Universidad Carlos III de Madrid

Escuela politécnica superior

INGENIERÍA INFORMÁTICA

Proyecto Fin de Carrera

Desarrollo de un Juego Web Multiusuario con HTML5

Autor: Pedro Henrique Pereira-Castillo Nunes.

Director: Jesús Carretero Pérez.

Fecha de terminación: 14 de Septiembre de 2012.

"Soportaré ser quemado, herido, golpeado y asesinado por la espada."
Parte del juramento del gladiador Romano.

"Batou: ¿sabes qué son los derechos del hombre?
Motoko: Creo que no…
Es un concepto nacido de la frontera entre la moral y la realidad.
Consigo entender su ideal de fondo,
pero nunca he visto que se aplicase."
Ghost in The Shell, Masamune Shirow.

"All fled, all done, so lift me on the pyre;
The feast is over and the lamps expire."
Nota aparecida de Robert E. Howard.
(Extraídas del poema "The House of Caesar" de Viola Garvin)

A todos los animales de compañía,
Porque solos sus dueños saben apreciar su valor.
En especial a Newton y Leo.

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Índice de Códigos

1 Introducción

El proyecto del que trata este documento está enfocado a diseñar e implementar un juego sencillo para navegador web, basado en HTML5, nuevo estándar de la W3C que actualmente está en fase experimental, pero que muchos de los grandes navegadores hacen uso de él.

La elección del uso del nuevo estándar es principalmente debida a que al ser una estructura heterogénea, permite una gran interacción con diferentes dispositivos, consiguiendo una mayor facilidad a la hora de portar el sistema e independizarlo en gran medida de la plataforma en donde se desarrolla.

1.1 Problema

Con la aparición de los computadores, el mundo del ocio electrónico ha ido creciendo exponencialmente, haciéndolo un negocio muy lucrativo. El mundo del videojuego ha tenido la mayor parte de culpa a la hora de imponerse en muchas ideas, o ha conseguido cambiar muchas de las tendencias. Con estas nuevas tecnologías han ido imponiendo desarrollos cada vez de un grado mayor de conocimiento y de eficiencia. En particular, los videojuegos de navegador, que son aquellos enfocados a poder utilizarse en un navegador web, han ido creciendo junto a la misma web.

Por otra parte, las grandes empresas que hacen un uso exhaustivo de la web, haciendo millones de operaciones incluso en un segundo, han necesitado mejorar el estándar de la web, para obtener una mayor eficiencia en la comunicación, en la presentación y de conseguir que el desarrollo en la parte cliente del navegador, sea más fácil produciendo nuevas funcionalidades.

El videojuego de navegador ha ido utilizando esos nuevos conocimientos, para también beneficiarse de ello. De esta forma se consigue una mayor interoperabilidad y divertimento por parte del usuario. Además, los videojuegos por navegador se han caracterizado por ser unos juegos que tenían ciertas limitaciones a la hora de que los usuarios pudiesen interactuar entre sí. Para evitar esos inconvenientes, las lógicas de los juegos han tenido que posicionarse para que no fuese un problema. Así, han conseguido convertir el aspecto desfavorable en una mejora que da nuevas oportunidades.

Es por ello que con la llegada del nuevo estándar que se ofrece para HTML5, impulsado por parte de todos, hace que los videojuegos de navegador sufran un nuevo impulso, que habrá que demostrar si es para mejorar o cambiar su perspectiva actual.

1.2 Objetivos

Existen varios objetivos que se persiguen con la consecución de este proyecto. El más general es conseguir desarrollar un juego sobre una plataforma web, la cual está limitada, tanto en recursos como en bibliotecas a utilizar. Es esencial intentar minimizar estos inconvenientes y es por ello primordial tener claros ciertos factores para tomar las decisiones necesarias.

Para cumplir el objetivo general, es necesario saber sobre qué y dónde se va a diseñar y desarrollar todo lo necesario. Ya que, por ejemplo, todos los navegadores web no disponen de unas librerías tan específicas como es necesario para HTML5, resultando muy poco abarcable el desarrollarlo para todas las plataformas existentes y con diferentes estándares.

A la hora de estudiar el protocolo de comunicación, porque es quizá el aspecto más importante a la hora de desarrollar una aplicación web, es imprescindible saber en qué plataforma se desarrollarán por ser significativo tanto por coste como por el tiempo en aprendizaje al usarlo.

Por ello es necesario realizar un estudio del uso del protocolo de comunicación que ofrece HTML5 porque puede plantear inconvenientes. Su bajo rendimiento en comparación a otros protocolos limita en muchos aspectos el tipo y finalidad del programa que se desea implementar. Por ejemplo, puede resultar absurdo intentar conseguir con el protocolo una aplicación que funcione en tiempo real, en cambio otros sistemas más livianos en cuanto al uso de la comunicación serían los idóneos.

En consecuencia es necesario estudiar las posibles alternativas de la implementación del protocolo y qué posibilidades hay de poder mejorar su rendimiento. Será clave el conocer todos los aspectos de la comunicación que se engloban y seleccionar el que mejor se pueda adaptar a la consecución del proyecto.

Otro objetivo considerable es el de definir claramente las reglas del juego a desarrollar, para saber qué va a ser necesario realizar y seguir ciertas pautas en el desarrollo que sólo sabiendo las reglas se podrán solucionar correctamente.

Con la combinación de los dos puntos anterior; y conociendo la historia de los juegos por navegador, se intentará comparar su rendimiento al usar las tecnologías ofertadas en el estándar y qué mejoras se ofrecen en comparación. De esta forma se sabrá si la selección del juego y del protocolo de comunicación son los idóneos.

Y por último, sabiendo en qué plataformas se va a realizar el sistema, el objetivo directo del proyecto es el de asociar tanto la parte cliente (el navegador web con la lógica del juego) con el servidor web que dispone de los servicios que se ofrecen al cliente, apoyándose sobre las reglas del juego para saber cuándo y cómo realizar la unión de las dos partes.

No se busca en ningún momento el desarrollar un juego completo, como los juegos comerciales que se venden actualmente, ni tampoco llevar al máximo rendimiento las plataformas existentes, con el uso por ejemplo de tiempo real o de librerías en tres dimensiones. Usar un protocolo de comunicación en fase de desarrollo conlleva una posible reducción del rendimiento, y por tanto no es el mejor sistema para conseguir un juego de última generación. Pero como ya se ha comentado, se consigue una gran flexibilidad haciendo completamente transparente el sistema donde están los servicios, para que los clientes que hagan uso de ellos no tengan que preocuparse de saber cómo comunicarse con él.

1.3 Estructura del documento

El documento está estructurado para que se pueda ver como evoluciona el sistema desde su concepción hasta los últimos pasos realizados para su elaboración. En un primer lugar se realiza una introducción en la que se presenta un problema y se establecen los objetivos a conseguir.

Luego se realiza un pequeño estudio sobre las tecnologías actuales de cada uno de los elementos que se requieren utilizar. A continuación se detalla la lógica necesaria a implementar, para después establecer un alcance y definir las características necesitadas.

Posteriormente se detalla un diseño con los elementos que se necesitan para cumplir con los requisitos detallados en el apartado anterior. Y seguidamente se detallan los aspectos más relevantes de la implementación y de la interfaz gráfica.

Por último se realizará una evaluación de ciertos aspectos a destacar y se establecerán las conclusiones para comprobar si se han conseguido llevar a cabo todos los objetivos establecidos.

2 Estado del arte

Es necesario realizar un análisis de las actuales circunstancias para realizar la mejor valoración. Se hará un breve repaso del nuevo estándar de HMTL, realizando una comparación con el actual.

Se realizará un breve análisis a la situación de los videojuegos, haciendo especial hincapié en los juegos por navegador. Teniendo en cuenta las principales diferencias con los videojuegos clásicos/actuales, se estimarán las diferencias entre realizar una aplicación empotrada en un sistema en concreto, a hacer aplicaciones multiplataforma, incluidas las realizadas para navegadores web.

Es también muy importante el entorno del servidor que se escogerá, por lo tanto es necesario hacer también una evaluación de las necesidades y de qué sistemas se disponen en la actualidad.

Por último, al ser una aplicación multiusuario, en la que los clientes estarán continuamente enviándose información entre sí, un apartado que no se puede pasar por alto será la comunicación. Se estudiarán brevemente las tecnologías de comunicación que se podrían aplicar.

2.1 HTML

El lenguaje de marcado de hipertexto, que son las siglas de HTML, es un subconjunto del SGML, y sirve para crear documentos estructurados, conteniendo en formato texto tanto la estructura como el contenido. HTML utiliza etiquetas o marcas para definir las estructuras, cada navegador las utilizará para mostrar según dichas marcas la información que se ha expresado en su interior.

La W3C (siglas para definir la World Wide Web Consortium), es la encargada de llevar a cabo las nuevas recomendaciones, diseñando y preparando las nuevas normativas. Actualmente se encuentra desarrollando una nueva versión, denominada HTML5, pero aunque es considerada como experimental por la W3C, ya se encuentra implementada en su gran mayoría por muchos de los navegadores del mercado.

La versión actual de HTML, data de 1999, hace ya más de 10 años. Con el paso del tiempo han surgido nuevas necesidades, por lo cual el nuevo estándar era de esperar que surgiese tarde o temprano. Muchos contenidos de la actual web, han tenido que utilizar herramientas de terceros para poder hacer lo que se deseaba, como por ejemplo la visualización de vídeos incrustados en el lenguaje. El nuevo estándar intenta subsanar esas evidentes desventajas, para que principalmente se aprenda de los errores cometidos, intentando solucionar la mayoría de incidencias con los que un desarrollador web se pueda encontrar.

Con relación a esto último, es evidente que el enfoque ha cambiado en esta versión con respecto a versiones anteriores. En esta versión se tiene en cuenta que ya el contenido web no es sólo texto (como si fuesen documentos en sí), sino que contiene páginas cada vez más dinámicas, más relacionadas con aplicaciones de escritorio.

En el presente, como se ha mencionado, la mayoría de navegadores implementan parcialmente el nuevo diseño. Se estima que para 2014 sea ya el estándar de facto, y recomiendan encarecidamente que los desarrolladores lo usen.

Como hay tantas diferencias con el estándar actual, solamente se repasarán por encima las más importantes, sobre todo de los elementos que se añaden. La omisión u olvido de alguna de las diferencias no se hace intencionadamente, pero es importante resaltarlo.

- Se han añadido elementos para mejorar la estructura a la hora de controlar mejor las diferentes secciones de una página, además de añadir nuevos elementos a los formularios.

En el caso del control de la estructura, se hacía un uso excesivo de un elemento denominado div para dividir las diferentes secciones de una página web. Ahora con estas etiquetas será más sencillo de entender por personas ajenas como por motores de búsqueda u otra aplicación que necesite interpretar las páginas web.

Para los formularios se han añadido nuevos elementos que permiten un mayor manejo de los datos que se introducen, algo que se tenía que controlar manualmente.

- Se han añadido las etiquetas de audio y video. De esta forma se pueden incrustar tanto video como audio en la propia página, sin necesidad de utilizar elementos externos o de terceros.

- Hay un nuevo elemento denominado canvas, que permite generar gráficos dentro de él, abriendo muchas posibilidades a los usuarios para crear gráficos de manera sencilla y sin necesidad de instalar programas de terceros.

- Se añaden nuevos atributos a las etiquetas, tanto atributos globales (que pueden aparecer en cualquier etiqueta) como los que pertenecen a un subconjunto.

  Todos los nuevos atributos permitirán un mayor control de ciertas funcionalidades que antes no eran posibles, consiguiendo así que el desarrollador se beneficie de ello.

- Se añaden nuevas librerías para los scripts que se utilicen en el documento HTML. Entre las muchas que se añaden, las más valorables son las de manejo del elemento de canvas, el geo posicionamiento para los dispositivos que lo permitan, el poder trabajar offline, el uso de una base de datos local basado en SQLite, el poder tener hilos de ejecución.

  Y quizá la más importante para el entorno del proyecto: una librería para poder realizar comunicación bidireccional, denominada Websockets. De esta forma el servidor podrá comunicarse con el cliente y viceversa. Anteriormente, y de manera sucinta, la comunicación era solo en un sentido, el cliente lanzaba una petición con una url y el servidor le respondía con la página web. Ahora, es posible que el servidor pueda enviarle ciertos datos al cliente, siempre que sea posible.

2.2 Videojuegos

Existe una gran variedad de categorías y géneros en las que se podrían englobar cualquiera de los videojuegos de la actualidad. Se ha especializado tanto y es un mundo tan lucrativo que actualmente algunas producciones pueden llegar a rivalizar con otras de otros ámbitos.

Un ejemplo claro de ello es el marketing que realizan actualmente para poder venderse al mayor público posible. Realizan anuncios con personajes conocidos o videos promocionales que su gasto es casi similar a la producción del videojuego.

La mayoría de videojuegos podrían aparecer en varias de las categorías y por lo tanto no debería considerarse cada categoría como cerrada. Con el tiempo también han aparecido nuevos desarrollos que son más difíciles de identificar, ya que para convencer al jugador, se van necesitando más y más funcionalidades que no corresponden a su presumible género.

Los géneros o categorías donde podrían englobarse los juegos son las siguientes:

- **Aventura:** Es un género que posiblemente es el que más ha cambiado y cada vez se le considera menos evidente, porque es muy fácil que los juegos acaben abarcando otros de los muchos géneros. Se considera un juego de aventura aquel que el protagonista debe avanzar en la trama haciendo uso de objetos o interactuando con otros personajes.

- **Acción:** Son los juegos en los que el jugador debe disparar a los rivales para poder avanzar. Dependiendo de la temática puede ubicarse en muchos otros subgéneros. Como por ejemplo puede depender de la perspectiva (acción en primera persona, en tercera persona), como en el objetivo (los conocidos mata marcianos).

- **Puzle / Educativos:** El objetivo en estos juegos suele ser adquirir algún tipo de conocimiento, ayudándose de tramas o algún otro género para adquirir ese conocimiento. Actualmente existen juegos que ayudan a aprender idiomas o a que los niños se familiaricen con el nuevo entorno que le ofrece la informática.

- **Estrategia:** Este género se caracteriza por la gestión o manipulación de elementos del juego para alcanzar el objetivo del juego, que puede ser tanto bélico, como económico o social, aunque muchas veces se pueden mezclar.

- **Arcade / Lucha:** En este género el jugador debe combatir, normalmente usando técnicas de artes marciales o algún tipo de combate cuerpo a cuerpo, contra uno o más contrincantes.

- **Survival Horror:** Como la misma palabra indica, el jugador debe sobrevivir a situaciones muy dispares, pero en el que se suele utilizar el terror psicológico para influir sobre el jugador.

- **Plataformas:** Se ha convertido en un género clásico de las videoconsolas o máquinas recreativas. El jugador debe avanzar en la partida esquivando una serie de obstáculos, las plataformas o enemigos, para conseguir el objetivo.

- **Rol:** El jugador en este género irá mejorando su personaje mientras interactúa con su entorno, ya sea por medio de objetos o por combatir contra otros seres.

- **Musicales / Party:** Con la aparición de nuevos periféricos, tales como micrófonos, mandos a distancia, cámaras, etc. han aparecido nuevos videojuegos que quizá no se podrían englobar en otras categorías. Los musicales son aquellos que usan los periféricos para componer y/o competir con otros jugadores. Los Party Games pueden ser musicales, o pueden utilizar el mando clásico para avanzar y competir contra los otros jugadores, pero su finalidad es utilizar la consola para que un grupo de jugadores compita entre sí.

- **Simulación:** El objetivo es, como su propia palabra indica, simular un entorno real para que el jugador pueda interactuar. Un juego clásico de éste tipo podría ser el de un simulador de vuelo, en el que el jugador debe ser el piloto de un aeroplano y hacerlo volar. Otros tipo de simuladores pueden ser el de gestionar una ciudad o una actividad en concreto, por ejemplo un hospital, un parque de atracciones, etc.

- **Deportivo:** El jugador en este género se centra en el control de un deporte, ya puede ser el manejar un equipo de fútbol, desde las facetas de un entrenador, hasta de manejar el equipo en un partido de fútbol. Puede ser cualquier tipo de deporte, siendo los juegos dedicados a las olimpiadas los más completos a la hora de manejar diferentes deportes.

- **Conducción / Carreras:** Éste género debería estar a medio paso entre el género deportivo y el de simulación, pero se ha especializado tanto que es quizá mejor englobarlo en un género nuevo, porque es muy fácil identificarlo y a veces hace uso de otros géneros para diferenciarlo de los demás. El jugador puede manejar cualquier tipo de vehículo, pero muchas veces la competición entre los jugadores es más un tipo de Arcade que un juego de simulación propiamente dicho.

- **No Lineal:** Se caracterizan por hacer uso de cualquiera de los géneros anteriores, pero su faceta más relevante es que el jugador es libre completamente de hacer lo que quiera en el escenario que se le presenta.

Es muy probable que con el auge y el desarrollo actual que tiene el mundo del videojuego, los géneros se difuminen cada vez más y que muy probablemente aparezcan géneros que antes ni se pensaba que se podrían hacer.

Para no divagar mucho más en el asunto de los videojuegos, una clasificación que puede interesar más es en el tiempo de respuesta que interesa:

- **Tiempo real:** Se caracterizan por juegos en los que el jugador o jugadores realizan sus acciones a la vez en el tiempo y que en el transcurso del juego no se detiene para poder continuar.

- **Turnos:** Los jugadores hacen sus acciones en un intervalo de tiempo que quizá no esté limitado. Aquí cada jugador puede definir sus acciones independientemente y muchas veces se paraliza el tiempo para que cada jugador pueda tomar sus decisiones.

Otro apartado que interesa mucho es el de la posibilidad del multijugador. Con la llegada de la red global, muchas plataformas y juegos hacen uso de la conexión a internet para poder ofrecer mayor entretenimiento a los jugadores. Con éste nuevo uso, hace que el género de un juego pueda cambiar cuando está el modo multijugador.

Por otro lado, algo que quizá no parezca importante, y menos aún a grandes empresas que se lo pueden permitir, es el tema de las licencias. Muchas de las plataformas de desarrollo suelen ser de pago, sobretodo en entornos tan cerrados como pueden ser el mundo de las videoconsolas. Quizá el entorno de desarrollo pueda ser gratuito, pero si quieres desarrollar algo y publicarlo, en algún momento se cobre la parte que corresponde.

Para una gran empresa quizá esto no sea un inconveniente, incluso no hacer el gasto necesario puede ser la diferencia de conseguir el buen posicionamiento deseado. Pero por otro lado, existen alternativas gratuitas o de muy bajo coste. Cada vez los grandes entornos permiten que las pequeñas empresas o desarrolladores, puedan publicar sus aplicaciones con cierta facilidad. Esto hasta hace unos años era inviable, pero gracias a las nuevas innovaciones y a los cambios de tendencias actuales, es algo habitual.

Algo también muy común que se hace en la industria es desarrollar para una plataforma concreta, para que luego un tercero se dedique a portarlo a otras plataformas. Esto se hace

comúnmente en la actualidad, porque el desarrollo se centra especialmente para las videoconsolas, que son las que están manteniendo un auge mayor.

Puede que al hacer la portabilidad, se pierda cierto grado de rendimiento y que muchas de las funcionalidades estén para unas plataformas mejor preparadas que otras. Muchas de las críticas vienen en este sentido, al intentar que una franquicia pueda estar en el mayor número de plataformas y llegar al mercado con mayor facilidad.

Aunque puedan reducir sus prestaciones al hacerlo así, también es considerable el posible ahorro económico que conlleva al no tener que desarrollar el mismo producto varias veces. Por eso, una de las soluciones es crear aplicaciones completamente multiplataforma.

Evidentemente, al realizar un juego multiplataforma, también hay muchos inconvenientes, quizá el más importante, y seguramente el por qué no se utiliza, es el rendimiento gráfico. Cada vez los juegos hacen un uso tan exhaustivo del apartado gráfico que se debe exprimir al máximo, simplemente para hacer frente a la competencia y no quedarse al margen de los demás desarrollos.

A parte de los posibles inconvenientes, es quizá muy difícil de conseguir una aplicación multiplataforma al 100%. El por qué puede venir probablemente de la mayoría de dispositivos existentes: utilizan tanta cantidad de arquitecturas tan diferentes, que se hace incierto que el lenguaje o la plataforma seleccionada tenga las mismas características en los apartados más significativos.

Por poner un ejemplo: si se desarrolla un juego en Java, es probable que no te funcione en todos los dispositivos, debido en mucha medida a que los productores de las videoconsolas, no les interesa ni económicamente ni estructuralmente tener la máquina virtual de Java en su Sistema Operativo. Quizá el más probable de conseguir el 100% sea el de los juegos para navegador, ya que es muy posible que los dispositivos existentes y los que saldrán al mercado, dispongan de un navegador que pueda ejecutar scripts.

Los juegos de navegador más comunes sufren una ligera modificación en su clasificación, porque han sufrido las limitaciones existentes en el estándar. Para paliarlo, han tenido que jugar con ello para hacer un tipo de juegos que llamase la atención a los posibles jugadores.

La clasificación más tradicional de los juegos para navegador gira en torno al modo multijugador. Se utiliza el término multijugador masivo porque prácticamente juegan cientos de miles de usuarios a la vez, siendo el factor más predominante de los juegos más conocidos. Con

eso y teniendo en cuenta las limitaciones del estándar, han sabido captar cualquier de los géneros existentes y que se han comentado anteriormente, intentando adaptarlos de la mejor manera posible. Los más comunes son principalmente los de estrategia, de rol y de simulación de gestión, pero no significa que los otros géneros no se utilicen.

Por otro lado, los juegos suelen ser por turnos, aunque también los hay de tiempo real. Estos suelen ser mucho más restrictivos y suelen usar ciertas limitaciones heredadas de las restricciones que impide el estándar.

Con el uso de plugins de terceros (los más conocidos son Flash y Java) que se instalan en los navegadores, muchas de esas limitaciones se han conseguido superar, haciendo posible que se ejecuten muchos tipos de juegos en los navegadores. A pesar de todo, no consiguen frenar las barreras de rendimiento que tienen, no logrando de esta forma imponerse sobre el resto de desarrollos.

Pero en definitiva, uniendo lo mencionado con la idea del anterior apartado sobre lo que se intenta encaminar con el nuevo estándar del HTML: es que las aplicaciones que se desarrollan para navegador, no dependan de terceros y que sobre todo no sean tan deficientes en el tema de rendimiento. Para ello proveen de nuevas funcionalidades que posiblemente con el tiempo aún se mejoren mucho más.

