el servidor contesta con este mensaje, siendo *X* la respuesta de si se ha podido llevar a cabo con un *OK* o en caso contrario *FAIL*.

-Mensaje a los usuarios:

```
<MSG USER="A" CONTENIDO="Y" PLAYER="X" ESTADO="1"/>
```

Este mensaje será enviado en caso de que un usuario quiera enviar algún mensaje al chat o bien para informar que un jugador ha entrado o abandonado la sala.

En el primer caso únicamente se envían los campos *USER* y *CONTENIDO*, con el nombre del usuario que lo ha enviado y el contenido del mensaje que quiere enviar.

En el caso de ser consecuencia de que un usuario ha entrado o abandonado la sala, el campo *USER* contiene el nombre registrado *MASTER* y un mensaje estándar para entrada o salida del usuario. En el campo *PLAYER* aparecerá el usuario que toma parte en el evento, y *ESTADO* contiene un valor de 0 o 1; el primer caso es si entra en la sala, y el segundo si la abandona.

-Respuesta a crear partida:

```
<CREAR STATUS="OK" ID="X" MAXPLAYERS="M" SERVERNAME="Y" MASTER="Z" TIPO="'T'"
VEL="N"/>
```

Una vez un usuario intenta crear una partida, servidor responderá con este mensaje, todos los campos corresponden a los enviados por el usuario, el único añadido es el campo *STATUS* donde se indica si se ha podido crear, con *OK* o en caso contrario con *FAIL*.

-Respuesta cerrar partida:

```
<CERRAR STATUS="Y" IDPARTIDA="x"/>
```

Responde esto cuando se intenta cerrar la partida *X* y devuelve *OK o FAIL* para indicar si se ha conseguido o no y avisar a todos los jugadores.

-Aviso de un usuario unido a partida:

```
<UNIR STATUS="X" ID="Y" MAXPLAYERS="M" ACTPLAYERS="A"SERVERNAME="Z"/>
```

El servidor envía este mensaje debidamente cumplimentado con los datos de la partida a todos los usuarios cuando un usuario intenta unirse, con el estado para decir si se ha conseguido y actualizando el nuevo número de jugadores.

-Aviso de un usuario deja la partida:

```
<DESUNIR STATUS="X" ID="Y" MAXPLAYERS="M" ACTPLAYERS="A"SERVERNAME="Z"/>
```

El servidor envía este mensaje debidamente cumplimentado con los datos de la partida a todos los usuarios cuando un usuario intenta dejar una partida, con el estado para decir si se ha conseguido y actualizando el nuevo número de jugadores.

-Iniciar una partida:

```
<INICIAR STATUS="OK" IDPARTIDA="x" SERVERHOST="x" SERVERPORT ="X"/>
```

Cuando el servidor quiere iniciar una partida envía este mensaje, bien por petición del usuario que la creo o por decisión propia. Sea como fuere, envía el estado, si se puede o no, la identificación de la partida, y la IP y puerto en el que se encuentra el servidor donde jugarla.

### 6.2.2 Jugador-Jugador

En este apartado se explica el protocolo que se usará en la comunicación entre jugador y jugador durante el juego, para llevar a cabo tanto el inicio de partida como los movimientos. En este caso, se puede diferenciar al jugador normal del anfitrión, puesto que este segundo contendrá diferentes mensajes, pero seguirá teniendo lo del jugador.

**Jugador**

Esta parte hace referencia a mensajes que pueden enviar todos los jugadores independientemente de que sea el anfitrión o no.

-Jugador listo y en espera:

```
<Waiting playerId="X" sub="Y"/>
```

Para confirmar que un jugador está listo para comenzar envía el mensaje *Waiting*. El primer campo es *playerId*, y en dicho campo se dice el identificador de este jugador. El segundo campo es *Sub* este campo es una cadena de texto que contiene toda la configuración del submarino enemigo; tanto que submarino es, como sus características, puesto que puede estar mejorado. También envía los colores de las partes del submarino para que aparezcan visualmente.

Tras recibir el mensaje, si no se trata de nuestra identificación, creará el submarino enemigo. Las características de este submarino vendrán dadas en el mensaje.

-Posición del submarino:

```
<subPos id="I"  posx="X" posy="Y" rotado="Z"  velx="VX" vely="VY" />
```

Para que el resto sepa dónde está nuestro submarino, hemos de enviar, cada vez que se mueve, el mensaje *subPos*. Este mensaje da todos los datos necesarios para que se pueda controlar nuestro submarino para el otro jugador. El primer campo necesario es el identificador, en este caso *I* .Seguidamente viene la posición en la que se encuentra nuestro submarino, tanto en el "eje X" como en el "eje Y", por lo tanto se lo enviamos en *posx* y *posy,* respectivamente. Otro valor importante para que se dé una buena visualización, es enviar *rotado*. Este campo contiene el ángulo de rotación de nuestro submarino, especialmente necesario para saber para qué lado mira el submarino. Finalmente vienen las velocidades, en ambos ejes también *velx* y *vely.* Estas son las velocidades que lleva el submarino en ese preciso momento.

Con estos datos podemos situar al submarino enemigo en la posición indicada, y las velocidades nos sirven para emular su movimiento en caso de que este mensaje se retrase al llegar.