Los websockets, el video, el canvas y los demás elementos que se añaden, especialmente las nuevas librerías que permiten nuevas funcionalidades para los scripts que se ejecutan en el navegador, hacen que ahora el desarrollo en el navegador tenga un mayor apogeo y más posibilidades que antes no se permitían.

2.3 Servidores

Existe también en los servidores web una alta competencia para albergar contenidos tanto dinámicos como estáticos, ofreciendo múltiples y muy variadas alternativas a los muchos tipos de usuarios que hoy en día hacen uso de contenidos en internet.

La competencia existente gira en torno a otros tipos de problemas, muy diferentes a lo visto en los anteriores apartados. Están orientados hacia implementaciones más eficientes en cuanto a la concurrencia de manejo de cientos de usuarios y bases de datos globales.

Cada vez se hace más importante el tener en cuenta estos datos al hacer uso de redes más grandes e interconectadas con todo tipo de entornos y dispositivos. Ya no solo se necesita una computadora, pesada y voluminosa. Ni tampoco un portátil de dimensiones más manejables, pero siendo todavía voluminoso. Ahora los dispositivos más compactos y versátiles necesitan cada vez con más frecuencia y mayor uso, contenidos dinámicos que se puedan utilizar en muy diferentes y variadas tecnologías.

Los servidores por tanto necesitan disponer de una mayor funcionalidad. De poder manejar diferentes protocolos y abstraerlos tanto para usuarios como desarrolladores. Se han ampliado en tal amplitud sus metas que los orígenes ya no son comparables a lo que hoy en día se maneja.

En consecuencia, sus virtudes o defectos son menos visibles en comparación, y a raíz de ello, son más difíciles de identificar. Como no es un objetivo primordial, se abstraerá todo lo relacionado con la eficiencia y manejo de datos, porque no es el objetivo el conseguirlo.

No obstante se tienen en cuenta tanto aspectos económicos, como temporales. También se tiene en cuenta la facilidad de uso para el desarrollo, como los posibles conocimientos previos adquiridos.

Estos aspectos quizá sean de los más importantes en el mundo empresarial actual, porque de ellos deriva principalmente el que una empresa pueda sostenerse económicamente. Aunque verdaderamente los otros muchos factores que pueden llegar a decidir una u otra tecnología, influyen de forma drástica en estas características. Ya sea para diferenciarse de los competidores o por hacer que el cliente quede más satisfecho.

Como existe tal cantidad de servidores y de lenguajes que soportan, se han seleccionado los que se consideran más identificativos de un grupo de plataformas también reducida. No significa que sean los únicos de cada tipo y plataforma, sino todo lo contrario. Hay que

considerar que muchos de los servidor que no se mencionan puedan ser los que mejor cumplan con las necesidades, pero como se ha mencionado, se han seleccionado por aspectos económicos, temporales y conocimiento previo adquirido.

- **Plataforma .Net:** Micrososft utilizó esta plataforma para competir y responder a las otras plataformas en auge en su momento, por el año 2002. Se puede considerar que no es multiplataforma y la licencia de uso es de pago, aunque depende de las funcionalidades que se desean comprar. También dispone de un entorno de desarrollo muy completo y que con los años de experiencia es quizá de los mejor preparados.

  Como contrapunto, es muy sencillo desarrollar y tener con facilidad las primeras pruebas. Se caracteriza por ofrecer de una forma simple todas sus funciones.

- **Plataforma PHP:** Existen en el mercado versiones gratuitas para utilizar de modo profesional toda su infraestructura. Al ser su lenguaje de programación interpretado, es quizá de los más eficientes y rápidos en ofrecer contenido, al no tener que precompilar los ficheros.

  No se le puede considerar multiplataforma, pero en la actualidad existen servidores para casi cualquier sistema operativo que puedan tener los servidores. Además existen variados entornos de desarrollo que facilitan su uso, siendo común su sencillo manejo y la licencia gratuita.

- **Plataforma J2EE:** Es quizá la que podría considerarse la más versátil. Al ser multiplataforma, es muy probable que todos los sistemas operativos puedan disponerse de su máquina virtual, que es la que se encarga de ejecutar los archivos compilados.

  Existen muchas versiones para ofrecer contenido dinámico, por suerte algunos de ellos gratuitos son de los más conocidos y usados.

- **Plataforma JavaScript:** Es la plataforma más joven y es anecdótico que se utilice el lenguaje de programación que se usa en el navegador. Su orientación ofrece unos paradigmas algo diferentes a los habituales utilizados en el mercado de los servidores.

  La orientación al evento es el mayor de sus logros, intentado de esta forma que el programador no tenga que hacer uso de hilos para la alta concurrencia. A parte de ser un lenguaje interpretado y orientado al objeto, en un principio debería ser multiplataforma, pero al ser tan joven, las implementaciones aún no lo ofrecen para todos sistemas operativos. Su licencia es gratuita y abierta a colaboración en la mayoría de los casos.

Por otra parte, ofrece algo que los demás no pueden ofrecer: la programación en el mismo lenguaje en ambos lados. La parte del cliente con los scripts que se ejecutan en el navegador, como la parte ofrecida en el servidor.

2.4 Comunicación

La comunicación es un elemento fundamental que en la actualidad se hace indispensable. Las grandes compañías destinan grandes inversiones en proyectos que se apoyan en la comunicación entre diferentes entornos: aplicaciones web, aplicaciones de escritorio que necesitan consultar información privada, aplicaciones para relacionarse con otras personas mediante texto, voz o video, etc.

El uso de este tipo de elementos ha permitido que la comunicación sea algo invisible al usuario y al programador, abstrayendo todo lo relacionado para no confundir en su uso. Para este caso en concreto, se hace un elemento muy importante y esencial. Existen tantos protocolos de comunicación que es fácil no saber cuál se debe utilizar. Pero por el objetivo del proyecto no hay muchas opciones en las que seleccionar la tecnología adecuada.

Basándose en que el objetivo es tener una aplicación web, el protocolo que todo navegador web debe tener es el comúnmente conocido HTTP (siglas del protocolo de transferencia de hipertexto). Cualquier desarrollador web está familiarizado con él, aunque la mayoría de veces como parte de la infraestructura que garantiza la comunicación del servidor con los navegadores. Esto hace que muchas veces toda la comunicación sea transparente. Es el protocolo que se utiliza en la World Wide Web y quien se encarga de estandarizarlo y mejorarlo es el mismo que el del estándar HTML: W3C.

El protocolo está orientado a transacciones, donde el cliente (en este caso el navegador) realiza peticiones. El servidor correspondiente le devuelve la información dependiendo de lo que haya solicitado, dicha petición del recurso solicitado se le identifica con un localizador uniforme de recursos (la URL).

Como originalmente fue ideado para transmitir documentos, o resultados de programas, es destacable que es un protocolo sin estado, sin guardar ninguna información de conexiones anteriores. Muchas de las aplicaciones web que actualmente se utilizan necesitan guardar esta información, para mantener un estado lógico.

Por ejemplo, cuando un usuario se conecta al correo electrónico utilizando el navegador web, es necesario guardar su estado de conectado. Si no sería posible que cualquiera pudiese entrar a revisar su correo. Para solventar esto, tanto los navegadores como los servidores permiten guardar sesiones.

Un punto negativo que tiene el protocolo, y que ya se ha comentado anteriormente, es que la comunicación se realiza en una dirección. El servidor no puede realizar peticiones al

cliente utilizando este protocolo. En realidad podría hacerse, pero los navegadores no son servidores que contemplen recibir peticiones HTTP. Por tanto muchos desarrollos han tenido que ingeniar soluciones para solventar los problemas que le ocasionaba esta limitación.

- **AJAX:** Acrónimo de JavaScript Asíncrono y XML (Asynchronous JavaScript And XML). Permite que por medio de Scripts ejecutados en el navegador, se puedan realizar nuevas peticiones. Produciendo cambios sobre las páginas sin tener que realizar nuevas peticiones de URL. Esto se traduce en aumentar la interactividad y la usabilidad de las aplicaciones web.

  Adicionalmente, si la petición que se realiza tiene que hacer uso de una cantidad importante de información, es preferible trocearla antes que tener que enviar todos los datos de una vez. Y una vez que se presentan los datos de modo reducido o resumido, cuando el usuario lo desee, se muestran las partes de datos que tengan mayor carga, realizando las peticiones en segundo plano y sin interferir sobre la visualización o comportamiento.

  Esta técnica tiene algunos inconvenientes, por ejemplo: es necesario dedicar más tiempo para desarrollar las aplicaciones web, los motores de búsqueda evidentemente no identifican el contenido JavaScript. Tampoco se almacena en el historial de las páginas visitadas al utilizar la técnica y es posible que no sea compatible con software para personas con discapacidades.

  No obstante, muchos de los inconvenientes se intentan solventar de la mejor manera posible, para que interfiera lo menos posible.

- **Comet o Reverse AJAX:** Lo que permite esta técnica es la comunicación del servidor al cliente, dejando que el servidor pueda empezar la comunicación en la que previamente se ha establecido conexión entre ambos.

  Haciendo uso de AJAX, lo que se realiza es una petición http, pero manteniendo abierta la conexión para que el servidor pueda enviar datos al navegador, sin que se soliciten previamente.

- **Websockets:** Con el nuevo estándar, va a aparecer un nuevo elemento denominado Websockets. Permite crear un canal de comunicación que es bi-direccional, economizando recursos y reduciendo latencias.

  Estas dos últimas técnicas permiten que muchas aplicaciones web puedan mejorar y cambiar su filosofía. Por ejemplo, una aplicación de conversación entre varias

personas, herramientas de monitorización, etc. En definitiva, cualquier aplicación que necesite de datos en tiempo real, o datos lo más actuales posibles.

Http no fue diseñado para conservar el estado, pero con el paso del tiempo, las necesidades han ido cambiando y haciendo que lo que en su origen no era posible, sea actualmente una realidad.

Por último, la información que se envía con las tecnologías y protocolos existentes es importante porque tanto el cliente como el servidor deben conocer y entender la información que se envía.

La información compartida en los diferentes entornos debe estar estructurada, para que así si hay cualquier otro entorno que deba utilizarla, sea sencillo o de fácil acceso su entendimiento.

Existen muchos modelos utilizados para compartir la información, pero se escogerán los más representativos como posibles para utilizarlos.

- **XML:** Siglas de Lenguaje de marcas Extensible. Otra vez la W3C se encarga de su estandarización y actualización. Es un metalenguaje extensible de etiquetas, y como HTML es una simplificación de SGML. Pero en este caso en concreto permite definir gramáticas de lenguajes específicos, por eso no es que sea un lenguaje en sí, sino como un metalenguaje que permite definir otros lenguajes.

  Está pensado principalmente como un estándar para el intercambio de información estructurada entre diferentes plataformas. Con lo cual es perfecto para lo que se desea, y muchas empresas lo utilizan para ofrecer sus servicios de manera segura, fiable y fácil.

  Aunque esto último no es compartido por todo el mundo: se le critica por su excesivo nivel de detalle y complejidad, haciendo que los analizadores requieran un tiempo necesario para comprobar que un documento XML está bien formado.

- **JSON:** Es el acrónimo de Notación de Objetos de JavaScript. Se le considera como un formato ligero para el intercambio de información, siendo un subconjunto de la notación literal de objetos que existe en JavaScript.

  De su sencillez y facilidad de uso, ha dado lugar a su proliferación y generalización para sustituirlo en el intercambio de información en la web, en detrimento de XML.

Muchas veces se confunde su sencillez con que es más abreviado que XML, pero en muchos casos esto no es completamente cierto. También se confunde porque se tiene la creencia de que el analizador requiere un menor tiempo en analizar, lo cual no tiene por qué ser cierto.

Por el contrario, sí que es cierto que en muchos casos representa mucho mejor la estructura de los datos, requiriendo menos tiempo para codificar y procesar.

2.5 Sumario

Durante el capítulo se ha podido ver los cambios establecidos con el nuevo estándar de la web que está por llegar y que en parte ya está implementado en muchos de los navegadores actuales. También se ha realizado un somero repaso a la industria del videojuego actual, atendiendo a las categorías o géneros existentes y las características que puede ofrecer cada tipo de implementación.

Después se ha analizado la actual tecnología en referencia a los servidores para ofrecer cualquier tipo de contenido y que existen en el mercado, viendo de forma breve los más relevantes y utilizados. Y ya por último y relacionado con el anterior apartado, se ha visto el tipo de comunicación existente entre el navegador y el servidor.

3 Lógica del juego

Uno de los apartados más importantes a la hora de abarcar un proyecto es la decisión de lo que va a tratar la aplicación, y mucho más si se trata de un juego. Sin tener unas reglas bien definidas siempre el resultado final será un completo desastre. Se hace necesario por tanto un completo análisis de las reglas, el tipo de juego y su ambientación en particular.

Para este caso en concreto, se ha decidido utilizar una mecánica reducida pero lo bastante compleja para que a los jugadores les resulte interesante jugar. En los siguientes apartados se irá explicando todo el proceso para que posteriormente sea más sencillo el poder aplicar lo deseado.

En una partida, el jugador será un rico corredor de apuestas. Deberá competir contra otros apostantes que serán los adversarios. Se deberá aplicar todos los conocimientos necesarios para apostar en combates dónde diversos luchadores participarán en las conocidas peleas de gladiadores. Al finalizar la contienda, el jugador recaudará los beneficios y se comprobará quien es el ganador, siéndolo el que más dinero ha recaudado.

3.1 Generalidades

El jugador debe realizar una serie de apuestas sobre los gladiadores para que luego estos luchen entre si hasta que sólo quede un luchador en pie. Para que los combatientes entren en contienda, los jugadores deben decidir qué tipo de ataque hace cada guerrero. Cada tipo de ataque se define con un color característico de cada carta: verde, amarillo, azul y rojo. Y cada carta tendrá un valor numérico que define el grado de ataque que realiza.

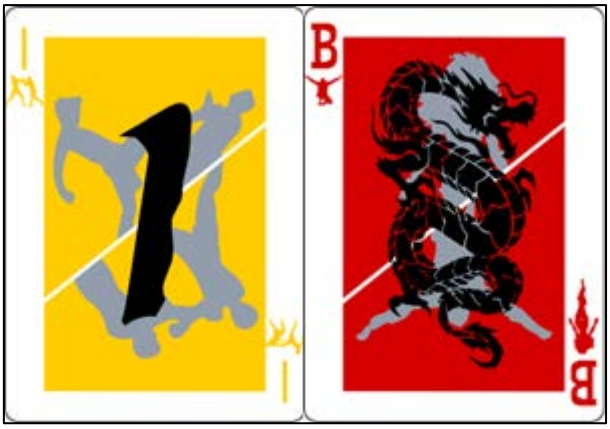

Ilustración 3-1. Ejemplo de cartas.

En una ronda de juego, un jugador decide utilizar una carta y el resto de jugadores deben utilizar el mismo color, siempre y cuando sea posible. Y ya cuando sólo quede un contendiente porque los demás gladiadores han caído, cada jugador recibirá dinero por las apuestas realizadas.

Ilustración 3-2. Ejemplo de fichas de apuestas.

3.2 Objetivo

El objetivo es conseguir el mayor dinero posible dependiendo de las apuestas realizadas sobre los gladiadores. El que consiga el mayor dinero será el ganador una vez que todos los luchadores menos uno sean eliminados del combate.

3.3 Preparación

Lo primero que hay que decidir es el número de jugadores que van a participar, para saber cuántas cartas se reparten a cada jugador y cuantos gladiadores lucharán en este torneo.

Para que una partida sea interesante, el número de jugadores debe oscilar entre 3 y 6 jugadores, aunque podría ser posible que las partidas fuesen de menos o más jugadores, quedando la decisión para el jugador que crea la partida. Aunque siempre habrá un límite mínimo de 2 jugadores y un máximo de 14.

3.3.1 Elegir luchadores

Una vez que ya hay suficientes jugadores para empezar una partida, se decide qué luchadores se enfrentarán entre sí. Los combates son todos contra todos, y no serán combates individuales hasta que sólo queden 2 gladiadores.

El número de contendientes que tendrá el torneo dependerá del número de jugadores, que será a razón de 1 a 2. Es decir, por cada jugador, el torneo tendrá 2 luchadores. Por ejemplo, si hay 4 jugadores, el torneo empezará con 8 combatientes.

Este número es arbitrario y no obligatorio, como el número de jugadores. El creador de la partida es quien decide el número de gladiadores que tendrá la partida, pero siendo el mínimo de 4 luchadores y un máximo de 28.

La elección de los contendientes será aleatorio, pero si se cree necesario se puede dar la posibilidad de que el creador de la partida pueda elegirlos totalmente, o seleccionando algún luchador en concreto y los demás seleccionados aleatoriamente.

3.3.2 Apuestas

Decididos los gladiadores, cada jugador tiene un número de apuestas, cada cual contiene un valor numérico, y que debe ir colocando en 4 rondas consecutivas sobre los guerreros que luchan en el torneo.

Ilustración 3-3. Fichas disponibles para un jugador.

Las fichas de apuestas tendrá los siguientes valores: 0, 1, 1, 2, 3, 5. De esas seis apuestas, cada jugador debe elegir las cuatro que utilizará e ir poniéndolas en cada combatiente.

El jugador que empieza a colocar la primera ficha será el creador de la partida y le seguirán los siguientes jugadores en el orden en el que se han unido a la partida.

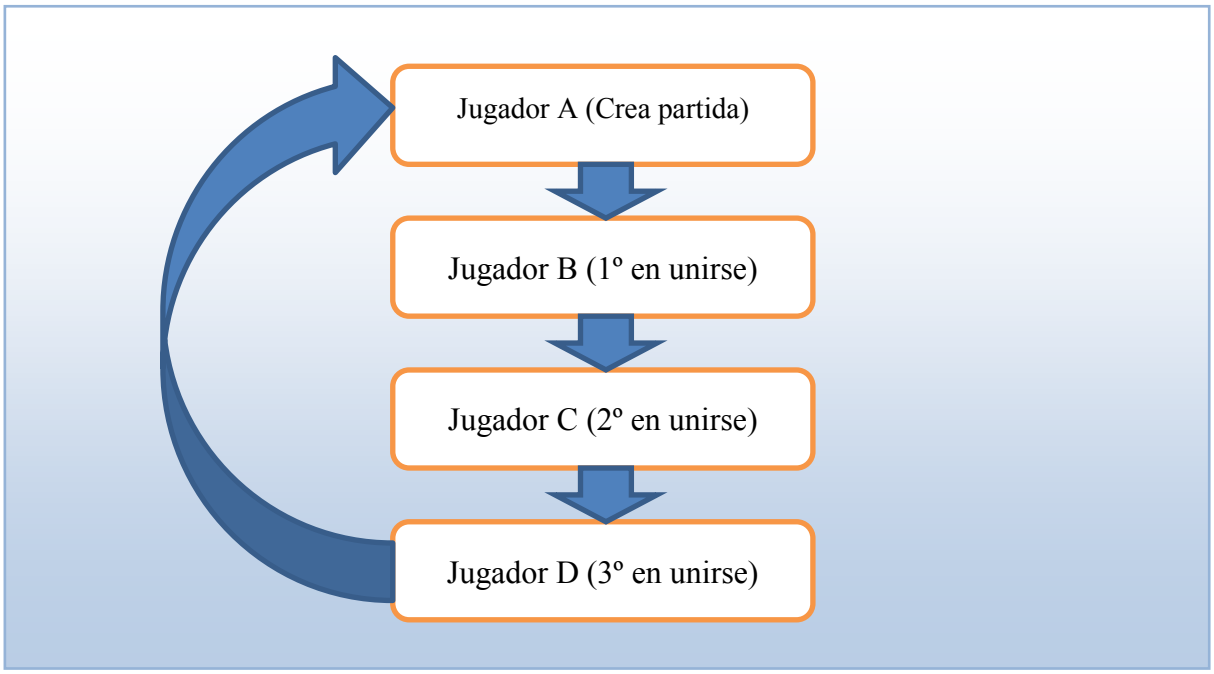

Ilustración 3-4. Orden de juego para colocar apuestas y de la secuencia de juego.

Una vez que todos los jugadores han puesto su primera ficha, empezará la siguiente ronda que irá en el mismo orden que se ha explicado anteriormente. No hay ninguna restricción al colocar las apuestas, se pueden colocar todas las apuestas en un mismo luchador o repartirlas según el jugador lo crea oportuno.

Las apuestas son ocultas para los demás jugadores. Cada jugador sólo conocerá sus apuestas y dónde han colocado los demás jugadores sus fichas, pero no su valor numérico, que se conocerá al finalizar la partida.

Después de haber terminado de colocar las 4 apuestas, a cada jugador le quedarán dos fichas de apuestas, que podrán colocarse cuando caigan los dos primeros combatientes. La primera se colocará cuando cae el primer luchador y la segunda con el segundo. El proceso de colocación de estas dos últimas fichas será el mismo que el explicado en este punto, pero se recordará que estas nuevas apuestas han sido durante la partida, que servirá para saber que su valor será diferente de las iniciales.

3.3.3 Repartir cartas

Después de la colocación de todas las apuestas, darán comienzo las rondas del juego que se explicarán a continuación. Pero antes es necesario repartir las cartas de juego por primera

vez. Dependiendo del número de jugadores se repartirá un número de cartas, porque depende para que el reparto sea equitativo y ningún jugador se quede con más cartas que los demás.

La baraja contiene 4 colores, de los que cada color tiene 10 cartas numeradas del 1 al 10, y 4 cartas denominadas figuras, que se corresponden con las letras de la A a la D. La carta con valor numérico 1 es la de menor valor y la figura con la letra A es la carta de mayor valor. De esta forma el orden sería de la siguiente manera: 1-2-3-...-10-D-C-B-A.

Ilustración 3-5. Cartas ordenadas de menor a mayor del color verde.

Teniendo cada color 14 cartas, en total hay 56 cartas en la baraja para repartir a todos los jugadores. Para el reparto se tendría que utilizar el siguiente procedimiento:

- Se cuenta el total de la baraja, al ser la primera vez ese total corresponde a 56, pero cuando se van repartiendo las cartas ese número decrecerá.
- Se divide el total actual por el número de jugadores. La división correspondiente sería el número de cartas que corresponde a cada jugador.
- Si el número de cartas que corresponde a cada jugador es mayor que 7, es decir 8 o más cartas, entonces se repartirá la mitad a cada jugador.
- Si es menor, se repartirán esas cartas a cada jugador.
- El proceso repetirá los pasos 1 a 4 cuando los jugadores no tengan cartas.