-Final de partida:

```
<Fin id="I" tiempo= "T" muerto="M" acabado="A"  />
```

En cuanto un jugador finaliza la partida tiene que indicárselo al otro, pues éste será el fin del juego. Para tener un buen control de qué motivo ha llevado al final de partida y definir el ganador, se envía el identificador del jugador, y en el campo *tiempo* el momento en el que se ha producido el evento. Los otros dos campos que se envían son dos boléanos, para indicar si el jugador ha muerto, y el otro si ha finalizado la carrera.

Tras recibir este mensaje ambos jugadores pueden saber quién es el ganador de la partida y declarar un vencedor y un perdedor.

**Jugador anfitrión**

Este jugador contiene todos los mensajes del jugador normal, pero además añade otros de control, o los modifica para su propósito.

-Comenzar partida:

```
<Start />
```

El anfitrión es el encargado de ir contando cuántos jugadores están listos, mediante el mensaje *Waiting*. En cuanto estén todos listos envía este mensaje *Start*, que indica a todos jugadores que la partida puede comenzar.

-Cuenta atrás:

```
<Cuentaatras  frames="N"/>
```

Antes de que dé comienzo la carrera se produce una cuenta atrás. Esta cuenta atrás va de 3 a 0, y el encargado de llevar la cuenta es el anfitrión. Así pues, cada vez que aparece un numero envía el mensaje *Cuentaatras* cambiando el valor *N* por el momento en el que estamos, y en cuanto llega a 0 da comienzo la carrera.

-Posición del submarino:

```
<subPos id="I"  posx="X" posy="Y" rotado="Z"  velx="VX" vely="VY" segundos="S"/>
```

Igual que el resto de jugadores, el anfitrión envía el mensaje de la posición del submarino. Sin embargo, con la intención de evitar más envíos de la cuenta, para no sobrecargar la red, y puesto que es el encargado de llevar el control del tiempo, añade el campo *Segundos*. Este campo no se rellena con segundos exactamente, ya que contiene los segundos y las milésimas de segundos que llevan transcurridas.

## 6.3 Depuración de errores

Como siempre que se desarrolla un código considerablemente largo, aparecen errores. Estos errores pueden ser detectados por el compilador, en este caso flash, o bien dar errores en tiempo de ejecución. Este apartado está dedicado a exponer las experiencias vividas en el desarrollo de este proyecto, los errores encontrados y las armas para combatirlos.

En el caso de los errores en tiempo de compilación, éstos son detectados automáticamente por el compilador y avisan la línea y el error que se genera, por tanto sólo basta con ir allí para solucionarlo. Sin embargo, no es tan simple en caso de ser un error en tiempo de ejecución, puesto que estos tienen que buscarse manualmente dentro de la función que genera el error y encontrar el motivo, ya que, en este caso, es más genérico el comentario.

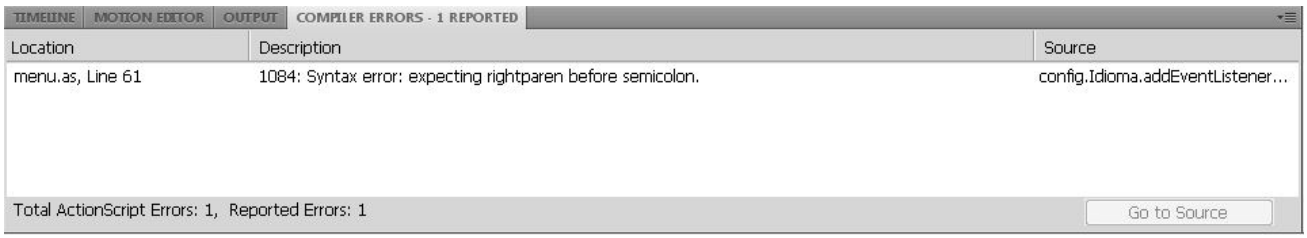
### 6.3.1 Errores en tiempo de compilación

Los errores en tiempo de compilación suelen ser los más sencillos de corregir. Al detectar el error flash te indica línea y código de error.

Estos errores son debidos al código que no es válido a la hora de compilar. Los errores más comunes que suelen darse de este tipo son:

- Algún corchete mal puesto, al poner mal el corchete no encuentra final de código.
- Errores de puntuación, ya sean un *punto y coma* mal puesto, una mala concatenación de caracteres o la falta de algún paréntesis en igualaciones u operaciones.
- Igualación de tipos diferentes, bien sea en un condicional igualando dos tipos que no se pueden comparar, como en una cadena de texto y un número, o asignándole a una variable un tipo diferente.

Todos estos errores son sencillos de corregir; simplemente hace falta percatarse. Flash indica la línea, por lo que se vuelve algo mecánico y, con el tiempo, este tipo de errores se minimizan.




Error detectado en tiempo de compilación

### 6.3.2 Errores en tiempo de ejecución

Una vez compilada la película flash, al iniciarla pueden darse errores que generen un mal funcionamiento del programa, o bien lo bloqueen. Éstos son los errores en tiempo de ejecución.