Por ejemplo, al empezar una partida de 4 jugadores, a cada jugador le corresponden 14 cartas, que es el resultado de dividir el total de cartas por el número de jugadores (56 / 4). Como ese valor es mayor que 7, entonces se repartirán 7 cartas a cada jugador.

Una vez que se gasten las 7 cartas de cada jugador, se tendrán que volver a repartir cartas a los jugadores. En este caso, se habrán gastado ya 28 cartas, quedando en la baraja otras 28 cartas. Con este valor a cada jugador le corresponden otras 7 cartas, que es el valor de dividir

el total que queda de la baraja por el número de jugadores (28 / 4). Como este valor no es mayor que 7, se repartirán esas cartas a cada jugador.

Una vez que la baraja no tiene suficientes cartas para repartir a cada jugador, se recogen todas las cartas y se vuelve a barajar para repartir de nuevo desde cero.

3.3.4 Color de cansancio

Después de repartirse las cartas por primera vez, se tiene que decidir cuál es el color que hará que los ataques hagan que se cansen los luchadores involucrados. Los colores que pueden verse afectados son todos menos el rojo. Se decide aleatoriamente entre los 3 colores posibles.

Este color, junto con el rojo, hará que los luchadores se cansen a la hora de pelear en el torneo. En los siguientes apartados se explicará con mayor detalle cómo funciona el proceso. Pero cada vez que la baraja se agote, se tendrá que mezclar de nuevo las cartas y realizar un nuevo reparto. Entonces se tendrá que volver a decidir el nuevo color que afectará a partir de ese momento, de la misma forma que se realiza desde el inicio.

3.3.5 Atributos de luchadores

Antes de explicar cómo funciona el turno de una ronda de juego, es necesario explicar los atributos de los gladiadores, para que se entiendan algunos factores necesarios y que deben saber los jugadores para decidir sus acciones futuras.

Los combatientes disponen de los siguientes atributos:

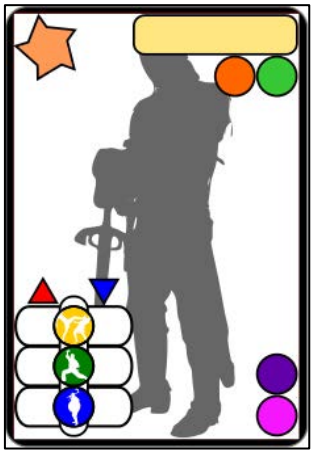

Ilustración 3-6. Ficha que representa a un gladiador.

3.3.5.1 Ataque y defensa

Ilustración 3-7. Ataque y defensa.

Cada personaje dispone de un valor numérico positivo o negativo para cada color, excepto el color rojo. Esta es la habilidad para infringir daño o parar los ataques de los adversarios que tiene el luchador (representados en la figura para los ataques con el triángulo rojo y con el triángulo azul invertido para la defensa).

El valor de ataque es el modificador que utilizará el gladiador a la hora de atacar con el color correspondiente. Y el valor de defensa es el que tendrá que utilizarse si el combatiente es el objetivo de un ataque. Una vez que se saben los datos de ataque y defensa, se utilizarán para modificar los valores de los ataques o de los daños que se causen y que más adelante se explicarán en la secuencia de juego.

Muchas veces, un personaje será mejor para realizar un ataque o para defenderse de un ataque. Lo que quiere decir es que ese personaje está mejor preparado, ya sea por entrenamiento o por nociones naturales, para un tipo de ataque, por eso cada contendiente del torneo tendrá diferentes valores en estos atributos.

3.3.5.2 Cansancio

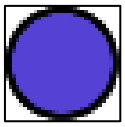

Ilustración 3-8. Cansancio.

Un atributo que tener en cuenta también es el cansancio del personaje. Es un valor numérico positivo, que al principio del juego empieza en cero y cada vez que el luchador realiza alguna acción durante la partida, irá oscilando.

Este valor se comparará con el adversario para saber quien de los dos personajes está más cansado y el grado de diferencia existente. El gladiador atacante, recibirá una penalización o una bonificación si es el que está más cansado o no. El valor de esta modificación se explicará en la secuencia de juego.

3.3.5.3 Recuperar energía

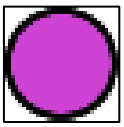

Ilustración 3-9. Recuperar energía.

Este atributo hace referencia al grado de recuperación de cansancio que tiene el luchador. Es un valor numérico positivo que identifica el grado que el personaje recupera energía una vez que la baraja se ha agotado totalmente.

Hará reducir el atributo de cansancio hasta cero y no menos de cero. Cada vez que la baraja o mazo se queda sin cartas y hay que volver a barajar todas las cartas, el gladiador se recuperará del cansancio acumulado hasta el momento, a razón de un punto por punto que tenga en este atributo. Se puede identificar como tomar aliento en algún momento del combate. En la secuencia de juego se explicará con más detalle su funcionamiento.

3.3.5.4 Cotización

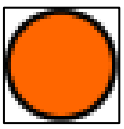

Ilustración 3-10. Cotización.

Es el valor económico del gladiador antes de comenzar el combate, representando lo que presumiblemente se espera de él. Este valor es el que identifica al personaje como un combatiente valioso o no, ya que si ha participado en otros combates y dependiendo de si los ha ganado o perdido, puede ser mayor o menor respectivamente. Al finalizar el torneo, este valor se utilizará para calcular la recaudación de cada jugador y por lo tanto se explicará con más detalle en los últimos apartados.

3.3.5.5 Arrojo

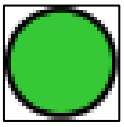

Ilustración 3-11. Arrojo.

Este atributo representa las acciones realizadas por el luchador en el torneo actual, porque el público se agitará más con los gladiadores que más ataques realicen. Hará que los apostantes ganen más dinero por cada luchador que realice con éxito sus acciones de ataque. Y por el contrario, los combatientes que menos hagan por el espectáculo, no obtendrán tantos beneficios, incluso si quedan ganadores de la lucha.

El arrojo es un valor numérico positivo, que empieza en cero y que con las acciones de ataque exitosas irá aumentando. Junto con el atributo de la cotización, se valorará lo que realiza el personaje con lo que presumiblemente puede hacer un combatiente, por tanto se explicará con más detalle en los siguientes apartados.

3.3.5.6 Fortaleza

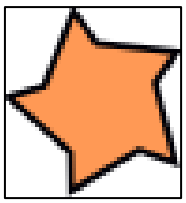

Ilustración 3-12. Fortaleza.

Y por último el atributo que identifica el grado de aguante de recibir golpes que tiene el luchador. Es un valor numérico positivo y que durante la partida se irá reduciendo cuando el luchador es golpeado. Cuando llegue a cero, querrá decir que el gladiador en cuestión es vencido.

3.4 Secuencia del juego

Cuando todos los jugadores tienen sus cartas, la lucha da comienzo. El jugador que creó la partida será el que inicie la primera ronda de la partida, o si se prefiere, el jugador que inicia la partida sea seleccionado aleatoriamente. Tal decisión la toma el creador de la partida al configurar los parámetros necesarios.

En cuanto está decidido quien comienza, la ronda irá pasando de jugador a jugador en el orden en el que se han unido a la partida, como ha ocurrido al colocar las apuestas. Empezará por el jugador designado y le seguirá el siguiente que le corresponda de la lista de los jugadores que se fueron uniendo al torneo.

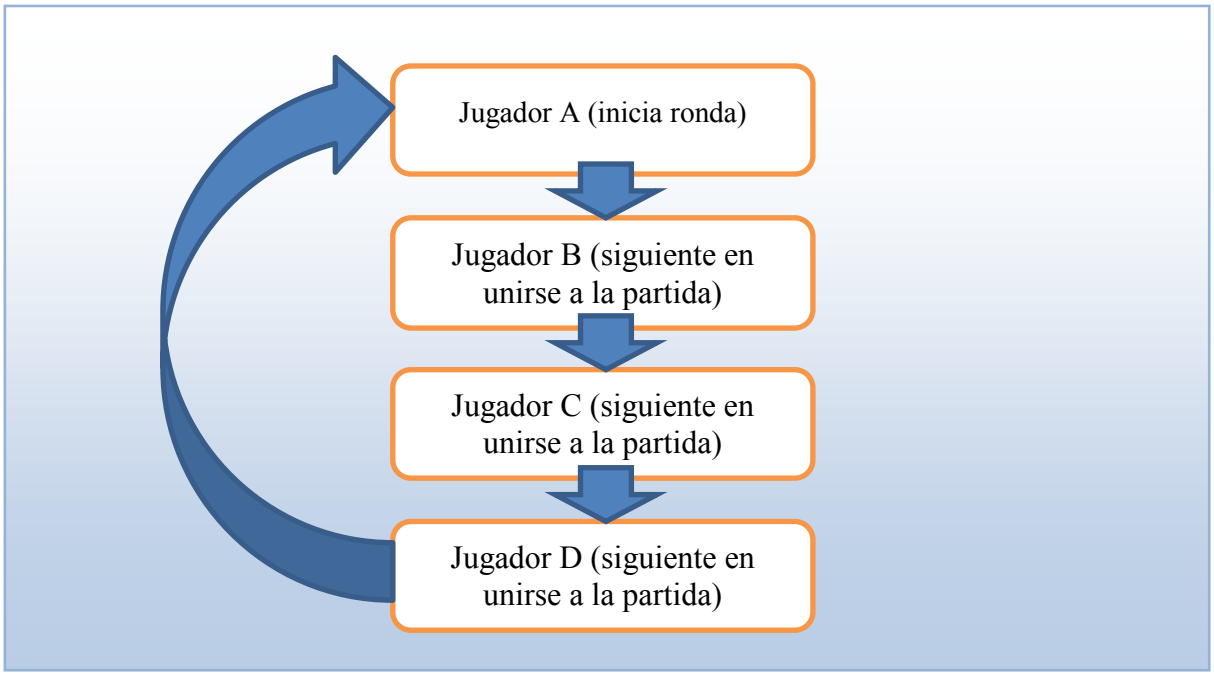

Ilustración 3-13. Orden de juego para la secuencia de juego.

Sabiendo ya el orden de la ronda actual, el primer jugador debe realizar los siguientes pasos:

- Decidir atacante y defensor.
- Jugar una carta.
- Elegir y aplicar modificadores.

Los demás jugadores deberán jugar también una carta y posteriormente se resolverá la ronda para saber qué es lo que ha ocurrido. Por último se decide quién es el que iniciará la siguiente ronda de juego si la hay.

3.4.1 Decidir atacante y defensor

El jugador que empieza la ronda debe seleccionar un atacante y un defensor de los contenientes que hay en el actual torneo. No hay ninguna restricción para seleccionar uno u otro, salvo que el atacante y el defensor no pueden ser el mismo luchador.

3.4.2 Jugar carta por el jugador que inicia la ronda

Después de tener decidido qué luchador ataca y a quién ataca, el jugador debe decidir que carta debe utilizar. Para decidirlo tiene que tener en cuenta los colores correspondientes del atacante y del defensor, utilizando el que más le convenga según las cartas que tenga.

Esto es así, porque la carta utilizada por el jugador que inicia la ronda, es la carta que representa el ataque que realiza el luchador. Cuanto mayor es el valor de la carta, mayor será el golpe que se realiza y más fácil será hacer daño al rival. La carta jugada la deben conocer todos los jugadores y no hay ninguna restricción a la hora de utilizar su carta, puede ser cualquiera que disponga en su mano.

3.4.3 Elegir y aplicar modificadores

Justo después de utilizar la carta es necesario decidir dónde aplicar los modificadores existentes. El jugador inicial debe decidir si los modificadores que se aplican por los atributos de los contendientes se aplican para modificar el valor de la carta jugada, o al daño si el luchador atacante gana la ronda.

Para calcular los modificadores, se debe tener en cuenta la siguiente fórmula:

Modificador = [modificador ataque de atacante] - [modificador defensa defensor] + [factor cansancio]

El *[modificador ataque de atacante]* se corresponde al atributo de ataque para el color seleccionado en esta ronda.

El *[modificador defensa defensor]* se corresponde al atributo de defensa para el color seleccionado en esta ronda.

El *[factor cansancio]* se corresponde al cansancio de los dos combatientes, pero dependiendo del valor de cada atributo de cansancio, su valor podrá cambiar:

- Si el cansancio del atacante como del defensor es el mismo, el valor de *[factor cansancio]* es 0.

- Si el cansancio del defensor es mayor que el del atacante pero no llega al doble, el valor de *[factor cansancio]* es 1.
- Si el cansancio del defensor es el doble o más que el del atacante, el valor de *[factor cansancio]* es 2.
- Si por el contrario, el cansancio del atacante es mayor que el del defensor pero no llega al doble, el valor de *[factor cansancio]* es -1.
- Si el cansancio del atacante es el doble o más que el del defensor, el valor de *[factor cansancio]* es -2.

3.4.4 Resto de jugadores

Después de que el jugador inicial haya utilizado su carta y decidido dónde aplicar el modificador, el resto de jugadores deberán utilizar una carta que representa la protección del contendiente que es defensor. La carta que jueguen la deben conocer todos los jugadores. El orden de juego en este caso es similar al de las apuestas.

Una vez que se inicia la ronda, el siguiente jugador en utilizar una carta es el siguiente en el orden de llegada a la partida. Debe terminar de forma cíclica hasta el jugador que inició la ronda.

El jugador después del inicial debe jugar unas cartas determinadas, que dependen directamente de la carta del jugador inicial:

- Debe usarse una carta del mismo color que la carta utilizada por el jugador inicial.
- Si no dispone de una carta del color, puede usarse cualquier carta.
- Las cartas rojas pueden jugarse como comodín para el color. Es decir, si el jugador dispone de cartas rojas y de cartas del color asignado para la ronda, puede jugar o la carta del color, o la carta roja.

3.4.5 Determinar resultado del combate

En cuanto todos los jugadores han jugado una carta, se debe determinar quien es el gladiador ganador de la ronda: si es el atacante o el defensor. El atacante es representado por la carta del jugador que inició la ronda. El resto de jugadores representan al defensor a la hora de determinar quién es el vencedor.

La carta más alta determinará quién es el ganador, pero sólo se considerarán las cartas con el mismo color a la de la carta inicial. Y además se deben aplicar los modificadores si se decidió aplicarlos al valor de la carta.

Si el vencedor del combate es el atacante, este inflige daño al defensor. El daño es el valor de la carta que representa el ataque más los modificadores si se decidieron aplicar al daño. Las figuras al no tener un valor numérico, se consideran que todas aplican como una carta de valor 11.

Además, cada carta roja jugada, independientemente de quién la jugase, produce daño: si es una carta numérica, produce un daño adicional, si es una figura, produce 2 daños adicionales.

Después de haber calculado el daño total, se resta el valor obtenido a la fortaleza del defensor. Si en este combate el defensor llega a tener una fortaleza de cero o menos, entonces se retira al defensor del combate. No se podrá utilizar más en los combates y todas las fichas colocadas sobre él se retiran junto a él. Tampoco se podrá colocar nuevas fichas sobre el gladiador retirado si aún les quedasen a los jugadores.

3.4.6 Cansancio

A la vez a la determinación el resultado del combate, hay que calcular el cansancio de los contendientes. El luchador que obtiene puntos de cansancio será solamente el vencedor del combate.

Las cartas que cansan serán específicamente las rojas, y las cartas cuyo color sean del mismo que se ha decidido al repartir el mazo de cartas.

Todas las cartas numéricas del color rojo y el seleccionado, harán que el gladiador ganador se canse en una unidad. Las figuras en este caso serán de dos unidades. Pero hay una excepción, la figura de dragón del color seleccionado cansará 10 unidades, para representar que esa acción es más difícil tanto para el atacante como para el defensor.

Una vez identificadas las cartas que producen cansancio, se suman todas junto con el atributo de cansancio del luchador que ha vencido en la ronda.

3.4.7 Repartir puntos

Si la ronda actual culminó con un ataque exitoso, el personaje atacante recibirá una recompensa por el ataque, al hacer que el público enfervorizado se anime aún más. Esto se traduce en aumentar el valor del Arrojo del gladiador, dependiendo del tipo de ataque que se ha realizado:

- Si la carta que representa el ataque tiene un valor del 1 al 5, el valor de arrojo del gladiador aumentará en uno su valor.
- Si el valor se comprende entre el 6 y el 10, el arrojo aumenta en 2.
- Si por el contrario es cualquier tipo de figura, el arrojo aumenta en 5.
- Independientemente de la carta utilizada, si el ataque tiene éxito y vence al contrincante, es decir, la fortaleza del adversario se reduce a cero, el arrojo aumentará en 10 unidades. Pero no es acumulativo, es decir, si un gladiador consigue reducir a su adversario la fortaleza a cero, conseguirá como mucho los 10 puntos anteriormente citados, pero no podrá ser mayor de 10 en ningún caso.

3.4.8 Jugador que inicia las siguientes rondas

Lo último que se realiza es determinar quién iniciará la siguiente ronda de juego para declarar un nuevo ataque. Para ello se comparan las cartas jugadas por los jugadores. El que haya jugado la carta con el valor mayor será el que inicie la nueva ronda, independientemente de la aplicación de cualquier tipo de modificador y color de carta.

En caso de empate, el que empezará será el primero según el orden de juego de la anterior ronda. Es decir, si el jugador que inició la ronda jugó una carta de valor 9, y es la carta más alta, junto a otro jugador que jugó otro 9, el jugador que inició la ronda será el que empezará la siguiente ronda.

3.4.9 Datos a comprobar al finalizar la ronda

Una vez terminados todos los pasos necesarios de la ronda, es necesario decidir algunos aspectos que son parcialmente ajenos a lo que se realiza en la secuencia de juego.

Lo primero es comprobar las cartas de cada jugador. Mientras tengan al menos una carta no será necesario realizar ninguna comprobación adicional. Sin embargo, si ya no tienen cartas, se tiene que realizar un nuevo reparto de cartas.

Para ello se realiza el mismo proceso explicado en la preparación de la partida. Pero si también se ha agotado la baraja, se coge de nuevo la baraja de cartas, se mezclan de nuevo y se realiza un nuevo reparto con el mazo con las 56 cartas que lo compone.

Además, si se ha realizado un nuevo reparto mezclando las cartas, hay que decidir de nuevo el color que produce cansancio. Y todos los personajes recuperarán tantos puntos de cansancio como hay en el atributo de recuperar energía.

Por ejemplo, un luchador tiene el atributo de cansancio en un valor de 25, y el atributo de recuperar energía tiene el valor de 7. Cuando se acaba la baraja, además de repartir las nuevas cartas y decidir el color que produce cansancio en las sucesivas rondas, este personaje recupera 7 puntos de cansancio, quedándose en 18 (25 - 7) el atributo de cansancio.

3.5 Cómo ganar (Pago)

La secuencia del turno acaba cuando ya sólo quede un combatiente y no pueda atacar a nadie más. Entonces se debe calcular las ganancias de los jugadores y determinar el vencedor del torneo. Para ello, es necesario conocer todas las apuestas de los jugadores, el valor de arrojo y la cotización de los gladiadores. También es necesario conocer la posición en la que han quedado los luchadores a lo largo del torneo.

La posición de los contendientes determinará su cotización, que hará que aumente si es de los primeros. Esto es así porque muchas veces puede ocurrir que los que han llegado a las últimas fases del combate sin hacer nada, no obtendrán el beneficio esperado, pero si por el contrario han demostrado su valía, obtendrán más dinero.

Por tanto, para los que han quedado 1º, 2º y 3º, su cotización aumentará para este torneo en 10, 8 y 6 respectivamente.

Después, junto con el arrojo que han obtenido los personajes durante el combate, se calculará si han conseguido llegar hasta la cotización y si el dinero que se recauda para el combate será positivo y negativo. Si un gladiador no ha conseguido por lo menos igualar su arrojo a su cotización, entonces el dinero que se recauda por ese luchador será su cotización por la mitad.

Si un gladiador no ha quedado entre los 3 primeros y su arrojo no llega al menos igualar a su cotización, entonces no se recauda dinero. En este caso los apostantes que tengan sus apuestas en ese luchador obtendrán pérdidas iguales a la mitad de su cotización.

Ya sabiendo lo que se recauda por cada gladiador, para cada jugador se calculará su dinero con las apuestas que ha realizado. Mientras un jugador tenga al menos una ficha de apuesta sobre un luchador, obtendrá mínimo el dinero que se ha calculado.

Ahora bien, cuando hay varias apuestas sobre un contendiente, se verá que cada uno obtendrá beneficios dependiendo de las fichas y el valor de cada una de esas fichas:

- Para el jugador que tenga más fichas, independientemente de su valor, se obtendrá 3 puntos más de beneficio, 2 para el segundo y 1 para el tercero. El resto no recibirá nada y si hay cualquier empate, los que empatan se llevarán la misma recompensa.

  Por ejemplo, sobre un gladiador hay dos jugadores que tienen 3 fichas, un jugador que tiene 2 fichas y otros dos jugadores tienen 1 ficha. Los dos que tienen 3

fichas, obtendrán 3 puntos más de beneficio, el que tiene 2 fichas obtendrá otros 2 de beneficio y los que tienen una ficha, obtendrán 1 más de beneficio.

No obstante, si hay cualquier tipo de empate en el que haya un jugador que tenga más fichas que no ha colocado al principio, es decir, que las ha colocado cuando ha sido eliminado algún luchador, se considerará que queda por delante el que más fichas tenga colocadas al inicio y no durante el torneo.

- Ocurrirá lo mismo para el valor de la apuesta. Para el que tenga el valor mayor sumando la cifra numérica de cada apuesta, obtendrá 3 puntos más de beneficio, 2 para el segundo y 1 para el tercero.

  Si ocurre cualquier tipo de empate, quedará primero el que tenga más fichas colocadas al principio, pero si persiste el empate, a los dos se les considerará en el mismo puesto.

Habrá también dos inconvenientes para los gladiadores que más fichas tengan:

- El luchador que tenga el mayor número de fichas, independientemente de quién sea, el beneficio base que se obtenga será reducido a la mitad, tanto si es positivo como negativo. Esto representa que hay mucha gente que ha apostado a la misma persona y por tanto sus beneficios se tienen que repartir entre todos.