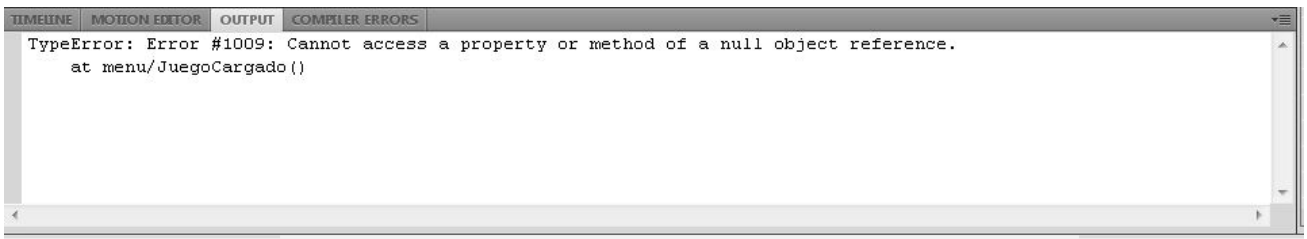
Para poder visualizar estos errores es necesario o bien ejecutarlo en Flash o bien con alguna consola de errores del navegador, siendo más común el primero.

En este caso, los errores nos indican en qué función se está dando y de dónde proviene ésta. A parte, nos indica el tipo de error con una breve descripción. Es por este motivo que encontrar el lugar del problema es más complicado que en los otros errores ya que existen más líneas donde buscar. Por ello, buscar este tipo de errores se convierte en un mecanismo manual en lugar de algo mecánico como era en el otro tipo de errores.

Estos errores son debidos a un mal funcionamiento en el tiempo de ejecución debido, normalmente, a un mal uso de una variable, ya que en la mayoría de los casos no está definida correctamente en el momento de su consulta. Esto lo podemos ver diversos casos:

- Consultando una variable no definida, como puede suceder en condicionales o asignaciones vacías (estos casos dan error).
- Consultando variables o clases mal construidas.
- Intentando enviar por un socket sin conexión. Este es uno de los errores que más se han dado este proyecto por su gran uso de la conexión a internet. Siempre que se produce un error de conexión y se intenta hacer algún envío por este socket da error al ser imposible llevar a cabo tal acción.
- Eliminando un objeto de una capa que no es de esa clase. Para una mejor distribución se crean en las clases unas capas que es donde, luego, se añaden los objetos que se ven por pantallas. Cuando se intenta eliminar de esta capa un elemento que no ha sido creado por la clase que intenta eliminar detecta un error porque únicamente puede quitarlo quien lo ha añadido.

El método de detectar estos errores se reduce a ir mirando los resultados de las variables mediante impresiones por pantalla para ver en qué momento tienen un valor incorrecto y poniendo avisos para encontrar la línea en que sucede.




Error detectado en tiempo de ejecución

# CAPÍTULO 7: MANUAL DEL JUEGO

## 7.1 Introducción

*Monturiol: el Juego* es un juego en red, multijugador, por internet en el que se puede competir contra otros jugadores en carreras. Lo puedes encontrar en ***www.monturiol.net/Multiplayer.html***, y tiene dos posibilidades de acceso, una para usuarios registrados y otra para usuarios sin registro. Las ventajas que tiene un usuario registrado son la posibilidad de guardar los avances, las puntuaciones y los estados de submarinos (tanto la edición de colores como las mejoras).

## 7.2 Creación de usuarios

Si se opta por la opción de jugar con un usuario registrado lo primero que se tiene que hacer es crear un usuario para el juego. Si, por el contrario, no se pretende registrar ningún jugador, puede pasarse directamente al siguiente apartado (Ingreso en el juego).
El registro de jugadores es rápido y sencillo, únicamente hace falta entrar en la web: **ww*w.monturiol.net/registro.php.***

En esta web se piden un nombre de usuario y contraseña, el registro es totalmente anónimo y la única intención es brindar la posibilidad de de conservar las puntuaciones y mejoras para poder retomarlas en otro momento sin ningún problema.


Pagina web de registro de usuarios

Tras este simple registro ya estás preparado para adentrarte en el mundo submarino con tu propio submarinista y retar al resto de jugadores y a ti mismo a superar las mejores puntuaciones en las carreras.

## 7.3 Ingreso en el juego

La primera pantalla que aparece nada más entrar en el juego es la de ingreso de usuario. En este punto tenemos dos opciones, entrar con un usuario registrado o como invitado. Si aún no tienes un usuario registrado y deseas hacértelo mira en el apartado anterior para ver cómo hacerlo. Si, de lo contrario, prefieres entrar sin usuario registrado y ver el juego, puedes utilizar la opción de entrar como invitado.

Para entrar como registrado puedes ingresar tu nombre y contraseña y clicar *continuar*. En cambio, si pretendes entrar como invitado sólo hace falta dar a *invitado*.
El usuario registrado tendrá todos los beneficios posibles: un nombre preseleccionado, guardado de puntos y mejoras, mientras que el invitado no podrá guardar sus avances y acceder en una siguiente partida. Sin embargo, éste último es una buena alternativa para todos aquellos que quieran probar el juego o simplemente jugar ocasionalmente.