  Si hay algún empate, ocurrirá lo mismo para todos los gladiadores involucrados.

  Además, si el valor ya se ha reducido a la mitad no se volverá a reducir a la mitad.

- El gladiador que tenga el mayor número de jugadores que han apostado sobre él también se reducirá a la mitad el beneficio que se obtengan de él, independientemente de que sea positivo o negativo. Si hay algún empate, se aplicará lo mismo a todos los gladiadores que tengan el mayor número de jugadores.

3.6 Ejemplos

3.6.1 Ejemplo 1: Preparación de partida.

Guillermo crea una nueva partida. Ha tenido que decidir antes alguno de los parámetros, por ejemplo que el número de jugadores esté entre 3 y 6. También el número de luchadores, el cuál será por cada jugador que haya en la partida: habrá 6 combatientes. Además elige que los luchadores sean completamente aleatorios por desconocerlos completamente.

Una vez creada, los usuarios que se conecten podrán ver las partidas que hay creadas o crear ellos mismo una. En este caso, se van uniendo a la partida varios jugadores. Primero se une Ester, después Néstor y a continuación Nuño. Aunque no se ha alcanzado el límite, Guillermo decide comenzar la partida ya, quedando cerrado definitivamente y con el siguiente orden para la siguiente fase:

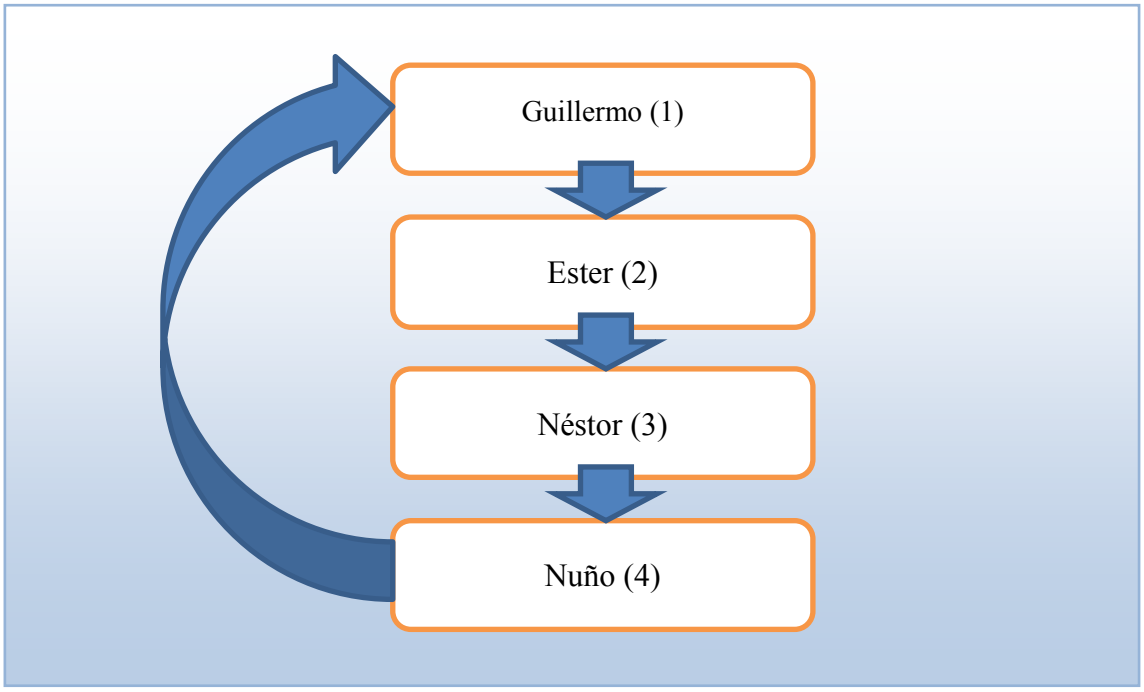

Ilustración 3-14. Orden de juego.

Como se ha seleccionado completamente aleatoria la elección de luchadores, se escogen 8 gladiadores y cada jugador los observa detenidamente para saber cuáles son los que mejor les vienen para combatir.

A continuación empieza la fase de colocación de apuestas. Primero deben seleccionar las 4 de las 6 apuestas que quieren colocar a los contendientes. Guillermo decide seleccionar las fichas con valor 0, 1, 2, y 5. Dejará para rondas posteriores las apuestas con valor 1 y 3. Los demás jugadores deberán seleccionar las suyas y esperar que les toque su turno para colocarlas.

Como es el creador de la partida, debe comenzar la ronda de colocación de apuestas. Decide colocar la apuesta con valor 2 sobre el luchador A. El siguiente jugador que debe colocar su apuesta es Ester, y decide colocar una ficha sobre el gladiador B, sin que los demás sepan el valor que tiene. Es el turno de Néstor y decide colocar una ficha sobre el luchador A. Nuño por el contrario coloca su ficha sobre el luchador C.

Cuando se ha acabado la ronda, vuelve a empezar una nueva con el mismo proceso, empezando por Guillermo y acabando en Nuño. En cuanto se colocan todas las apuestas, se dará comienzo a la partida.

Antes de empezar se deben repartir las cartas y elegir el color que hará que los personajes se cansen. Como son 4 jugadores, a cada jugador le corresponden 7 cartas y se les repartirá ese número de las 56 que hay en total de forma aleatoria.

Por último, se selecciona el color que producirá cansancio para las rondas que existan hasta acabar con las 56 cartas. Cuando se vuelvan a barajar y repartir de nuevo, se tendrá que volver a escoger un nuevo color. El color que sale de modo aleatorio de los tres posibles es el Amarillo.

Una vez decidido el color, dará comienzo a las secuencias del combate, que irá cada jugador ronda tras ronda utilizando sus cartas.

3.6.2 Ejemplo 2: Secuencia de juego.

Continuando con el ejemplo anterior, Guillermo al crear la partida dejó la opción por defecto de que el que crea la partida inicia la primera ronda de juego. Así que es él quien inicia la ronda.

Decide atacar con el gladiador A al gladiador B y a continuación usa una carta azul, la cual es una figura con valor B. Antes de que el siguiente jugador use una carta, hay que calcular el modificador existente entre los dos combatientes elegidos, por si hubiese un modificador positivo o negativo que deba aplicarse.

En este caso, el atributo de ataque que tiene para las cartas azules el contendiente A, es un +1. Después está el atributo de defensa que tiene el combatiente B, que es un +2. Y por último tanto el luchador A como el B tienen cansancio 0, porque aún no han realizado ninguna acción y por tanto no se aplica ningún modificador.

El resultado de los modificadores es de: *1 - 2 + 0 = -1*. Tiene un modificador negativo, que Guillermo decide usarlo para que se modifique el daño y no el impacto. Esto quiere decir

que para que el defensor gane, alguno de los otros jugadores debe jugar una carta con valor mayor que la carta jugada por Guillermo.

En el orden establecido, la siguiente en jugar una carta es Ester. Como no tiene cartas azules, decide jugar una carta roja, una figura de valor A. El siguiente en jugar es Néstor, que como le interesa que gane el defensor y al disponer de una carta superior y del color azul, juega la figura con valor A. El último es Nuño que visto que es imposible que inicie la siguiente ronda y de no disponer de una carta azul, decide jugar una carta amarilla, de valor 4.

Cuando todos han jugado sus cartas, se decide el ganador del combate y el resultado de la ronda. En este caso es el gladiador B, porque la carta superior del color inicial la jugó un jugador que no era el que empezó la ronda.

Como ha sido el defensor el ganador, no se producen daños, pero si existe cansancio porque se han jugado cartas rojas y cartas del color que realiza cansancio. Como la carta roja es una figura, produce 2 puntos de cansancio adicionales. Y la carta amarilla es una de valor numérico que produce 1 punto adicional.

El cansancio total es de 3 que se le añade al que ya tiene. Como es la ronda inicial, el atributo de cansancio del luchador B es de cero, dando como resultado final un cansancio del gladiador de 3 puntos.

Lo último que hay que hacer es ver quién iniciará la siguiente ronda. Esto se decide con la carta más alta que se ha jugado, que en este caso es la carta jugada por Ester. Ella iniciará la siguiente ronda y continuará Néstor, luego Nuño y por último Guillermo.

3.6.3 Ejemplo 3: Secuencia de juego.

Siguiendo con el ejemplo, tras varias rondas de juego, Néstor inicia la ronda de juego y decide atacar con el luchador C al luchador B. Usa la última carta que le queda, una carta amarilla con valor 9.

Los modificadores para este combate son por parte del atacante un +1 para los ataques amarillos. Para el defensor en las cartas amarillas tiene un -2. Y el factor de cansancio es de +2, porque el gladiador C tiene el atributo de cansancio con 7 puntos y el gladiador B tiene 15 puntos de cansancio. Como el defensor está el doble o más de cansado que el atacante, el factor es un valor positivo de 2 puntos.

Con lo que el resultado del modificador es de: $1 - (-2) + 2 = 5$. Néstor decide utilizar este modificador para el impacto, así de esta forma si otro jugador quiere que gane el defensor

es necesario que juegue la carta amarilla con la figura A. Esto es así porque al 9 jugado por Néstor hay que sumarle los 5 puntos, quedando como resultado una figura A: *9 --> 10 (+1) --> D (+2) --> C (+3) --> B (+4) --> A (+5)*.

El siguiente en jugar una carta es Nuño, que decide jugar su última carta siendo una verde de valor 8 que no afecta en nada al combate. Después juega Guillermo su última carta que es una de color rojo, una figura de valor C. Por último le toca a Ester que juega la carta amarilla que le queda, una figura de valor B, que no es suficiente para que gane el defensor, pero si para iniciar el siguiente turno de juego.

Como ya han jugado todos sus cartas, se tiene que decidir el resultado del combate. En este caso gana el atacante porque nadie ha podido igualar o superar la carta junto con el modificador que se le aplica.

El daño que se produce es igual a la carta sin aplicar ningún modificador, en este caso el daño es de 9. Pero como se ha jugado una carta roja, también produce daño independientemente de quien jugó la carta. Como es una figura el daño aumenta en 2 puntos, quedando en un daño total de 11 puntos.

La fortaleza del luchador B es de 8 puntos, por lo tanto es retirado del combate al no tener suficientes puntos de fortaleza para resistir el ataque. Su fortaleza ahora es de 0, se retiran junto a él todas las fichas de apuestas que tenía y ya no se puede utilizar para realizar ataques o recibir, como tampoco se le pueden asignar nuevas apuestas.

Hay que calcular el cansancio, en este caso sí que hay cansancio porque se han jugado tantas cartas rojas como las cartas del color que cansa, que es el amarillo. El cansancio en este caso va para el atacante por haber conseguido un ataque exitoso.

El cansancio total es de 2 puntos por la figura roja, 1 punto por la carta numérica amarilla y 10 puntos por la figura amarilla (esto es así porque la figura B cansa más que el resto). El total es de 13 puntos de cansancio que se suman a los que ya tenía, quedándose en un total de 20.

Después, como ha habido un ataque exitoso, se le asignan puntos de Arrojo. Como es una carta de valor numérico comprendido entre el 5 y el 10 sin contar modificadores, se le deberían asignar 2 puntos. Pero esto no es así porque el personaje defensor ha sido eliminado de la partida y por tanto se le asignan 10 puntos más al Arrojo que ya tenía, teniendo un total de 13 puntos.

Como aún hay al menos 2 contendientes, lo último por hacer en la ronda es la asignación del siguiente jugador en comenzar el turno próximo. En este caso será Ester en iniciar el turno sucesivo, al tener la carta más alta sin aplicar modificadores.

Pero antes de iniciar el siguiente turno se comprueba que los participantes no tienen cartas, así que es necesario repartir nuevas cartas. En este caso se hace el mismo proceso que en la preparación de la partida, porque la baraja se ha agotado: Se barajan de nuevo las 56 cartas, se reparten 7 cartas a cada jugador y se decide aleatoriamente el nuevo color de cansancio, que será el verde hasta que se vuelva a agotar la baraja. Además, todos los luchadores recuperan tantos puntos de cansancio como el valor que tengan en el atributo de recuperar energía.

3.6.4 Ejemplo 4: Calcular ganancias.

Tras unas cuantas rondas de juego, puede llegar un momento en que sólo queda un único luchador en el juego. Cuando ocurra, se terminará la partida y se pasa a realizar los cálculos para que cada jugador recaude sus ganancias en la partida.

En una partida que acaba de finalizar, el orden en que han sido vencidos cada uno de los gladiadores es el que se muestra en la figura, junto con todas las apuestas que se han realizado por parte de los jugadores:

Ilustración 3-15. Apuestas realizadas por 4 jugadores sobre 4 gladiadores.

Para empezar, los gladiadores que han quedado en primer, segundo y tercer puesto aumentan su cotización en 10, 8 y 6 respectivamente. Entonces la cotización queda 22, 20 y 20 para los luchadores A, B y C.

Ahora para todos los luchadores se compara el arrojo con su cotización. Si el arrojo iguala o supera su cotización, todo apostante que haya realizado al menos una apuesta sobre él,

obtendrá el valor de cotización. Si no ha conseguido cumplir con sus expectativas, es decir el arrojo no consigue por lo menos igualar su cotización, los beneficios son la mitad. Pero además si no está entre los 3 primeros, no solo se reduce a la mitad, sino que es negativo, considerando que ha obtenido pérdidas para ese luchador.

Teniendo en cuenta esto, tanto el gladiador A y el B consiguen superar su cotización, por lo tanto se recibirá como beneficio su valor íntegro: 22 y 20 respectivamente. El combatiente C ha llegado en un principio cumplir con las expectativas, pero al quedar tercero, no lo ha conseguido cumplir. Entonces se reducen los beneficios a 10 para los apostantes que colocaron al menos una ficha sobre él.

El último contendiente D, tampoco ha conseguido cumplir con lo que se esperaba de él. Pero como además no ha conseguido estar entre los 3 primeros puestos, entonces son pérdidas lo que cada apostante obtiene de este gladiador: cada apostante tendrá unas pérdidas por valor de 6 (la mitad de su cotización 12 / 6).

También se tiene que tener en cuenta a los gladiadores que más apuestas han tenido. Tanto en el número de fichas como en el número de jugadores. El gladiador B es el que más fichas tiene, sumando hasta nueve fichas, por lo tanto su cotización en vez de ser 20 se reduce a 10.

Y hay un empate para el mayor número de jugadores, siendo 4 los que más apuestas han tenido tanto en el gladiador A y el B. Como el B ya ha reducido a la mitad los beneficios por ser el que más fichas tiene apostadas, no se vuelve a reducir. Pero el A si que se reduce, quedando en 11 el beneficio que obtienen cada uno de los apostantes.

En la siguiente tabla se muestran los beneficios de cada uno de los apostantes sobre los gladiadores. Se tiene en cuenta la cotización base obtenida y si hay más de un apostante sobre un gladiador: el beneficio que se obtiene por ser el jugador que más fichas tenía sobre el gladiador y la suma del valor de cada ficha que tiene en el gladiador en cuestión.

		Jugador Amarillo	Jugador Verde	Jugador Naranja	Jugador Azul
Gladiador A	Base	11	11	11	11
	Más fichas	3	2	3	3
	Más suma	0	1	2	3
	Total	**14**	**14**	**16**	**17**
Gladiador B	Base	10	10	10	10

	Más fichas	3	2	1	1
	Más suma	3	1	2	0
	Total	**16**	**13**	**13**	**11**
Gladiador C	Base	10	10	10	0
	Más fichas	3	3	3	0
	Más suma	3	3	2	0
	Total	**16**	**16**	**15**	**0**
Gladiador D	Base	0	-6	-6	-6
	Más fichas	0	-3	-3	-3
	Más suma	0	-1	-3	-2
	Total	**0**	**-10**	**-12**	**-11**
Suma totales		46	33	32	17

Tabla 3-1. Beneficios obtenidos por los apostantes.

3.7 Sumario

En este capítulo se ha podido analizar la lógica del juego que se desea utilizar, describiendo desde el objetivo final hasta el resultado final en el que el usuario obtiene sus beneficios. Además se incluyen unos ejemplos de juego que sirven para clarificar y entender mejor lo que se ha explicado anteriormente.

4 Estudio de viabilidad y análisis del sistema

Conocido en gran medida el entorno y las directrices para abordar el proyecto, es indispensable en este momento realizar un análisis del conjunto de necesidades deseadas. Pero siempre teniendo en cuenta ciertas problemáticas relacionadas con las restricciones existentes, como por ejemplo necesidades económicas, temporales o limitaciones existentes en el software actual.

Para realizar el análisis ha sido necesario obtener información de los requerimientos tanto de la lógica del juego como de la documentación existente. Sin olvidarnos de los conocimientos adquiridos a lo largo de las diferentes experiencias en distintos entornos y apartados, para poder aplicarlo haciendo uso de las mejores técnicas posibles.

4.1 Estudio del proyecto

El sistema que se desea desarrollar se basa en la realización de un juego interactivo entre diferentes usuarios, haciendo uso de algunas de las nuevas especificaciones existentes en el estándar que se está desarrollando para HTML.

Una de las especificaciones más relevantes es el uso de los websockets, que permiten la comunicación bi-direccional y que resolvería muchos de los problemas a la hora de conseguir este objetivo.

Por otro lado, al existir tantas plataformas diferentes y sobretodo tantos navegadores existentes, se hace necesario una simplificación, porque no existe ni el tiempo, ni los conocimientos, ni el personal para abarcar el funcionamiento en cada uno de los navegadores existentes. Pero siempre que sea posible, al intentar reducir la lista de navegadores que puedan usarse, si siempre se consigue cumplir con las estandarizaciones, se logra de esta forma que pueda funcionar en cualquier navegador.

Si además se hace uso de las librerías que más se adapten al proyecto y en cuyas especificaciones se mencionen explícitamente que pueda funcionar en el mayor número de navegadores, ayudará a alcanzar el objetivo.

También, con la lógica ya descrita, un usuario podrá crear partidas nuevas con algunos parámetros iniciales o podrá unirse a una partida que acaba de crear otro usuario. Asimismo, un usuario en una partida deberá realizar unas apuestas sobre unos luchadores para que cuando finalice la lucha obtenga beneficios.

No obstante, para conseguir beneficios, durante una secuencia de juego el usuario podrá decidir las acciones de estos luchadores para así definir las posiciones de cada uno de los gladiadores.

4.2 Establecer el alcance

Con los objetivos claramente expuestos en el primer apartado, con las especificaciones existentes en la actualidad y que se han explicado de manera breve en el apartado segundo, y con la lógica bien definida en el tercer apartado, se pueden establecer las funcionalidades del sistema, que serán necesarias para determinar claramente el alcance que pueda tener la aplicación, para así intentar no llevar a equivocación.

A continuación se detallan las funcionalidades que deben ofrecerse:

- El usuario necesita registrarse para poder jugar cualquier partida.
- El usuario deberá identificarse en la aplicación para jugar cualquier partida.
- El sistema mostrará las partidas para que los usuarios identificados se puedan unir si aún no comenzaron.
- El sistema será online haciendo uso de un navegador web compatible.
- El sistema hará uso de las nuevas especificaciones del estándar HTML5, pero siendo compatible con el estándar actual siempre que sea posible.
- El usuario podrá crear partidas nuevas.
- El sistema establecerá al iniciar una partida un orden de juego que se corresponde al orden de llegada a la partida de los usuarios.
- El sistema repartirá cartas a todos los usuarios de una partida, siempre que estos no tengan cartas.
- El sistema mostrará a todos los usuarios participantes de una partida los gladiadores que están en esa partida, con su estado actual si cambia.
- Una vez que una partida se ha iniciado, el usuario podrá realizar 4 apuestas sobre los gladiadores, que deben colocar en el orden de juego.
- El usuario si inicia una ronda de juego deberá elegir un gladiador atacante y defensor.
- Una vez elegido atacante y defensor, el usuario que inició ronda deberá elegir dónde aplicar el modificador existente.
- Una vez elegido dónde aplicar el modificador, el usuario que inició ronda deberá jugar una carta.
- Después del usuario que inicia ronda, los otros usuarios de la partida deben jugar cartas según el orden de juego.
- El sistema avisará a los usuarios de si la carta que ha jugado es correcta.

- El sistema calculará el resultado de una ronda de juego mostrando el resultado y asignando un nuevo usuario para que inicie ronda.

- El sistema, cuando finalice una partida, calculará los beneficios de los usuarios mostrando su resultado a cada uno de ellos.

4.3 Definición y establecimiento de requisitos

Se procede a explicar y especificar los diferentes tipos de requisitos solicitados al establecer el alcance de la aplicación, así como una breve descripción de su identificación, uso y aplicación. Sobretodo se han tenido en cuenta las funcionalidades que están directamente relacionadas con el diseño y que en posteriores apartados serán más necesarios.

Cada requisito se constituye de los campos que a continuación se listan:

- **Identificador:** Código alfanumérico único que identifica al requisito y que sigue el siguiente esquema:

  ➢ RU-Unnn: Para los Requisitos de Usuario, que son los que engloban las necesidades que se deben proporcionar y especificadas en el alcance del sistema.

  ➢ RS-Tnnn: Para los Requisitos Software, que son las funcionalidades, necesidades y restricciones del sistema y sirve como una descripción completa del comportamiento del sistema.

  Dónde:

  - ❖ U: Especifica el subtipo de requisito de usuario, que en concreto será de Capacidad (C) o Restricción (R).
  - ❖ T: Especifica el subtipo de requisito software, que en concreto los valores que puede tomar son:
    - ✓ F: Requisitos Funcionales: Este tipo de requisitos engloban a todos los aspectos necesarios que debe tener la aplicación.
    - ✓ I: Requisitos de Implantación: Estos requisitos hacen referencia a las necesidades existentes para que la aplicación se ejecute correctamente.
    - ✓ R: Requisitos de Recursos: Estos requisitos se refieren a los recursos necesarios para el correcto uso de la aplicación.
  - ❖ nnn: Es el número secuencial que identificará al requisito (1 a 999).

- **Nombre:** Es el nombre que identifica al requisito, debiendo ser un nombre lo más simple y sencillo posible, pero completo para poder representarlo bien y que con sólo leerlo se le pueda identificar.

- **Prioridad:** Indica la importancia que pueda tener el requisito con respecto a los demás y a la importancia en el proyecto. Su valor está comprendido entre Alto, Medio y Bajo.