## 7.4 Sala

Una vez se ha ingresado un usuario, tanto si se trata de uno registrado como de un invitado te preguntará qué submarino pretendes usar. Este submarino no tendrá por qué utilizarse siempre, ya que en cualquier momento puedes cambiar el submarino que controlas en el taller.

Una vez seleccionado el submarino tendrás visible toda la pantalla de la sala. Este es el menú principal por el que te moverás para llevar a cabo todas las acciones disponibles antes del juego. Desde aquí puedes escribir al resto de usuarios, crear partidas , unirte a partidas creadas tanto por ti como por otros usuarios, acceder al taller para mejorar el submarino y salir del juego.


Pantalla al entrar en la sala, para seleccionar submarino.

## 7.5 Chat

El chat es un medio de comunicación entre todos los jugadores conectados al servidor. Los jugadores conectados son siempre visibles en el cuadrado superior. En el recuadro justo debajo de éste aparecen todos los mensajes que escribe la gente, y todos los eventos que suceden como el hecho de que un usuario acabe de entrar o que haya salido. Estos mensajes son enviados por el *Máster* (servidor).

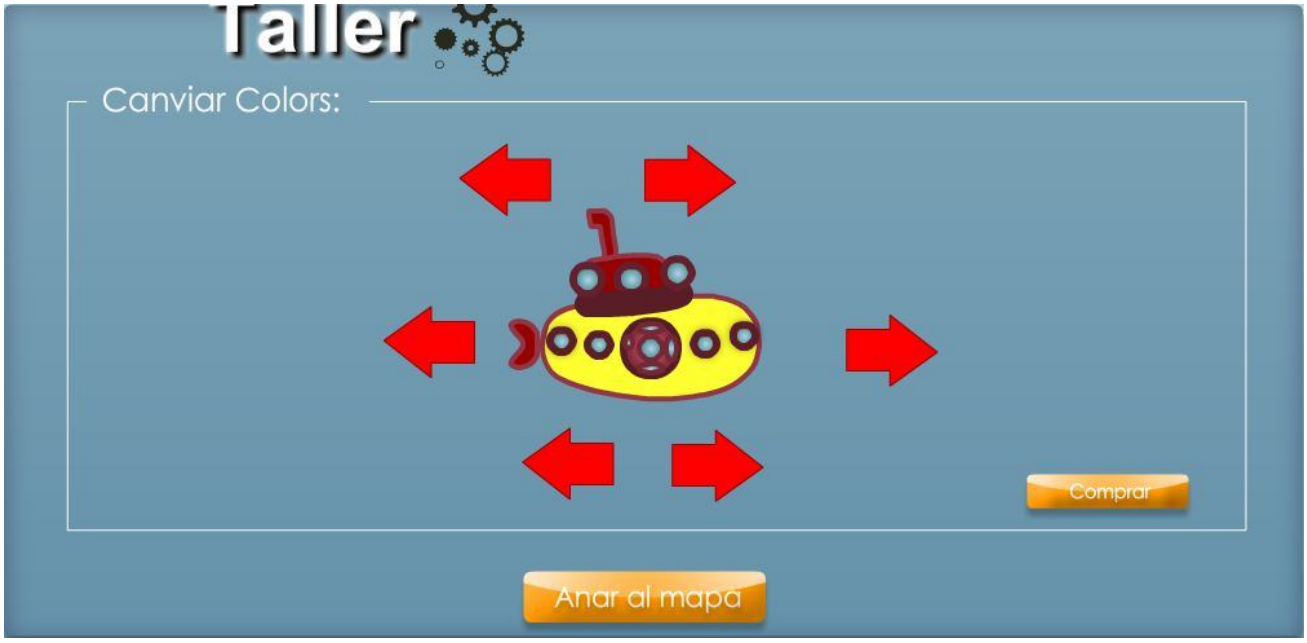
Para escribir un mensaje únicamente tienes que escribir lo que pretendes enviar en el recuadro inferior y darle al botón de *enviar*. Hay que tener en cuenta que estos mensajes que envíes serán legibles por todos los jugadores. Este medio puede ser muy útil para compartir experiencias y puntuaciones con otros jugadores, y también para encontrar competidores para tu siguiente carrera.

## 7.6 Taller del submarino

Si clicas en el botón de *Taller* entras en la selección y edición de submarinos.
En este punto podrás seleccionar el submarino que quieres llevar, entrar a pintar tu submarino o mejorar sus aspectos técnicos.

## 7.7 Pintar submarino

En esta pantalla puedes cambiar el color del submarino en diferentes partes. Lo único que debes hacer es darle a las flechas en la pantalla para cambiar los colores del submarino, y lo podrás hacer tantas veces como quieras sin ningún coste de puntos.




Sección del taller donde pintar el submarino.

El submarino está dividido en tres partes, una es la cabeza o morro del submarino, la otra es el cuerpo y la última es la cola o las aletas de este. Cada parte tiene dos flechas que al clicarlas van cambiando el color de esa parte entre los colores que hay.
Gracias a este apartado podrás hacer tu submarino único y reconocible en todas las carreras.