- **Necesidad:** Indica la obligación que puede tener el requisito, sobre todo para indicar si el requisito es deseable cumplir bajo ciertas circunstancias, como por ejemplo temporales o falta de recursos. Su valor está comprendido entre Alto, Medio y Bajo.

- **Descripción:** Representación detallada de lo que aborda el requisito, intentando que se detalle de manera breve y sencilla.

4.3.1 Requisitos de Usuario

Identificador:	RU-C001	Nombre:	Registrarse
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario necesita registrarse para poder realizar cualquier acción.		

Tabla 4-1. RU-C001.

Identificador:	RU-C002	Nombre:	Identificarse
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario deberá identificarse para poder realizar cualquier acción.		

Tabla 4-2. RU-C002.

Identificador:	RU-C003	Nombre:	Mostrar partidas
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá visualizar todas las partidas que aún no han comenzado.		

Tabla 4-3. RU-C003.

Identificador:	RU-C004	Nombre:	Crear partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá crear partidas nuevas.		

Tabla 4-4. RU-C004.

Identificador:	RU-C005	Nombre:	Unirse partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá unirse a una partida que no haya comenzado.		

Tabla 4-5. RU-C005.

Identificador:	RU-C006	Nombre:	Orden de juego
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá visualizar el orden de juego de la partida que esté jugando.		

Tabla 4-6. RU-C006.

Identificador:	RU-C007	Nombre:	Cartas de partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá visualizar las cartas que tiene disponible en la partida que esté jugando.		

Tabla 4-7. RU-C007.

Identificador:	RU-C008	Nombre:	Luchadores de partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá visualizar los luchadores de la partida que esté jugando.		

Tabla 4-8. RU-C008.

Identificador:	RU-C009	Nombre:	Colocar apuestas
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario colocará las apuestas que tiene disponibles sobre los luchadores al iniciarse la partida en la que está participando.		

Tabla 4-9. RU-C009.

Identificador:	RU-C010	Nombre:	Apuestas de partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá visualizar las apuestas que ha realizado en la partida que esté jugando.		

Tabla 4-10. RU-C010.

Identificador:	RU-C011	Nombre:	Atacante y defensor
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario deberá elegir un atacante y un defensor si inicia una ronda de juego en la partida que participa.		

Tabla 4-11. RU-C011.

Identificador:	RU-C012	Nombre:	Aplicar modificador
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario seleccionará a qué aplica el modificador, si al impacto o al daño, cuando inicia una ronda de juego en la partida que participa.		

Tabla 4-12. RU-C012.

Identificador:	RU-C013	Nombre:	Jugar carta
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario jugará una carta de las que tiene disponibles, una vez por ronda.		

Tabla 4-13. RU-C013.

Identificador:	RU-C014	Nombre:	Aviso de carta jugada
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario visualizará si la carta jugada es la correcta.		

Tabla 4-14. RU-C014.

Identificador:	RU-C015	Nombre:	Resultado ronda
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario visualizará el resultado de la ronda una vez finalizada, con la asignación del nuevo usuario que iniciará ronda.		

Tabla 4-15. RU-C015.

Identificador:	RU-C016	Nombre:	Resultado partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario al finalizar la partida visualizará el resultado de la partida, viendo los beneficios obtenidos por cada usuario.		

Tabla 4-16. RU-C016.

Identificador:	RU-R017	Nombre:	Interfaz web
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá ser online siendo operativo bajo un navegador web compatible, como Mozilla Firefox o Google Chrome.		

Tabla 4-17. RU-R017.

Identificador:	RU-R018	Nombre:	Uso estándar
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá hacer uso de la nueva especificación del estándar HTML5.		

Tabla 4-18. RU-R018.

Identificador:	RU-R019	Nombre:	Uso otro navegador
Prioridad:	Baja	Necesidad:	No
Descripción:	El sistema deberá ofrecer las mismas funcionalidades a otro tipo de navegadores que sea posible.		

Tabla 4-19. RU-R019.

Identificador:	RU-R020	Nombre:	Uso estándar anterior
Prioridad:	Baja	Necesidad:	No
Descripción:	El sistema ofrecerá las mismas funcionalidades para navegadores que no tengan disponible el estándar HTML5.		

Tabla 4-20. RU-R020

4.3.2 Requisitos Funcionales

Identificador:	RS-F001	Nombre:	Registrarse
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá permitir la creación de nuevas cuentas registradas mediante una interfaz de acceso.		

Tabla 4-21. RS-F001.

Identificador:	RS-F002	Nombre:	Identificarse
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema permitirá al usuario autenticarse en la aplicación.		

Tabla 4-22. RS-F002.

Identificador:	RS-F003	Nombre:	Salir
Prioridad:	Alta	Necesidad:	Si

Descripción:	El sistema permitirá al usuario salir de la aplicación siempre que lo desee, haciendo uso de la opción correspondiente.

Tabla 4-23. RS-F003.

Identificador:	RS-F004	Nombre:	Mostrar partidas
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema mostrará a los usuarios las partidas que han sido creadas pero que aún no han comenzado.		

Tabla 4-24. RS-F004.

Identificador:	RS-F005	Nombre:	Crear partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá permitir al usuario crear una nueva partida.		

Tabla 4-25. RS-F005.

Identificador:	RS-F006	Nombre:	Unirse partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá permitir al usuario unirse a una partida creada previamente.		

Tabla 4-26. RS-F006.

Identificador:	RS-F007	Nombre:	Orden de juego
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar el orden de juego de una partida a sus participantes.		

Tabla 4-27. RS-F007.

Identificador:	RS-F008	Nombre:	Cartas partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar las cartas de un usuario que está en una partida.		

Tabla 4-28. RS-F008.

Identificador:	RS-F009	Nombre:	Luchadores partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar los luchadores de una partida a sus participantes.		

Tabla 4-29. RS-F009.

Identificador:	RS-F010	Nombre:	Colocar apuestas
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema permitirá colocar apuestas a los jugadores en el orden de juego.		

Tabla 4-30. RS-F010.

Identificador:	RS-F011	Nombre:	Control apuestas inicio
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema controlará el número de apuestas que coloca cada usuario cuando se inicia la partida, siendo obligatorio colocar 4 apuestas.		

Tabla 4-31. RS-F011.

Identificador:	RS-F012	Nombre:	Control apuestas ronda
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema controlará la apuesta que se coloca cada vez que un luchador es vencido, siempre que al usuario de la partida aún le queden apuestas por colocar.		

Tabla 4-32. RS-F012.

Identificador:	RS-F013	Nombre:	Apuestas colocadas
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar las apuestas realizadas por el usuario en una partida.		

Tabla 4-33. RS-F013.

Identificador:	RS-F014	Nombre:	Apuestas sin colocar
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar al usuario las apuestas que aún no ha colocado durante cualquier partida.		

Tabla 4-34. RS-F014.

Identificador:	RS-F015	Nombre:	Atacante y defensor
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema permitirá al usuario que inicia una ronda de juego de una partida, elegir un luchador atacante y un luchador defensor.		

Tabla 4-35. RS-F015.

Identificador:	RS-F016	Nombre:	Aplicar modificador
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema permitirá al usuario que inicia una ronda de juego de una partida, elegir el modificador existente, si al impacto o al daño, después de elegir atacante y defensor.		

Tabla 4-36. RS-F016.

Identificador:	RS-F017	Nombre:	Jugar carta
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema permitirá al usuario jugar una carta de las que tenga disponibles, cuando le toque en una ronda de juego.		

Tabla 4-37. RS-F017.

Identificador:	RS-F018	Nombre:	Aviso carta jugada
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar un aviso si la carta jugada por un usuario en una partida es correcta o errónea.		

Tabla 4-38. RS-F018.

Identificador:	RS-F019	Nombre:	Aviso cartas jugadas
Prioridad:	Alta	Necesidad:	Si

Descripción:	El sistema deberá mostrar a todos los usuarios de una partida, la carta jugada por el usuario que le toque en la ronda correspondiente.

Tabla 4-39. RS-F019.

Identificador:	RS-F020	Nombre:	Resultado ronda
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar el resultado de una ronda una vez finalizada a todos los usuarios de una partida, avisando de quién será el siguiente usuario en iniciar la nueva ronda.		

Tabla 4-40. RS-F020.

Identificador:	RS-F021	Nombre:	Aviso fin partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar un aviso cuando finalice la partida para no jugar más rondas.		

Tabla 4-41. RS-F021.

Identificador:	RS-F022	Nombre:	Resultado partida
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema deberá mostrar el resultado obtenido en la partida finalizada a todos los usuarios, mostrando todos los beneficios.		

Tabla 4-42. RS-F022.

4.3.3 Requisitos de recursos

Identificador:	RS-R001	Nombre:	Acceso navegador
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema estará desarrollado de manera que contemple una óptima visualización en los navegadores web compatibles con HTML5: Mozilla Firefox y Google Chrome.		

Tabla 4-43. RS-R001.

Identificador:	RS-R002	Nombre:	Resolución pantalla
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema aportará una resolución de pantalla de 1024x768, de manera que garantice el buen acceso desde los diferentes navegadores web.		

Tabla 4-44. RS-R002.

Identificador:	RS-R003	Nombre:	Resolución otros
Prioridad:	Baja	Necesidad:	No
Descripción:	El sistema aportará una resolución de pantalla inferior a 1024x768, de manera que garantice el buen acceso desde los diferentes navegadores web en otro tipo de dispositivos, como tabletas o móviles.		

Tabla 4-45. RS-R003.

Identificador:	RS-R004	Nombre:	Entorno gráfico
Prioridad:	Baja	Necesidad:	No
Descripción:	El sistema funcionará completamente bajo un entorno gráfico, entendible por el usuario.		

Tabla 4-46. RS-R004.

4.3.4 Requisitos de implantación

Identificador:	RS-I001	Nombre:	Necesidades periféricos
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario podrá interactuar con el sistema por medio de un ratón o una pantalla táctil si la dispone.		

Tabla 4-47. RS-I001.

Identificador:	RS-I002	Nombre:	Instalado navegador
Prioridad:	Alta	Necesidad:	Si
Descripción:	El usuario deberá tener instalado un navegador web compatible: Mozilla Firefox y Google Chrome.		

Tabla 4-48. RS-I002.

Identificador:	RS-I003	Nombre:	Necesidad rendimiento
Prioridad:	Alta	Necesidad:	Si
Descripción:	El sistema podrá ejecutarse en un equipo que pueda soportar cualquiera de los navegadores compatibles.		

Tabla 4-49. RS-I003.

4.4 Alternativas de solución

4.4.1 Preselección de alternativas de solución

A continuación se presentarán las diferentes partes del sistema a desarrollar, atendiendo a los requisitos solicitados y planteados anteriormente, considerándose casos que no pertenecen al sistema pero que influyen sustancialmente.

- **Sistema operativo.**

  En primer lugar, habrá que tener en cuenta qué sistemas operativos se disponen para el desarrollo del sistema, porque es bastante importante saber qué herramientas puede tener cada uno de ellos, incluyendo su coste y su conocimiento para valorar la decisión final.

  Atendiendo a las estructuras actuales, los más destacables son Microsoft Windows en su última versión disponible (Windows Seven) y Linux en cualquiera de las distribuciones existentes. Existe alguna posibilidad más que podría utilizarse, pero no se valora porque o puede ser un poco más anticuado, o su coste se considera excesivo como opciones potenciales.

  Microsoft Windows ofrece una buena actualización y un buen soporte, pero su adquisición es por licencia de pago. Linux es de distribución libre y por lo tanto no requiere un desembolso para obtenerlo. Pero su seguimiento puede ser un tanto caótico, porque existen muchas versiones diferentes al igual que diversas actualizaciones.

- **Lenguaje de programación.**

  Es fundamental saber el lenguaje de programación ya que con ello se sabrán las herramientas y los soportes en los que se sostendrá el producto desarrollado. Además, para el tipo de aplicación que se solicita hay lenguajes muy específicos para este tipo de soluciones y que por lo tanto es esencial saber, sobre todo por ser aplicaciones orientadas a Internet y que generan páginas de contenido dinámico.

  Así mismo, se considera la plataforma de desarrollo que se ofrece en cada alternativa, porque está fuertemente ligado al lenguaje que se desarrolla.

  Los más relevantes y considerados como opciones son:

  - **Java:** Es un lenguaje que goza de una popularidad importante y que en la actualidad es utilizado muy comúnmente en diversos ámbitos. Un

entorno de desarrollo que se ha extendido junto con Java es Eclipse, que es originario para este lenguaje, pero que con el tiempo y sus diversas actualizaciones se utiliza para otros muchos lenguajes. Además es multiplataforma y de libre distribución, por lo tanto podría usarse en cualquiera de los sistemas operativos que se han ofertado.

- **Php:** Es un lenguaje interpretado que goza de una popularidad muy importante tanto en el ámbito laboral como en ámbitos más académicos. Su popularidad se debe principalmente al ser multiplataforma y de libre distribución, haciendo que su instalación sea sencilla y rápida sin demasiados inconvenientes a la hora de hacer unas primeras pruebas. También existen muchos entornos de desarrollo, pero Eclipse es de los más conocidos y permite desarrollar con Php, además de disponer de licencia gratuita.

- **C#:** Es uno de los lenguajes más utilizados porque Microsoft lo ha potenciado en gran medida en sus plataformas de desarrollo. Es un lenguaje muy versátil, es sencillo poder realizar cualquier desarrollo e implantarlo. Visual Studio es el entorno de desarrollo que lo dispone, siendo casi el único que lo permite. Pero tiene inconvenientes, no es multiplataforma, ya que solo está disponible para los sistemas operativos de Microsoft y además su licencia es de pago.

- **Javascript:** La última tendencia existente. Su desarrollo está orientado a los navegadores, porque fue ahí donde se originó y donde goza de la mayor popularidad. En mayor o menor medida, todos los lenguajes de programación que están orientados al desarrollo web, entre ellos los que se han mencionado anteriormente, hacen uso de este lenguaje para que en el navegador web se pueda tener un control y una interfaz más amigable al usuario final. Por lo tanto se hace necesario su conocimiento y su uso. Pero además, están apareciendo nuevos desarrollos para que en los servidores pueda utilizarse para servir aplicaciones web, gracias a que es globalmente conocido por sus orígenes y que su distribución es gratuita y multiplataforma.

- **Servidor de aplicaciones.**

  El sistema que se solicita y que ha quedado reflejado en los requisitos de los apartados anteriores, deja entrever que es necesario el uso de un servidor de aplicaciones web, porque los usuarios tendrán acceso a través del servidor haciendo peticiones de los servicios de que disponga el sistema a desarrollar.

  Las alternativas están fuertemente ligadas al lenguaje de programación que se escoja, pero algunas de ellas permiten integrar algunas de las alternativas. Por lo tanto para cada uno de los lenguajes se ha realizado una selección, siendo probable que alguna de las escogidas, permita ejecutar cualquiera de los otros lenguajes. No obstante se ha realizado la selección atendiendo a términos de la mayor compatibilidad posible, a mayor facilidad de uso y por consiguiente menor tiempo de aprendizaje posible.

  - Internet Information Server es la alternativa para c#, siendo de pago y no es multiplataforma.
  - Nodejs es la alternativa para Javascript, su distribución es gratuita y multiplataforma.
  - Apache Http Server es la alternativa para Php, siendo una distribución libre y multiplataforma.
  - Apache Tomcat es la alternativa para Java, que también es multiplataforma y gratuito.

- **Gestor de bases de datos.**

  El sistema necesitará del uso de un gestor de base de datos, aunque la información que se almacene no deberá ser excesiva por las implicaciones que se obtienen de los requisitos. Quiere decir que se buscarán los gestores menos pesados y más sencillos, para que no requiera tener un conocimiento excesivo a la hora de instalar y/o utilizar. Pero que permitan un control lo suficientemente robusto como para no encontrar problemas o falta de funcionalidad por si en algún momento se necesitase ampliar las necesidades.

  Los gestores seleccionados son MySQL y SQLite. El primero no requiere realizar pago y es de libre distribución, aunque requiere comprar una licencia en caso de que su uso tenga un fin comercial. En cambio no ocurre lo mismo con SQLite, que no se requiere realizar ningún tipo de pago. Ambos se pueden considerar multiplataforma, pero quizá MySQL al ser un sistema más robusto y que permite un control mayor en los datos que almacena, puede perder en cuanto a rendimiento y ejecución.

En cambio, SQLite no es un proceso independiente en el que el programa principal realiza la comunicación pertinente, sino que es en sí una biblioteca que pasa a ser una parte integral del programa donde se utilice. De esta forma se consigue reducir la latencia en el acceso, ya que las llamadas a funciones son más eficientes que la comunicación entre procesos.

Otro dato importante es que SQLite es parte integral de muchos programas, entre ellos algunos navegadores web, como Mozilla Firefox y Google Chrome. Además, parte de sus especificaciones son usadas en el estándar HTML5, como se ha comentado en el apartado correspondiente del estado del arte.

Pero también tiene inconvenientes, como por ejemplo la falta del manejo de la alta concurrencia, procedimientos almacenados y otros muchos aspectos que no es necesario explicar por encontrarse fuera del ámbito del proyecto.

4.4.2 Descripción y valoración de las alternativas

Dispuestos ya los elementos necesarios, a continuación se especifican las alternativas existentes, atendiendo sobre todo a que cada elemento concuerde con todos los requisitos descritos, dando la respuesta necesaria a todos desde una vista general. No obstante cada una de las alternativas dispondrá de unas características que permitirán encontrar la solución de una manera diferente, valorándolo de la mejor manera posible.

Antes de exponer las diferentes alternativas, se ha decidido que el gestor de base de datos sea independiente, es decir, que cualquiera de las posibilidades que antes se han enumerado, sean válidas para todas las opciones. De esta forma se independiza la decisión de la base de datos que se va a usar, porque se considera que cualquier motor puede ser válido y se decidirá cualquiera que ofrezca las mejores garantías a la hora de desarrollar, aprender o conocimiento previo necesario. Quedará entonces fuera del ámbito del sistema otro tipo de decisiones, como mejor uso en alta concurrencia, alta capacidad o mejor manejo en sistemas con un elevado número de peticiones.

- Alternativa Microsoft

Sistema Operativo	Microsoft Windows Seven
Lenguaje de Programación	c#
Servidor de Aplicaciones Web	Microsoft Internet Information Server
Gestor de base de datos	MySQL / SQLite
Entorno de desarrollo	Visual Studio 2010

Tabla 4-50. Alternativa Microsoft.

Esta alternativa tiene un mayor coste económico porque la totalidad de sus componentes deben comprarse. Además dependen completamente del sistema en donde se instalen, porque son solo desarrolladas bajo los sistemas que da soporte Microsoft.

Por el contrario, al ser usado globalmente y ser de fácil acceso, permite un mayor conocimiento en su utilización, además que facilita en gran medida el aprendizaje con interfaces muy intuitivas y funcionales.

También ofrece muchos contenidos con al menos algunas nociones básicas para desarrollar, repercutiendo en la codificación y desarrollo de aplicaciones, aunque pueda impedir cierto grado de «creatividad» de las personas encargadas.

Las aplicaciones desarrolladas con estas herramientas dan seguridad y eficacia a la hora de presentar los productos finales, ofreciendo una buena imagen de garantía con continuas actualizaciones.

- Alternativa Java

Sistema Operativo	Microsoft Windows Seven / Linux
Lenguaje de Programación	Java
Servidor de Aplicaciones Web	Apache Tomcat
Gestor de base de datos	MySQL / SQLite
Entorno de desarrollo	Eclipse

Tabla 4-51. Alternativa Java.

Para esta alternativa y las siguientes, se ofrecen productos con un bajo coste u opciones en el que no existe coste alguno. Además, todas se pueden considerar que no dependen de la plataforma, así se puede asegurar su total funcionalidad en cualquier parte.

En este caso en concreto, gracias a la capacidad de ser multiplataforma, puede ir en detrimento de otras capacidades que en sistemas con un peso específico sean importantes o cruciales para su funcionamiento, como puede ser el consumo de memoria, la rapidez o la velocidad de procesamiento. Aunque en muchos sistemas esto es un dato a tener en cuenta, tampoco son aspectos especialmente relevantes para el sistema que se quiere desarrollar.

Por otro lado, es una alternativa globalmente usada y que en el mundo empresarial goza de relativa consideración. Es por ello que la imagen que ofrece inspira

en muchos aspectos seguridad y limpieza cuando se realiza un desarrollo usando las herramientas elegidas.

Quizá no ofrezca la misma capacidad de aprendizaje y conocimiento de la alternativa anterior, pero al ser herramientas que actualmente solicitan, el coste en formación podrá amortizarse no sólo en este proyecto, sino en futuros desarrollos que se pudiesen ejecutar.

- Alternativa Php

Sistema Operativo	Microsoft Windows Seven / Linux
Lenguaje de Programación	Php
Servidor de Aplicaciones Web	Apache Http Server
Gestor de base de datos	MySQL / SQLite
Entorno de desarrollo	Eclipse

Tabla 4-52. Alternativa Php.

Como ya se ha comentado en la anterior alternativa, esta alternativa es de muy bajo coste y no depende de la plataforma para su desarrollo.

Al poder considerarse multiplataforma, es de los lenguajes que más popularidad ha alcanzado en desarrollo web por su facilidad de uso y su rapidez a la hora de poner en funcionamiento una aplicación. Adicionalmente, por ser interpretado, es de las infraestructuras que más rápido responde, por no compilar o pre compilar el código, aunque para otros aspectos puede presentar problemas o deficiencias.

Requiere un fuerte grado de aprendizaje, aunque está influido por lenguajes de programación estructurada como C, que ayudarían en este sentido. Además, también permite aplicar técnicas de orientación a objetos, con lo que es posible reducir la curva de aprendizaje en gran medida.

- Alternativa Javascript

Sistema Operativo	Microsoft Windows Seven / Linux
Lenguaje de Programación	Javascript
Servidor de Aplicaciones Web	Nodejs
Gestor de base de datos	MySQL / SQLite
Entorno de desarrollo	Sin determinar

Tabla 4-53. Alternativa Javascript.

Como ya se ha comentado en las anteriores alternativas y apartados, es de muy bajo coste, tampoco depende de la plataforma y es requerido en casi todas las alternativas, para ofrecer mayor funcionalidad en el navegador web cuando el usuario hace uso del sistema que se desarrolla.

Por tanto, el aprendizaje puede suponer un problema, pero se hace necesario para todas las alternativas. Y ofrece algunos paradigmas que para la construcción de aplicaciones web pueden ser interesantes, pero a la vez tiene ciertas deficiencias o ambigüedades a la orientación a objetos que pueden equivocar al desarrollador.

Además no dispone de un entorno de desarrollo destacable, que pueda ofrecer con facilidad las necesidades que un desarrollador desea. Aunque sí que pueden usarse algunas de las herramientas comentadas que usan las otras alternativas, para poder solventar algunas de estas deficiencias.

4.4.3 Selección de la Alternativa

Después de estudiar todas las alternativas y valorando las necesidades, se ha decidido que la mejor solución que se aproxima a los requerimientos deseados, es usar Javascript y las herramientas estipuladas como la mejor decisión posible. Esta conclusión se basa en los siguientes puntos:

- Al considerarse multiplataforma, se utilizará el sistema operativo que mejor se adecue, para que así no sea un impedimento si no existen conocimientos suficientes en el sistema seleccionado.
- El conocimiento del lenguaje es muy destacable, sobretodo porque sería necesario conocerlo en cualquiera de las alternativas existentes. Pero si se tiene el conocimiento en esta alternativa se habrá ganado el doble de tiempo para su uso.
- Tampoco habrá excesivos costes para la adquisición de las herramientas, existiendo al menos una opción gratuita para todos los apartados necesarios.
- Aunque quizá no presente un marco de confianza como las otras posibles alternativas, al ser un desarrollo pionero proporcionará una imagen de innovación e investigación que muy probablemente pueda contrarrestar esos inconvenientes.

4.5 Sumario

El capítulo realiza un estudio teniendo en cuenta todas las características y objetivos establecidos en los apartados anteriores. Se establece un alcance y unos requisitos del sistema para saber los límites del proyecto.

Por último, con lo establecido hasta el momento, se generan, definen y describen las posibles alternativas para alcanzar los objetivos, para finalmente seleccionar la que se cree que mejor se adapta a lo requerido.

5 Diseño de la solución

5.1 Contexto del sistema

El producto que se genere tendrá una relación muy estrecha con los diferentes elementos indispensables, que en este caso en concreto es fundamental el navegador web para que funcione la aplicación.

Además es primordial la comunicación existente entre el servidor y el usuario para llevar a cabo las funcionalidades descritas. Y para ello se hace necesario disponer de una interfaz de comunicación homogénea, para que pueda funcionar en el mayor número de dispositivos existentes, en los que obviamente habrá navegadores en los que no esté implementado ningún elemento de HTML5.

Para conseguir ese objetivo será necesario el uso de librerías que permitan la mayor compatibilidad posible, haciendo transparente en la mayoría de los casos la existencia de un estándar diferente al deseado.

Pero todas las necesidades serán integradas en el sistema y no dependerá de otros elementos que ofrezcan algún servicio para cumplir con los funcionamientos deseados.

5.2 Selección y definición de la arquitectura

Es vital la importancia en la decisión de la arquitectura software que se va a emplear para la creación de todo sistema informático. Sobre todo porque en la arquitectura se destacan decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo que se desarrolla posteriormente.

Y es por ese motivo por el que ha sido necesario dedicar un mayor tiempo para escoger entre las distintas arquitecturas existentes. Sobre todo para escoger la más apropiada para resolver el problema, problema al que se debe encontrar solución factible y conseguir que esa decisión sea la más sencilla posible.

El estilo arquitectónico seleccionado para el diseño del sistema ha sido la arquitectura de 4 capas, aunque con alguna variación por las restricciones existentes en las funcionalidades, que se detallan a continuación del diagrama básico de la arquitectura:

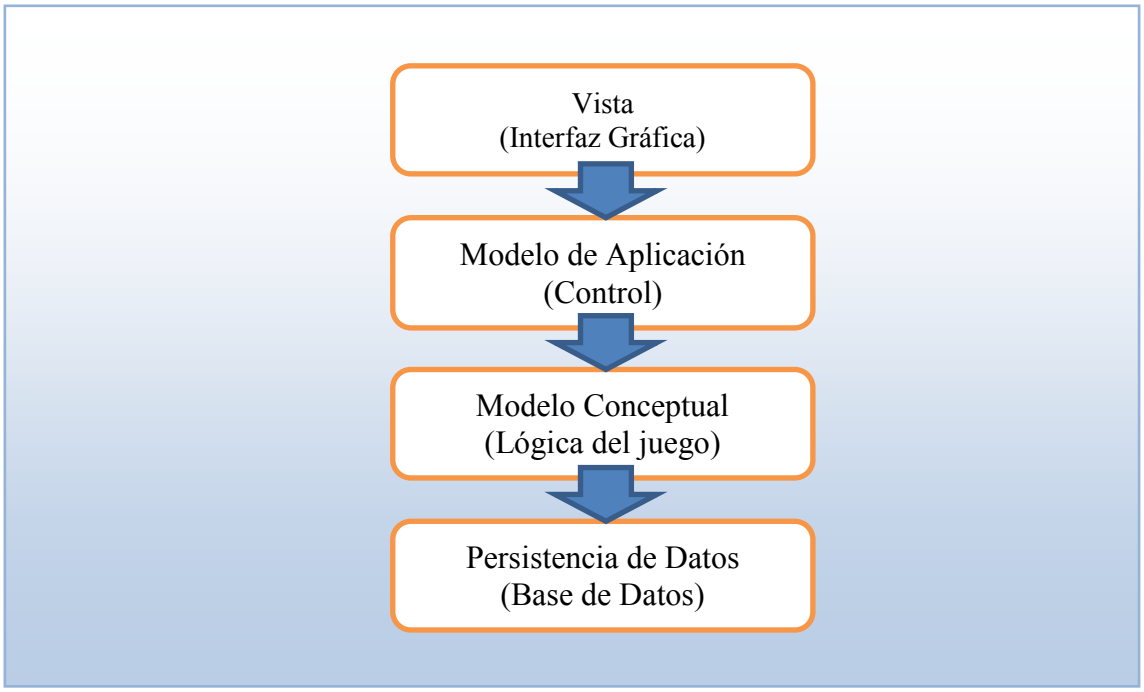

Ilustración 5-1. Arquitectura.

La definición exacta de la arquitectura de 4 capas exige que una capa sólo utilice los servicios de la que tiene por debajo y aunque ofrece servicios a la que tiene por encima, nunca los solicita. En nuestro caso concreto, se hace necesario tener una comunicación en ambas direcciones entre la capa de la vista, y la lógica del juego o la capa de control. Eso quiere decir que en la vista se hace necesario ofrecer servicios tanto al usuario por medio de la interfaz gráfica, como a las capas inferiores para saber los resultados obtenidos en los diferentes procesos de ejecución.

La arquitectura Cliente-Servidor solventa en gran medida estos problemas, pero existe el problema de la dependencia de los distintos módulos que conforman el sistema. Gracias al acoplamiento de ambas arquitecturas, se conseguiría solventar los inconvenientes existentes en ambas arquitecturas.

Una arquitectura MVC (Modelo-Vista-Controlador) podría ser útil, pero gracias a las 4 capas se consigue que se independicen muchas de las acciones de diferentes usuarios. Si se hace que la lógica del juego quede encapsulada en un componente a parte, se consigue que sus modificaciones no afecten a otros elementos.

Además se logra algo que no está estipulado en los requerimientos, pero que puede llegar a ser interesante llevar a cabo en un futuro: la inclusión de nuevos juegos con sus correspondientes lógicas de juegos. De esta forma sería más sencillo llevarlo a cabo, sin que se obligue a realizar grandes y laboriosos procesos de modificación.

5.3 Descripción de la descomposición

Después de seleccionar y describir la arquitectura que se desea, se hace necesario visualizar de la mejor manera posible los elementos que debe tener la arquitectura, además de describir y descomponer cada uno de los elementos existentes.

Para una mejor apreciación, se hace uso del estándar UML como el lenguaje de descripción de la arquitectura y del análisis detallado de cada componente. Porque con UML y gracias a los principios del paradigma de la programación orientada a objetos y a los patrones de diseño, se permite una mayor facilidad a la hora de visualizar y entender lo que se desea, además de permitir una fácil implementación y un desarrollo mantenible.

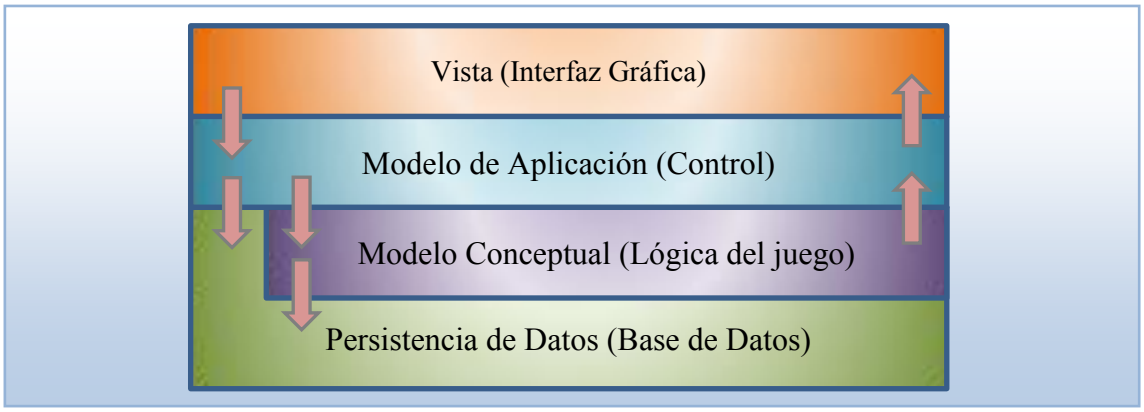

Ilustración 5-2. Visualización de la Arquitectura según los servicios que ofrecen las capas.

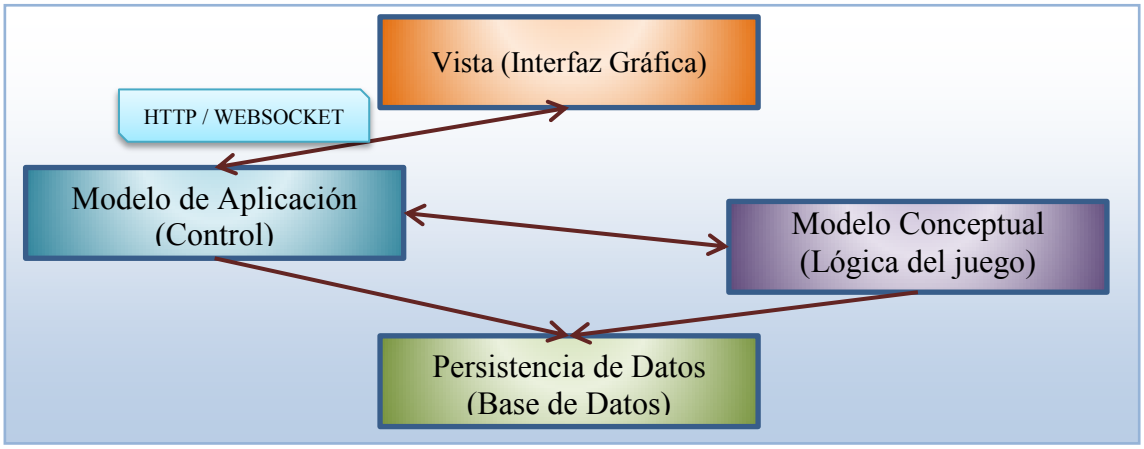

Ilustración 5-3. Visualización de la Arquitectura según el nivel de comunicación.

En la primera figura se representa la arquitectura en la que las capas inferiores ofrecen servicios a las capas superiores. La diferencia con la arquitectura de 4 capas clásica es que una de las capas inferiores tiene la obligación de hacer uso de los servicios de una capa superior, porque necesita enviar información de algún modo a los usuarios cuando hay algún tipo de evento que se ha recibido a lo largo del tiempo.

En este caso en concreto, la información que se necesita enviar es cuando un usuario está participando en una partida y ha ocurrido algo, como por ejemplo que otro usuario ha utilizado una carta. Por eso la capa de la vista que se estará ejecutando en el navegador web del usuario necesitará ofrecer servicios tanto al usuario como a la capa de la lógica del juego, esperando de cualquiera de los dos algún tipo de evento.

En la segunda figura se intenta representar el nivel de comunicación existente entre las capas. La única capa que se debería ejecutar en un entorno diferente al resto es la vista. Y es por eso que necesita realizar algún tipo de comunicación entre los diferentes elementos.

No obstante la capa de persistencia de datos deberá realizar algún tipo de comunicación para obtener los datos que se necesiten, aunque depende de la base de datos que se utilice y donde se ubique, porque existirá comunicación al menos por ser un proceso diferente (si la base de datos es MySQL). En la siguiente figura intenta representar de una forma un poco más clara ese tipo de comunicación entre los diferentes elementos.

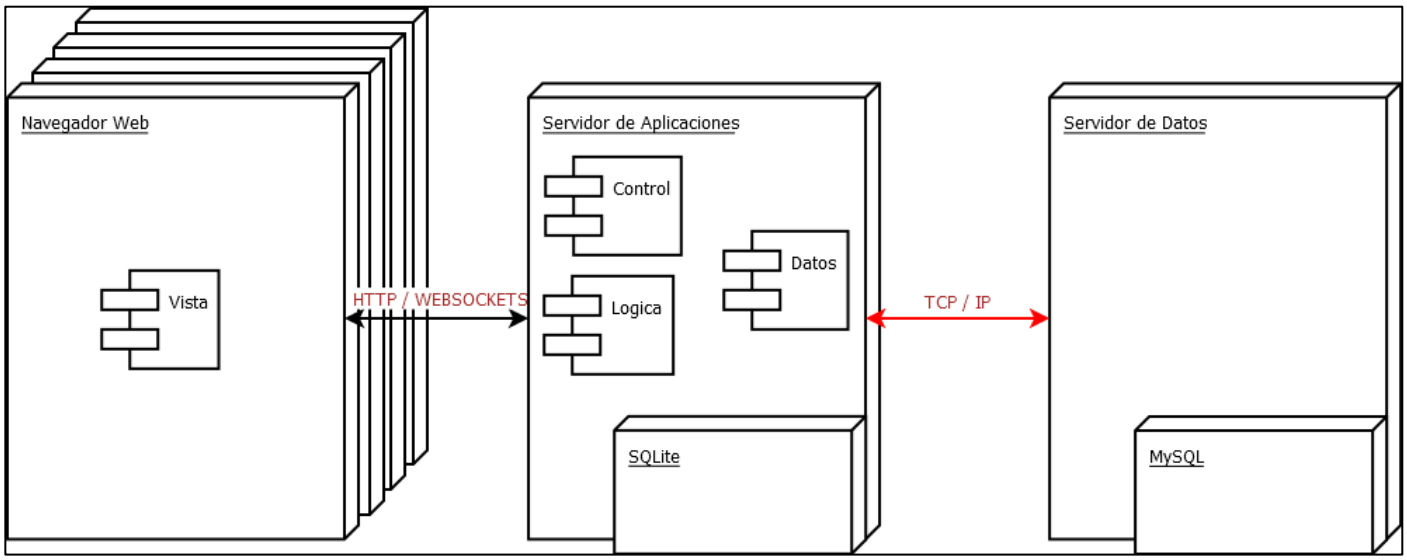

Ilustración 5-4. Visualización de la Arquitectura según comunicación entre servicios.

Destacar de la figura que aparecen ambos motores de base de datos para saber las circunstancias de uso en cada uno de ellos.

5.4 Descripción y especificación detallada de los componentes

Antes de empezar es necesario visualizar el diagrama de componentes con las dependencias existentes entre los diferentes elementos.

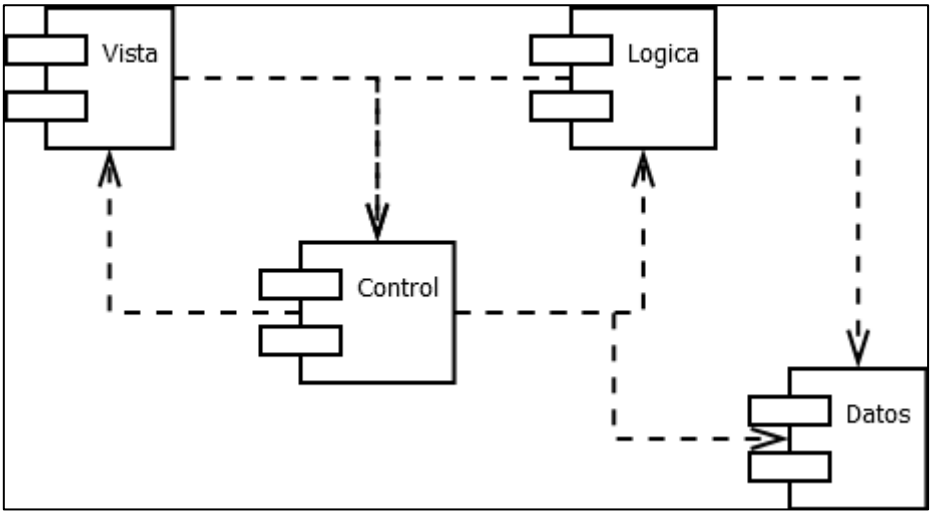

Ilustración 5-5. Componentes.

Como se observa en la figura anterior existirán 4 componentes, que a continuación se describirán y detallarán todos los elementos que contengan. Al ser una estructura bastante elaborada y para no hacer que el documento actual sea muy exhaustivo, algunas de las figuras y descripciones estarán resumidas. En cada caso en concreto se hará referencia a este aspecto.

5.4.1 Componente Persistencia de Datos (Datos)

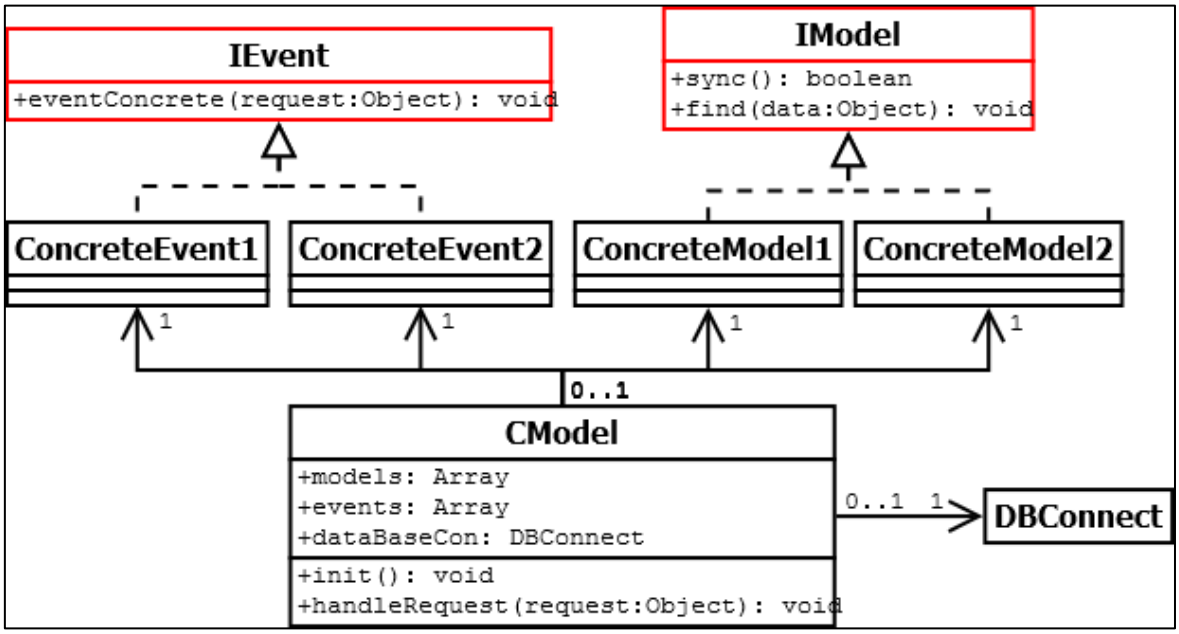

Ilustración 5-6. Componente Persistencia de Datos.

Este componente se encargará de comunicarse con los datos almacenados en la base de datos y ofrecerlos a la capa que lo ha solicitado. Para realizar este proceso se hará transparente todo el proceso de selección de base de datos, incluido su manejo o toda la configuración posible por si es necesario realizar una conexión a un elemento externo.

La clase de control se encargará de inicializar tanto los eventos como los modelos existentes. Una vez que recibe una petición delegará en el evento concreto lo que se debe realizar. El evento en concreto hará las operaciones necesarias para obtener la información que se ha solicitado, recibiendo dicha información en los modelos concretos para luego tratarlos y sólo dar lo que la capa superior necesite.

Para entender mejor este funcionamiento a continuación se incluye un diagrama de secuencia con la ejecución típica para un evento concreto. Es destacable del diagrama que si ocurre un error o no se obtiene ningún tipo de dato se enviará un error a quien realizó la petición. Una ejecución normal rellenará con lo obtenido de la base de datos los datos solicitados, como se puede apreciar en el diagrama.