## 7.8 Mejoras

En este último punto del taller no se ven las mejoras visualmente, pero son muy importantes. Aquí es donde se puede ganar velocidad, aceleración, vida, oxigeno o carga. En este apartado, a diferencia de en el anterior, no se hacen estas cosas gratuitamente, ya que mejorar cualquier aspecto tiene un coste, y ese coste son puntos. Durante las carreras conseguirás puntos y deberás canjear esos puntos por las diferentes mejoras a las que puedes optar.


Sección del taller donde mejorar el submarino.

Todas estas mejoras se verán reflejadas en las carreras. Adéntrate a mejorar tu submarino para hacerlo lo más rápido y maniobrable posible y así ganar a todos los que se te pongan por delante y superar todos los tiempos.

## 7.9 Crear Partida

Al clicar al botón *crear* en la sala nos aparecerá la pantalla donde crear una nueva partida.

En esta pantalla podrás poner el nombre que quieras a la partida que pretendes crear y seleccionar el nivel que quieres correr. Al principio no conocerás todas las pantallas que hay, pero a medida que juegues las conocerás y sabrás en cuál eres mejor para así retar a los demás en tu mejor pista.

Entorno para crear una nueva partida.


## 7.10 Unir partida

En el centro de la sala saldrá una lista de salas creadas. Estas salas pueden haber sido creadas tanto por ti como por otros usuarios.


Ejemplo de una partida creada. Lista para unirse y jugar.

Si eres el creador de la partida podrás iniciar y cerrar las partidas siempre que quieras, o unirte. En caso de no serlo, únicamente podrás unirte a la partida para correr esa carrera.

En el momento en que se inicie saldrás de la sala y empezará tu carrera.

Los usuarios que crean una partida tienen la opción de no unirse. De este modo, un usuario puede crear las partidas para que resto puedan jugar.

## 7.11 Juego de carreras

Una vez entrado en un juego saldrá el mensaje de bienvenida y empezará la cuenta atrás para tu carrera. Los controles son muy sencillos, el movimiento se controla con las flechas para dirigirse en esa dirección. Ten cuidado con los obstáculos y el submarino enemigo, pues te quitarán vida.

Hay dos tipos de circuitos: circuito abierto o circuito cerrado. El primero consta de recorridos lineales en los que tendrás que superar puntos de control hasta llegar a la meta, saltando obstáculos y evitando chocarte, siempre con la intención de llegar antes que el otro jugador. En el caso de los circuitos cerrados, son circuitos que como su nombre indica acaban donde empiezan y, por lo tanto, tienes que dar vueltas sobre él mismo, aunque dependiendo del circuito se tendrán que dar más o menos vueltas.

Corre contra tus adversarios y consigue ganarles a todos.

## 7.12 Fin del juego y puntuación

Al acabar, saldrá la pantalla de final de juego en la que se te mostrarán los puntos conseguidos y tus tiempos en caso de haber finalizado la carrera.
Estos puntos serán los que, más tarde, usarás para mejorar tu submarino.
Tras esta pantalla volverás otra vez a la sala del menú principal para correr tantas carreras como desees.

# CAPÍTULO 8: PLANIFICACIÓN Y COSTES

## 8.1 Planificación

Tras plantear los objetivos, y antes de empezar a trabajar hay que hacer una planificación, para de este modo conseguir hacer nuestro proyecto de la forma más eficiente. En los siguiente recuadros aparecen tanto la meta a conseguir, como el tiempo estimado y real.

**Documentarse ActionScript 3**…….. Estimada: **50 horas** - Real: **50 horas**

- Documentación para creación y comunicación.

**Decidir medio de comunicación**…….. Estimada: **5 horas** - Real: **5 horas**

- Decisión sí como comunicar con el servidor, socket con XML.

**Especificar diagrama de casos**..... ……. Estimada: **2 horas** - Real: **2 horas**

- Crear y especificar el diagrama de casos para saber cómo actuará el usuario.

**Especificación diagrama de clases**..... ..…… Estimada: **8 horas** - Real: **8 horas**

- Especificación de las clases y sus relaciones.

**Especificación comunicación XML**..…….. Estimada: **15 horas** - Real: **15 horas**

- Especificar la comunicación XML tanto entre cliente-servidor como entre jugadores.

**Implementación movimiento Submarino** .....…… Estimada: **20 horas** - Real: **20 horas**

- Implementar el movimiento del submarino con aceleraciones y frenada correcta.

**Implementación movimiento de Fondo**…….....…….. Estimada: **50 horas** - Real: **50 horas**

- Implementar el movimiento del fondo. De este modo simulará el movimiento del submarino.

**Implementación de las colisiones .** ……..…… Estimada: **60 horas - Real: 80 horas**

- Implementar todas las colisiones que se pueden producir y sus reacciones.

**Implementación Juego** …… Estimada: 3**0 horas - Real: 3**0 **horas**

- Implementar integración en una partida, con inicio final y los estados de la partida.

**Implementación puntuación** .....…… Estimada: **15 horas - Real: 15 horas**

- Implementar la cabecera con la puntuación, tiempo, vida y oxigeno.

**Implementar clase comunicaciones** .....…….... Estimada: **10 horas - Real: 20 horas**

- Crear una guía para instalar y configurar correctamente el entorno del servidor.

**Implementar sala** ……..……..…… Estimada: **90 horas - Real: 100 horas**

- Implementar la sala conectando al servidor y creando las partidas.

**Añadir modo multijugador al juego**…….....…… Estimada: **20 horas - Real: 20 horas**

- Añadir al juego el modo multijugador, para jugar contra otro jugador.

**Añadir comunicación a sala para crear juegos**……..…… Estimada: **5 horas - Real: 5 horas**

- Conseguir gestionar el arranque de una partida y salir de la sala.

**Reuniones** .....……..…… Estimada: **20 horas - Real: 20 horas**

- Reuniones de control.

**Total Estimadas: 400 horas Total Reales: 440 horas**

Como se puede ver el tiempo empleado total han sido 40 horas más de lo previsto. Esto tiene su explicación.

La primera dificultad que encontré está en las colisiones, puesto que las colisiones básicas son demasiado simples, por lo que se tuvo que idear un método más complejo, por lo que se tuvieron que hacer varias pruebas así que llevo más tiempo.

En el momento que empecé con las conexiones de socket, encontré un aspecto que no esperaba. Problemas con la seguridad del servidor, Flash suele hacer actualizaciones de su reproductor, y en la última habían añadido más seguridad, por lo que era más restrictivo y se tenía que mirar bien cómo funcionaba.

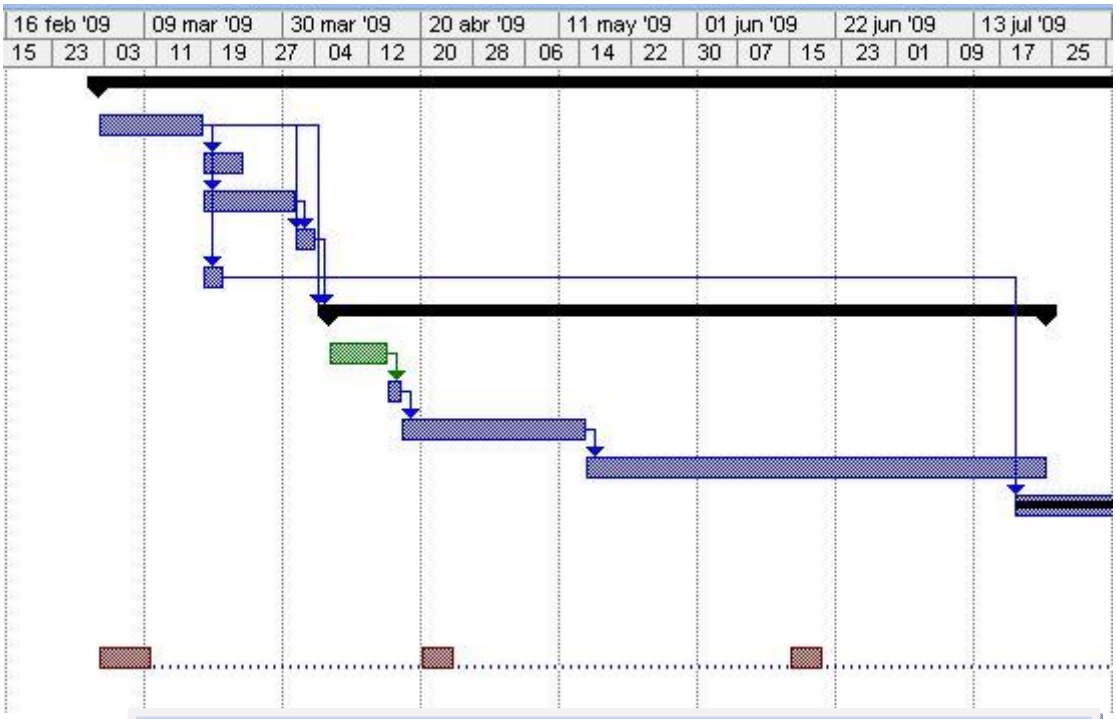
La última traba importante estaba en la sala. Ya que todo estaba programado, pero era el punto de poner en conjunto tanto la sala como los servidores, para entrar, salir, guardar y todas las funciones, y se prolongó más de lo esperado.