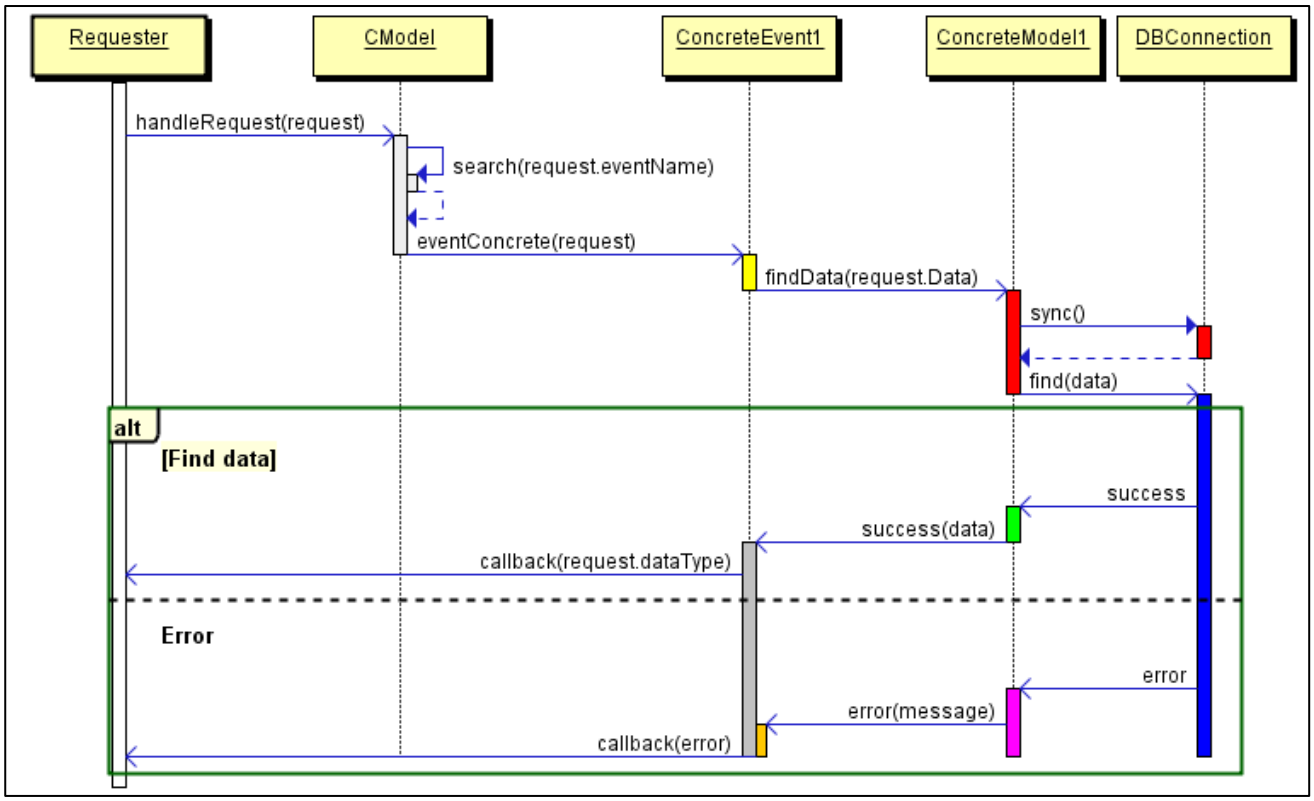

Ilustración 5-7. Diagrama de secuencia de Componente Persistencia de Datos.

Ambas ejecuciones aunque en el diagrama parezcan realizarse a la vez, no ocurre así: o se obtiene algún dato o se producirá un error que se delegará a quien realizó la petición. Por ello, la ejecución normal se muestra con los mensajes de '*success*', correspondiendo a una ejecución satisfactoria y los mensajes de '*error*' corresponden a una ejecución errónea, o que se ha producido algún tipo de error.

También Puede ocurrir que en alguno de los pasos se obtengan errores de algún tipo, por tanto puede pasar que aunque se obtenga un dato correcto de una llamada, ese objeto puede cursar una llamada errónea. Por eso se muestran ambos mensajes en cada paso para poder entenderlo mejor.

5.4.2 Componente Modelo Conceptual (Lógica del Juego)

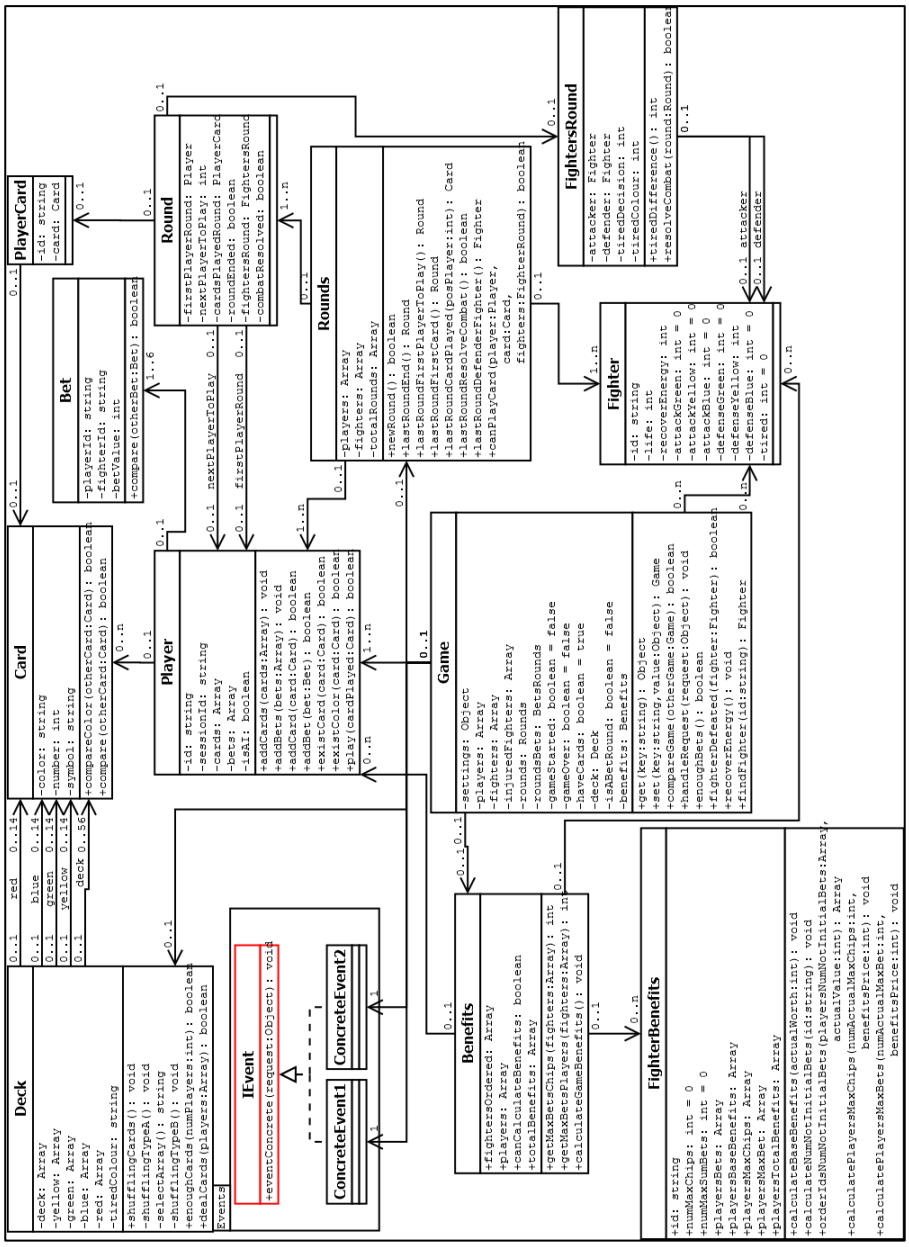

Ilustración 5-8. Componente Modelo Conceptual.

Para este componente es necesario hacer uso de diferentes clases para dividir de la mejor manera posible todo el trabajo que consiste en controlar una partida. El controlador encargado de manejar todos los lances, creará una partida asignándole los eventos necesarios para manejarla.

Cuando algo destacable ocurre, la clase que recibe los eventos (la Clase Game en este caso) se encargará de avisar a los elementos que están esperando ese evento. Esto se implementa con el patrón de diseño *Observer*, haciendo con facilidad el manejo y el aviso a todos los elementos involucrados. Este tipo de acciones son muy comunes en javascript, que se encargan de esperar por acciones del usuario, por ejemplo hacer click en un enlace o botón.

A continuación se incluye un diagrama de secuencia para explicar la ejecución más típica: un usuario decide usar una carta. Para no hacer que el diagrama se pudiese convertir en un caos, se ha decidido reducir significativamente algunos de los mensajes menos relevantes. Por ejemplo, en el caso de que un usuario juegue una carta cuando no le corresponde, donde se emitiría un mensaje de error a quien realizó la petición. O por ejemplo el usuario juega una carta que no debe, que daría como resultado el mismo tipo de acción del anterior ejemplo.

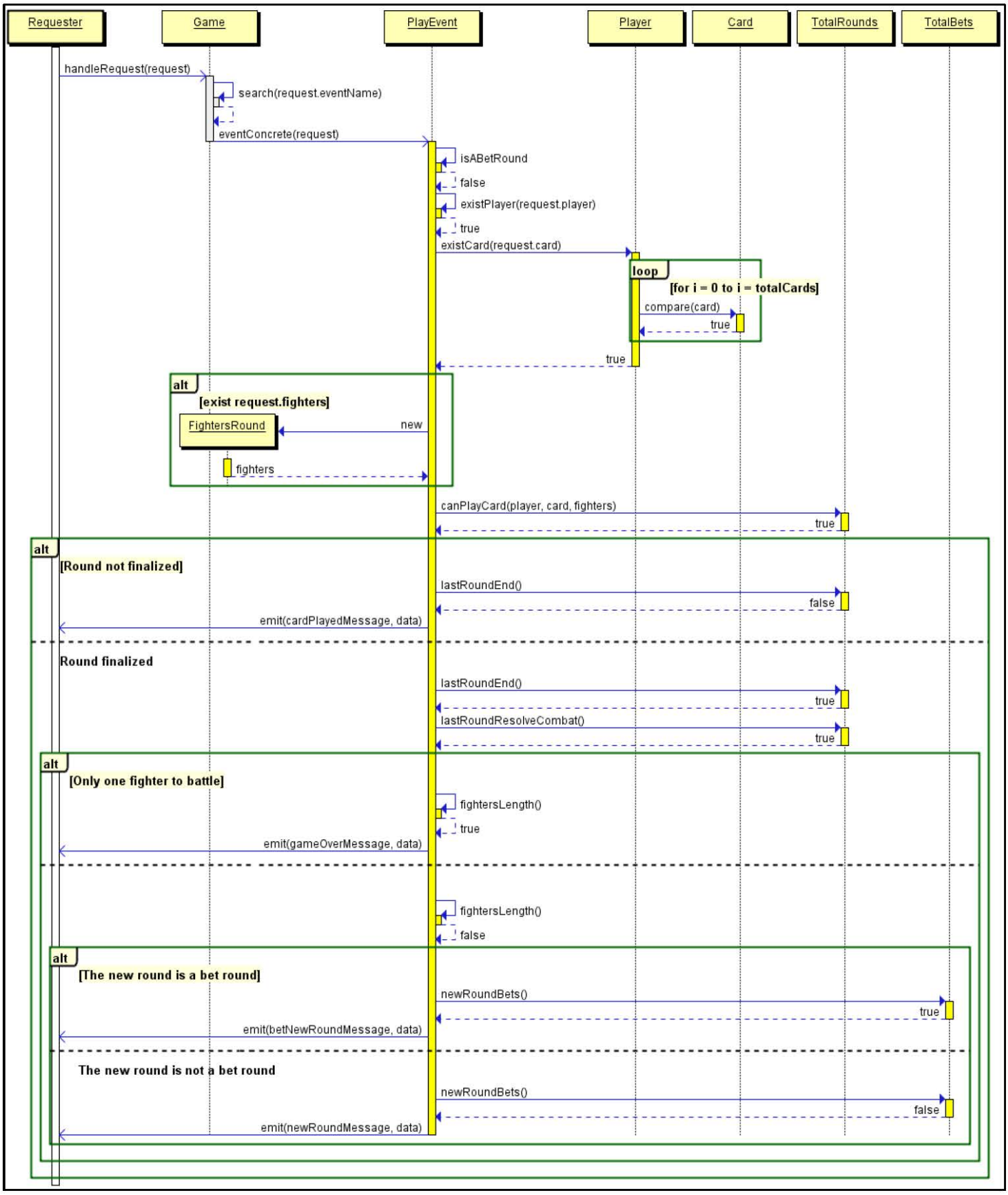

Ilustración 5-9. Diagrama de secuencia de Componente Modelo Conceptual.

5.4.3 Componente Modelo de Aplicación (Control)

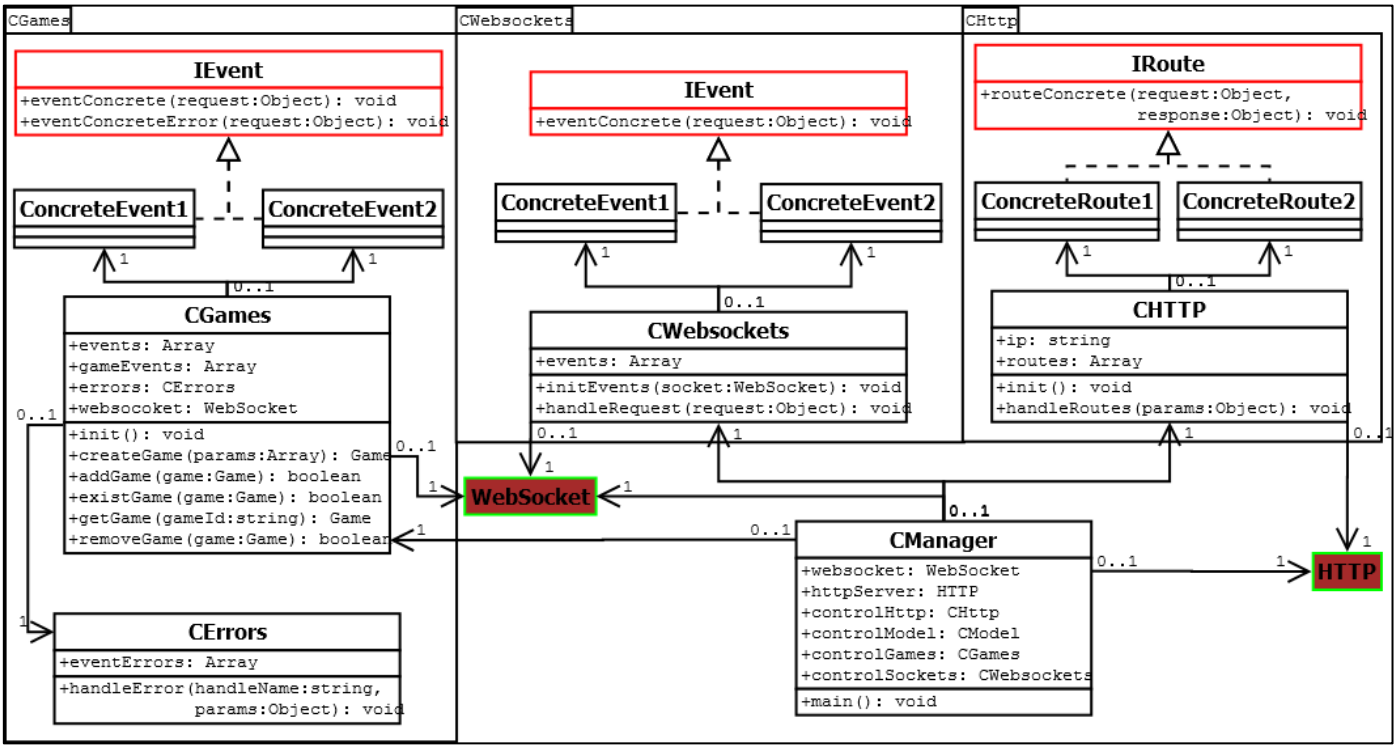

Ilustración 5-10. Componente Modelo de Aplicación.

El componente de control se encargará de recibir todas las peticiones de la parte de la vista y dependiendo de la solicitud se encargará un manejador u otro. También recibirá eventos de parte de la partida, es decir del componente encargado de la lógica del juego.

Además, se observa en el diagrama de clases, que para el manejo de todos esos elementos se hace necesario hacer uso de las librerías habilitadas que reciban la información, en este caso existirán dos librerías (Websockets y http) que traten toda esa información. La clase respectiva se encargará de inicializarlas, pero será el Manejador general quien controle su uso.

Como se ocupará de toda la comunicación entre los diferentes componentes, deberían aparecer en el diagrama los demás componentes que hace uso. Como en definitiva deberían aparecer todos, se ha reducido simplemente a hacer mención a los elementos que debería hacer uso y dejando sólo en el diagrama aquellos elementos más relevantes.

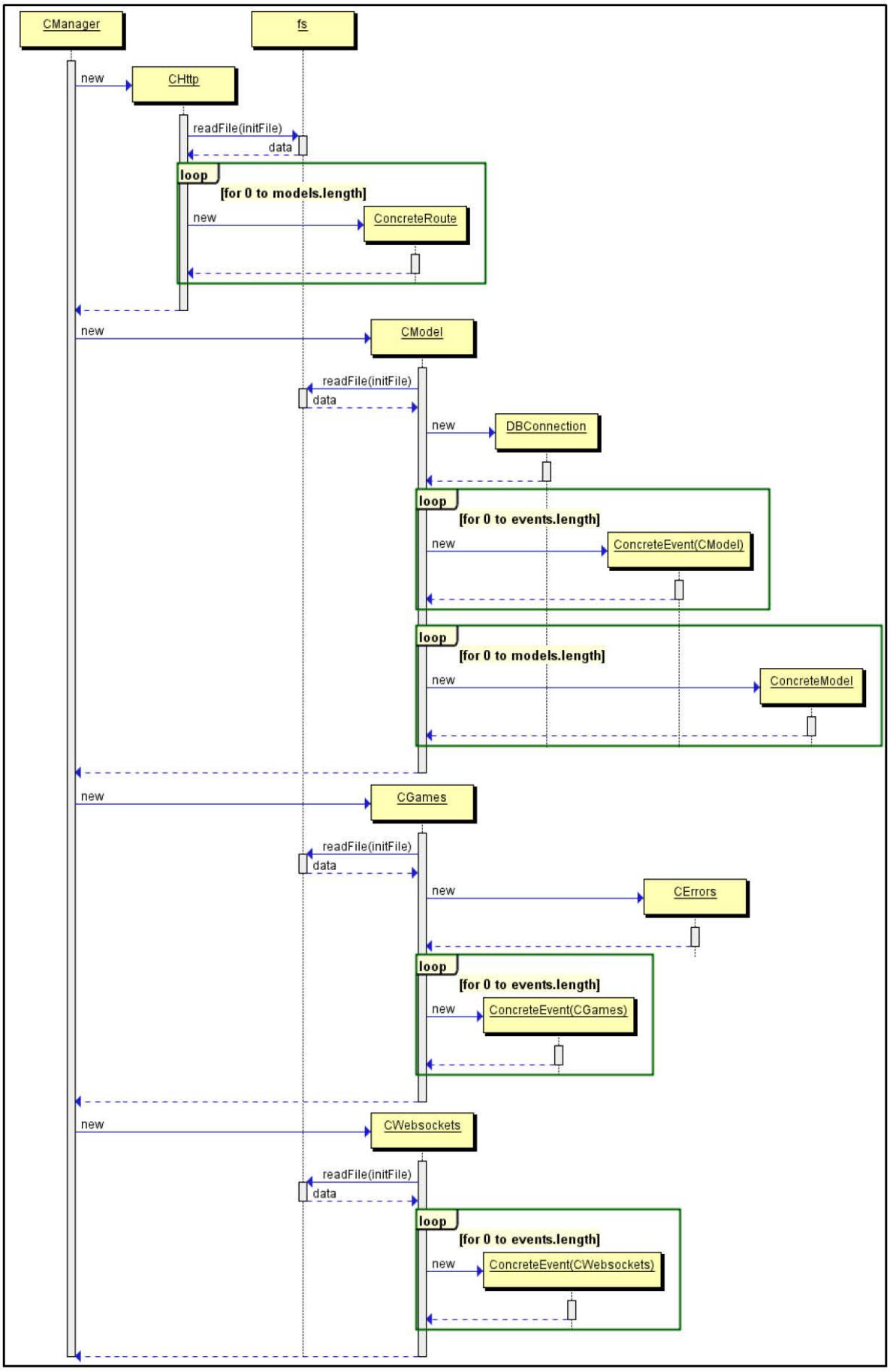

Ilustración 5-11. Diagrama de secuencia de Componente Modelo de Aplicación (Inicializar).

Como se observa en el diagrama anterior, el manejador general se encarga de inicializar todos los elementos que conforman el proyecto. Y como ya se ha comentado, no sólo se encarga de los miembros que conforman el componente de control, sino que además necesita iniciar los demás componentes que precisan una configuración inicial. Es el caso del componente de datos y la parte que se encarga de la lógica del juego.

Por otro lado, la mayoría de las peticiones están relacionadas con el manejo de la partida, pero habrá peticiones que lleguen de modo clásico, por HTTP, que sobretodo se utilizarán al iniciar la comunicación entre el usuario y el servidor, cuando solicita la página por primera vez y se registra o se autentifica en la aplicación.

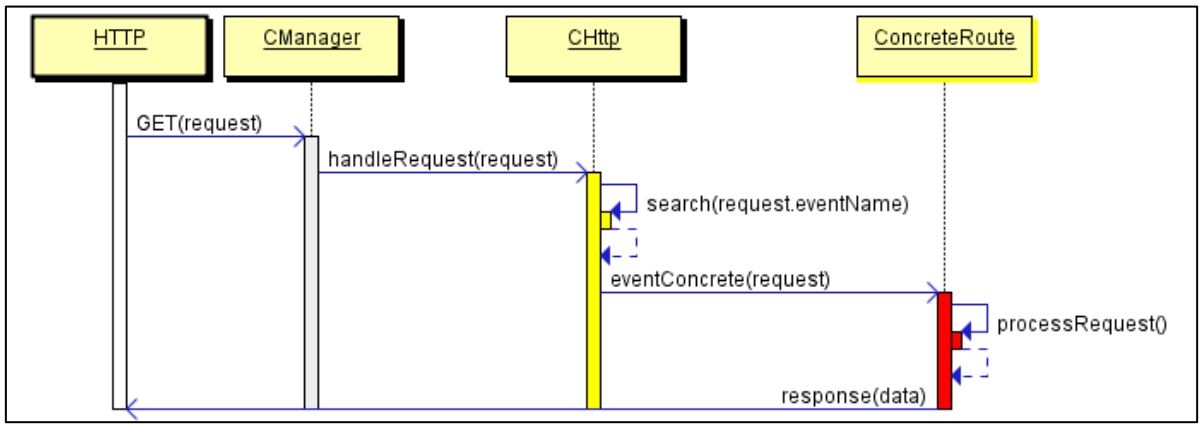

Ilustración 5-12. Diagrama de secuencia de Componente Modelo de Aplicación (Ejecución).

Se puede ver en este diagrama una secuencia de ejecución para lo que se ha explicado. Muchos de los procesos tendrán cierto parecido a la hora de recoger las peticiones y cursarlas, como ya ha ocurrido en los dos primeros componentes.

A continuación se muestra un diagrama de secuencia en el que se distingue el proceso de creación y eliminación de una partida. Una observación importante es que lo normal sería terminar la partida solamente cuando realmente haya terminado, es decir, llegase una acción avisando de que cierta partida ha terminado. En este caso, sucede en el diagrama con la acción '*gameEnded*', que ocurre cuando la partida finalizó.