## 8.2 Diagrama de Gantt

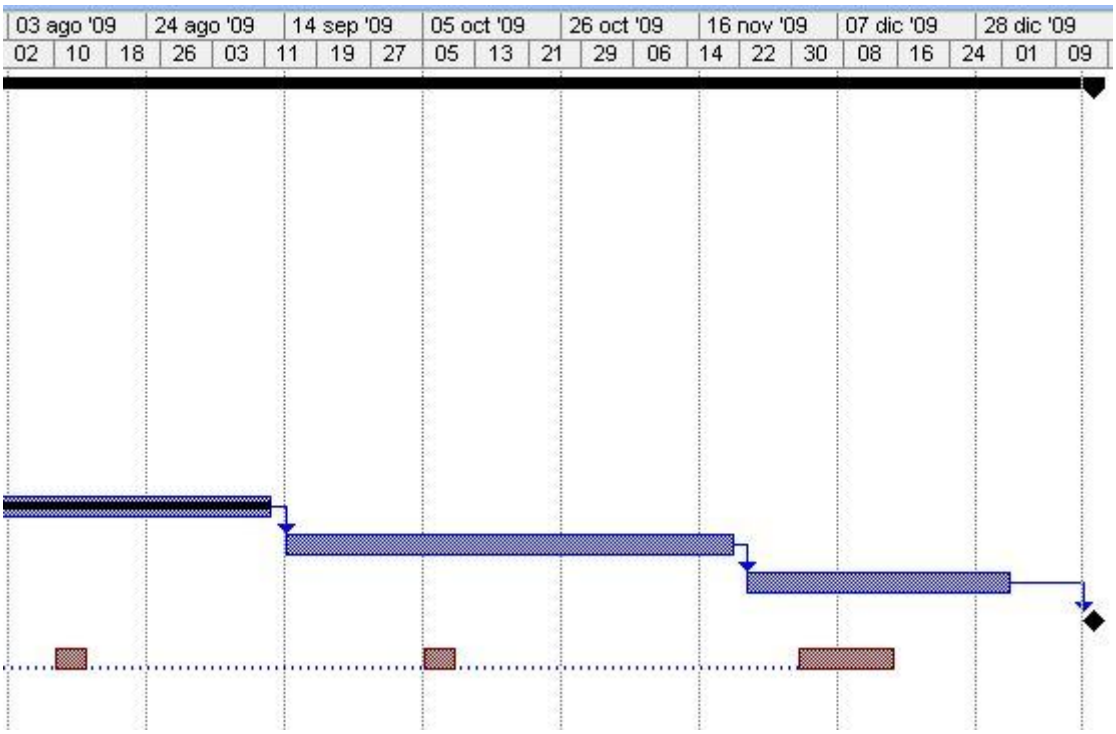
A continuación adjunto el diagrama de Gantt, con este grafico la intención es mostrar, de forma gráfica el tiempo de dedicación previsto para el proyecto.

| | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | **Proyecto** | **229 días** | **lun 02/03/09** | **vie 15/01/10** | |
| 2 | Documentarse ActionScript 3 | 12 días | lun 02/03/09 | mar 17/03/09 | |
| 3 | Decidir Medio de comunicación | 4 días | mié 18/03/09 | lun 23/03/09 | 2 |
| 4 | Especificar diagrama de casos | 10 días | mié 18/03/09 | mar 31/03/09 | 2 |
| 5 | Especificar diagrama de clases | 3 días | mié 01/04/09 | vie 03/04/09 | 4;2 |
| 6 | Especificación comunicación XML | 3 días | mié 18/03/09 | vie 20/03/09 | 2 |
| 7 | **Implementación Juego** | **79 días** | **lun 06/04/09** | **jue 23/07/09** | **5;2** |
| 8 | Implementación movimiento del subm | 7 días | lun 06/04/09 | mar 14/04/09 | |
| 9 | Implementación movimiento de Fond | 2 días | mié 15/04/09 | jue 16/04/09 | 8 |
| 10 | Implementación de colisones | 20 días | vie 17/04/09 | jue 14/05/09 | 9 |
| 11 | Implementación puntuación | 50 días | vie 15/05/09 | jue 23/07/09 | 10 |
| 12 | Implementación clase comunicaciones | 40 días | dom 19/07/09 | vie 11/09/09 | 6 |
| 13 | Implementar Sala | 50 días | lun 14/09/09 | vie 20/11/09 | 12 |
| 14 | Comunicación sala y juego | 30 días | lun 23/11/09 | vie 01/01/10 | 13 |
| 15 | Entrega memoria | 0 días | vie 15/01/10 | vie 15/01/10 | 14 |
| 16 | Reuniones de Control | 36,5 días | lun 02/03/09 | mar 15/12/09 | |

:




Primera parte diagrama de Gantt




Segunda parte diagrama de Gantt

## 8.3 Coste económico

Seguidamente está la tabla del coste económico.

Siendo los criterios o subgrupos escogidos los siguientes:

- (PM) Project manager → 50€/hora
- (IS) Ingeniero en Sistemas → 30€/hora
- (D) Diseñador → 35€/hora
- (P) Programador → 20€/hora

| Tarea | Horas | Especialidad | Precio unitario | Precio total |
|---|---|---|---|---|
| Decidir medio de comunicación | **5** | Ingeniero en Sistemas | 30€ | **150 €** |
| Especificar diagrama de casos | 2 | Diseñador | 35€ | **70 €** |
| Especificar diagrama de clases | **8** | Diseñador | 35€ | **280 €** |
| Especificar XML | 15 | Diseñador | 35€ | **525 €** |
| Implementación juego sin multijugador | **175** | Programador | 20€ | **3500 €** |
| Implementación clase comunicaciones | 10 | Programador | 20€ | **200 €** |
| Instalación del entorno de pruebas/desarrollo | **4** | Ingeniero en Sistemas | 30€ | **120 €** |
| Implementación de los componentes | **20** | Programador | 20€ | **400 €** |
| Implementar sala | **90** | Programador | 20€ | **1800 €** |
| Añadir comunicación entre sala y juego. | **5** | Programador | 20€ | **100 €** |
| Reuniones | **20** | Director de proyecto | 50€ | **1000 €** |

**TOTAL: 8145** €

# CAPÍTULO 9: CONCLUSIONES

## 9.1 Posibles ampliaciones

### Añadir juegos

En un futuro se podrían añadir más juegos al modo multijugador. En la versión de un sólo jugador existen diversos tipos de juegos, entre ellos el de carreas, igual que este se pasó también a modo multijugador se podrían añadir más o crear nuevos. En la sala está contemplado que hayan diferentes tipos de juegos, tanto a la hora de crearse como en la selección de este juego. Además para proceder a pasarlos a multijugador se tiene que seguir el mismo proceso que con el juego de carreras, con pequeñas variaciones y seguramente añadir alguna cosa.

### Añadir más jugadores

De momento en cada carrera pueden participar un máximo de 2 jugadores o un mínimo de 1. Pero se le podrían añadir más jugadores a las carreras. El protocolo está pensado con identificación de usuarios por lo que simplemente es añadir más submarinos enemigos y controlarlos. A parte de que aparezcan, se tendría que mirar cuánto se podría sin que se sobrecargue la red y haya problemas de comunicación.

### Sincronización de tiempos

Actualmente un cliente controla el tiempo de la partida y éste es el que manda, por lo que el máster siempre controlará esto, pero si uno tiene algún fallo en el envío el otro seguirá. Poniendo control de tiempos ambos irán enviando sus tiempos por lo que si uno tiene algún fallo de conexión se pararían los dos generando el conocido *lag* o retraso que se da en los juegos, pero los tiempos serán iguales.

## 9.2 Conclusión

Como se suele decir" *si tu trabajo te gusta, lo haces más a gusto y mejor*". Pues bien, ¿qué puedo decir de hacer un proyecto de un videojuego multijugador? Antes de entrar en la carrera, siempre había tenido la ilusión de acabar programando videojuegos, pero durante la carrera las cosas cambian y tus objetivos también.

Sin embargo, hace cosa de unos meses, al ver este proyecto, toda aquella ilusión que creía perdida, brotó. Crear un videojuego con el que jugar contra otro jugador mediante internet era algo muy tentador. Pero el proyecto no sólo constaba de la creación del videojuego, sino también en la del servidor con el que comunicarse.

Por lo tanto, ahora se le añadía un aliciente más al proyecto. Ahora constaba de dos integrantes para su producción, no solamente iba a experimentar en la creación de un videojuego, con lo que me realizaba plenamente, sino que además tendría que trabajar y cooperar con otra persona en un proyecto mucho más grande e importante de lo que hubiera trabajado antes. Esto aunque a simple vista parezca que sea más sencillo, puede traer más complicación de lo esperado, pues dos cabezas diferentes aportan más ideas, pero también diferentes puntos de vista, y tan importante es encontrar una solución como encontrar una solución que a los dos les parezca bien.

Tras las designaciones de trabajo, el proyecto comenzó a rodar. Nos vimos envueltos primero en un trabajo individual, cada uno con su entorno de trabajo. En mi primer contacto, tenía que conseguir que el juego funcionara para una sola persona, lo que significaba que tendría que trabajar mucho por mi cuenta, creando los movimientos y colisiones, pero pronto empezaba a ver resultados, y cada cosa que conseguía me motivaba más aún si cabe.

Finalmente, obtuve una faceta de un sólo jugador, totalmente practicable, y ¿porque no decirlo? muy divertida. Pero aun no estaba acabado, ahora teníamos que unirlo, comunicarlo y de este modo conseguir que dos jugadores consiguieran jugar uno contra el otro en una carrera de submarinos. Ahora empezaba lo más complicado, unir los dos puntos, el servidor y el juego, pero no podíamos esperar para conseguir un resultado y empezar a jugar uno contra otro.

Unir las dos partes no fue algo muy sencillo, pero al ver el producto, todo el tiempo empleado anteriormente se vio más que recompensado: correr el uno contra el otro, retarlo, chocarse contra él, y pronto pudimos probarlo con nuestros amigos.

Mirando ahora hacia atrás, todo lo que queda son aspectos positivos, conocimiento en un lenguaje como es ActionScript, y una experiencia, tanto a la hora de programar, como a la de trabajar en un pequeño grupo, ambas muy útiles para el mundo laboral. Sin olvidar los buenos ratos, probando submarinos y todas las colisiones, es decir, jugando. Me hace volver a pensar; "*si tu trabajo te gusta, lo haces más a gusto y mejor*".

# CAPÍTULO 10: BIBLIOGRAFÍA

## ActionScrip3:


**GARY ROSENZWEIG** *ActionScript 3 Game Programming* - University

**COLIN MOOCK** *Essential ActionScript 3.0* - O'Reilly Media Junio 2007

**WILLIAM SANDERS** *ActionScript 3.0 Design Patterns: Object Oriented Programming Techniques* - O'Reilly Media Diciembre 2008

**JOBE MAKAR** *ActionScript for Multiplayer Games and Virtual Worlds* - Paperback

http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/

http://www.cristalab.com/

http://www.forosdeflash.com/blog/

http://ffiles.com/