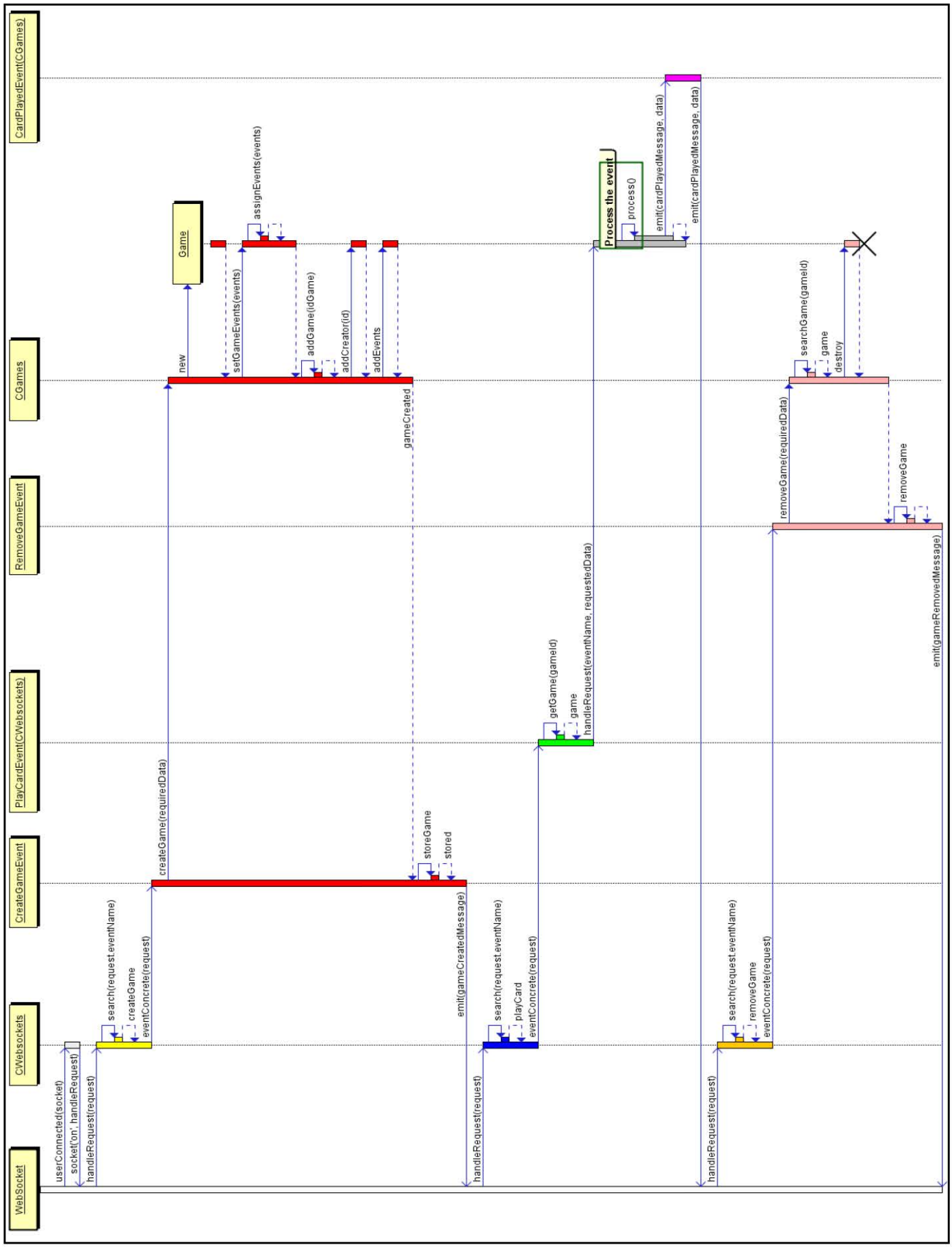

Ilustración 5-13. Diagrama de secuencia de Componente Modelo de Aplicación (Crear Partida).

En el momento en que se conecta un usuario, se inicializan todos los eventos que debe tener para recibir las peticiones por Websockets. Después, en el preciso momento de crear una partida nueva, el controlador de partidas creará un nuevo elemento y le asignará los eventos.

Hay que destacar la diferencia existente entre dos tipos de eventos en este caso. Existen los eventos propios de la partida, es decir, los que se realizan con acciones externas a la partida pero que son necesarios que ocurran para que la partida pueda avanzar. Estos eventos ya se explicaron en el componente de la lógica de juego. Aunque hay que destacar que son necesarios asignarse externamente para que pueda configurarse cada acción, así se consigue que si alguno de los eventos son opcionales, se asignen sólo los que correspondan.

Luego existen los eventos que se generan internamente en la partida. Estos últimos eventos están relacionados con el patrón de diseño '*Observer*'. Resumiendo muy brevemente el fin de este patrón es que cuando ocurre algo destacable, se avise a quien se ha mostrado interesado. Por eso el componente control se encarga de asignar estos eventos para que cuando ocurra algo en la lógica, salte el evento concreto y se realicen las acciones deseadas.

Por ejemplo y para que quede un poco más claro y como se puede observar en el diagrama anterior, cuando un usuario decide jugar una carta, la lógica hará las pertinentes acciones y posteriormente, avisará a todos los interesados en saber que se ha jugado esa carta. En ese momento saltará el evento concreto que esté suscrito, haciendo sus correspondientes acciones, como por ejemplo avisar a los demás jugadores de que un usuario ha jugado una carta.

No obstante se ha resumido parte de la acción porque ya aparece representada en el componente de lógica de juego y si se incluyese haría aún más difícil de entender el diagrama. Que lo que intenta explicar es claramente que los controladores al final delegan en los eventos toda la responsabilidad de tener que hacer las llamadas necesarias al elemento correspondiente.

5.4.4 Componente Interfaz Gráfica (Vista)

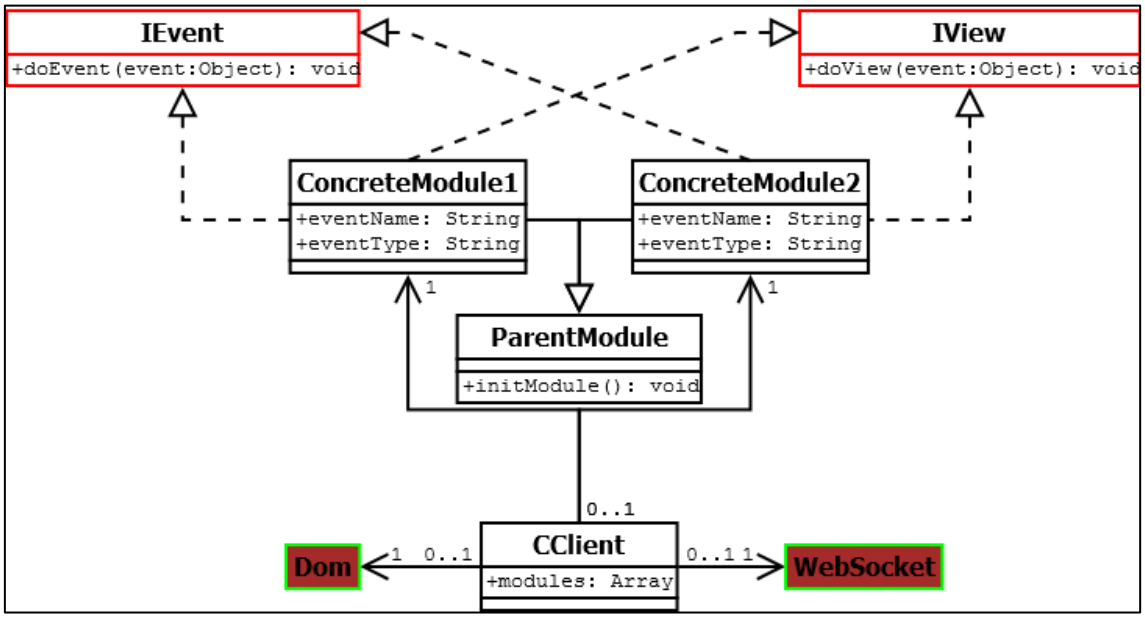

Ilustración 5-14. Componente Interfaz Gráfica.

Como se puede observar en el diagrama de clases, habrá una clase que controlará las peticiones que lleguen tanto del usuario como de las peticiones de la capa inferior al ocurrir eventos externos.

Para realizar este proceso, necesita tener control tanto de los elementos que se visualizan en el navegador para recuperar los eventos que ocurren en él por parte del usuario, como las librerías que controlan el manejo de los Websockets o la comunicación existente entre el navegador y el servidor.

Un aspecto importante es que la vista se tiene que actualizar de modo diferente al tradicional. El modo tradicional simplemente es hacer peticiones al servidor de aplicaciones y este devuelve, dependiendo de la petición, una página html con los datos solicitados, haciendo que la página cargue las veces que sea necesaria cuando el usuario va realizando acciones diferentes.

Como ese modelo requería para entornos muy complejos de mucha carga de datos, se han ido buscando modelos menos pesados, haciendo que la información que se envíe sea más reducida. Por lo tanto las peticiones tradicionales no deben realizarse en este caso, porque gracias al uso de una conexión constante se puede enviar información en ambos sentidos, sin saturar la red.

Como comprende un cambio sustancial al establecido, la vista requiere de una lógica mucho más elaborada que la comunicación web tradicional, haciendo que sea necesario el uso de los mecanismos típicos usados en servidores o aplicaciones comunes de escritorio.

A continuación se puede ver un ejemplo de una acción en un diagrama de secuencia. La petición puede venir de cualquiera de los dos lugares, tanto de acciones del usuario a través de lo que visualiza, como de información que llega del servidor a través de los Websockes.

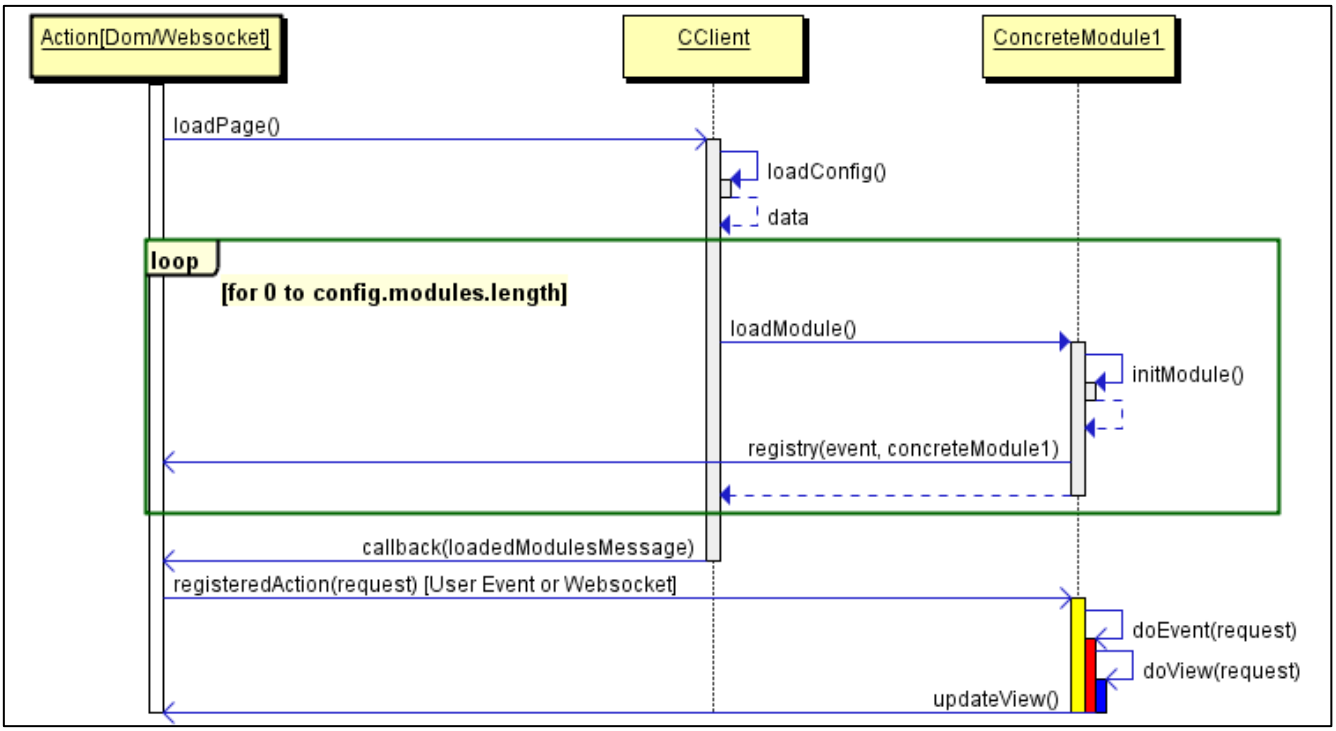

Ilustración 5-15. Diagrama de secuencia de Componente Interfaz Gráfica.

Para que quede un poco más claro el diagrama, las acciones pueden llegar desde el propio navegador, que se ha denominado "Dom", haciendo referencia al documento que carga el navegador cuando realiza una petición por internet. Y del Websocket cuando llegan acciones desde el servidor.

Al hacer la primera visita, el navegador realizará la carga de los scripts y el controlador efectuará una primera configuración, cargando todos los módulos disponibles, como se observa en el diagrama.

Cada módulo llevará a cabo un registro en donde le corresponda, si es en el "Dom" para capturar acciones del usuario, o si es en el Websocket, para capturar las acciones del servidor. Y cuando llegue una acción, la acción registrada saltará accionando la funcionalidad del módulo en concreto.

Éste a su vez realizará las acciones correspondientes, preparando valores y variables necesarios (que se encapsulan en la acción "doEvent"). Cuando ya esté todo listo, si hay que cambiar algo de la vista, se realizarán las acciones pertinentes en "doView" que cambiará los elementos necesarios. De esta forma queda encapsulado lo que es realizar procesamiento de

información, con la parte de la presentación de dicha información y que el usuario verá en el navegador web.

5.5 Sumario

Se ha visto a lo largo del capítulo las exigencias postuladas para preparar el diseño y luego ha sido necesario establecer y definir una arquitectura que se adecúe a lo necesitado. Con la arquitectura seleccionada, se ha establecido una descomposición de cada uno de los elementos existentes, describiendo detalladamente cada uno de los componentes:

- Se ha presentado un componente encargado de controlar la interacción con los datos que se almacenan de los usuarios y los luchadores.
- Existe también un componente encargado de la lógica del juego, para controlarla y que interactuará con los demás componentes involucrados.
- El componente de control se encarga de establecer las competencias de los demás componentes y de recibir y tratar cada llamada para que se ejecute el correspondiente elemento.
- Y por último el componente visual que el usuario ejecutará en su navegador.

6 Implementación

6.1 Estándares de proyecto, convenciones y procedimientos

A la hora de la implementación y codificación de los requerimientos obtenidos y desarrollados en todos los apartados anteriores, se hace necesario en primera instancia poner en común una normativa y unos procedimientos para mantener la unicidad en todos y cada uno de los elementos.

La mayoría de los casos son convenciones y procedimientos generales que se pueden adaptar a cualquier desarrollo. Pero atendiendo a este caso en concreto, se hacen necesarios métodos muy específicos para evitar problemas y para tener un código que pueda mantenerse con facilidad.

- **Nombre de Ficheros:** Los ficheros se identificarán con un nombre descriptivo y una extensión correspondiente al tipo de fichero. El nombre descriptivo comenzará siempre por minúscula y las siguientes letras deberán ser minúsculas. Si contiene más de una palabra, la primera letra de la palabra podrá ser en mayúscula y no habrá espacios entre las palabras. Tampoco estará permitido usar caracteres anglosajones en el nombrado de ficheros.

- **Idioma:** El idioma por defecto será el inglés, tanto a la hora de asignar nombres como en la descripción que documente los procesos que se llevan a cabo.

- **Identificadores:** Se empleará cualquier letra del alfabeto, incluidos los números del 0 al 9, aunque deberá empezar siempre por una letra y a continuación cualquier combinación deseada.

- **Constantes:** Las constantes tendrán todas sus letras en mayúsculas.

- **Variables:** Las variables tendrán un nombre descriptivo sobre el uso que se le va a dar, que seguirá la misma regla de nombrado de ficheros.

- **Funciones y procedimientos:** Las funciones y procedimientos seguirán el mismo nombrado que las variables y los ficheros: la primera letra será en minúscula y si existieran varias palabras, a excepción de la primera, deberá tener la primera letra en mayúscula sin ningún espacio entre las palabras.

- **Minimizar variables globales:** Esto es un caso concreto del lenguaje de programación. Dependiendo del entorno en donde se ejecute código Javascript (sobre todo en el caso de los navegadores), toda variable declarada, que no corresponda a una función, podrá ser usada en cualquier parte del código.

  El problema puede surgir cuando se hace uso de terceras librerías que puedan utilizar nombres de variables que las que se hacen uso globalmente, dando al final algún tipo de error, tanto en el código de la tercera librería como el código propio del desarrollador.

  Por tanto, para evitar este tipo de problemas, es conveniente no tener variables globales, o al menos reducirlo al mínimo posible: una única variable global que contenga todo lo necesario.

- **Declaración de variables una única vez:** Esto también es algo concreto del lenguaje de programación, aunque puede ser extensible a otros lenguajes que tengan cierto parecido. Las variables se declararán al inicio de las funciones, ayudando de esta forma la lectura del código, previniendo posibles errores y reduciendo el uso de variables globales.

  Además, de una manera muy simple de explicar algo aún más complejo: Hay dos fases a la hora de ejecutar el código, donde las variables, la declaración de funciones y los parámetros formales son creados en la primera fase, que se la puede denominar como parseo y preparando el contexto. Y en una segunda fase, es el estado de ejecución de código, donde las variables no declaradas son creadas.

  Es por eso que las variables aunque estén declaradas en diversos puntos del código, en la ejecución su asignación es de las primeras cosas que se realizan. De esta forma su declaración inicial hace que se parezca en cierta medida a lo que realiza el propio compilador donde se ejecuta el código.

- **Evitando la conversión automática de tipos en comparaciones:** Otro elemento concreto del lenguaje de programación. En las comparaciones es preferible hacer uso de la triple igualdad '= = =', para que además de comparar los valores, obligue a que los elementos sean del mismo tipo, algo que no ocurre con el comparador habitual '= ='.

6.2 Aspectos predominantes del lenguaje de programación

De las muchas características de Javascript, hay algunas de especial relevancia porque hacen de este lenguaje algo singular para entender y comprender. Sobre todo para que muchos de los elementos que se han descrito y diseñado puedan realizarse de la mejor manera posible.

- Javascript es un lenguaje orientado a objetos. Muchos desarrolladores se llevan cierta sorpresa al usarlo por primera vez ya que en un principio no parece serlo. Toda variable, excepto 5 tipos primitivos que también tienen su correspondiente representación (number es primitivo pero también tiene su representación en objeto: Number).

  Para no extender más la explicación, los objetos en Javascript son una colección o lista de pares clave-valor, denominados propiedades del objeto.

- Algo muy importante y que puede parecer un poco extraño considerando el apartado anterior, es que en Javascript no existen las clases. Para un desarrollador convencional esto es un tanto extraño y difícil de entender, sobre todo si se viene de desarrollos de los lenguajes más usados hasta el momento, algo que te obliga a desaprender para entender con mayor soltura.

  Además, un objeto puede crearse y editarse en tiempo de ejecución, algo que con un desarrollo de clases no existe. Y en Javascript es preferible la composición de objetos antes que la herencia de clases. Esto significa que un objeto es mejor crearlo con piezas de código que interesen de lo que se ha desarrollado, antes que hacer una aproximación más robusta de herencia padre-hijo.

- Javascript es un lenguaje interpretado sin un tiempo de compilación, haciendo posible desplegar código o programas con errores. Por ello muchas veces hace que el desarrollo sea un tanto difícil por no disponer de un control mayor en su ejecución.

  En sus inicios, para comprobar que el código que se desarrollaba era correcto no había otra forma que ver los resultados sobre el navegador. Actualmente esta faceta ha sido mejorada y con los nuevos estándares se hace más liviano poder ejecutar código en los navegadores.

- Las funciones en Javascript adquieren una especial relevancia. Pueden ser tanto propiedades de objetos, porque en la asignación de clave-valor, el valor puede ser una función (en ese caso la asignación se la puede conocer como método del objeto), como

también pueden ser objetos propios, que en este caso se le conoce como 'funciones constructor'.

Como se pueden considerar como objetos, las funciones pueden crearse dinámicamente en tiempo de ejecución, pueden asignarse a variables, tienen sus propias propiedades y métodos e incluso se pueden pasar como referencia en los argumentos. Esto último es lo que permite una de las características más valoradas de Javascript y que a continuación se comenta.

- Como las funciones son objetos, estas pueden pasarse por argumento a otras funciones. Las funciones que reciben por argumento una referencia a otra función, pueden ejecutarlas, llamando a esto '*callback*'.

  Es algo muy sencillo de utilizar, pero en Javascript es muy poderoso a la hora de desarrollar librerías. Permite que el núcleo de las aplicaciones sea más liviano y sencillo de realizar, proveyendo de lo necesario a la función de retorno y esta se encargará de lo necesario para hacer su uso.

- Con la idea del anterior punto, Javascript es un lenguaje orientado al evento, haciendo que sea una de las características más importantes y usadas. La programación en los navegadores está orientado a los eventos asíncronos, haciendo que su uso sea muy extenso a la hora de realizar cualquier acción del usuario. Asíncrono porque lo que permiten las funciones que se pasan por parámetro a un evento en concreto, es que se ejecute cuando ocurre tal evento, y sólo cuando éste ocurre.

  Es una característica que por extensión se ha convertido en algo intrínseco del lenguaje, haciendo que sea muy valorado por ello. A la hora de usarse en el lado del servidor, es su característica más importante.

  También se le asocia a la idea de Hollywood: "no nos llames, nosotros te llamaremos a ti". Porque no es computable que en una audición con cientos o miles de candidatos para un rol en una película, el personal del casting tenga que esperar o responder a todas las llamadas de los candidatos. Sino que si el candidato da su número de teléfono, sólo se le llamará si es seleccionado.

  Esto mismo ocurre en los eventos asíncronos de Javascript. Si se proporciona una función para un evento concreto, entonces cuando ocurra el evento se ejecutará el código. Si este evento no ocurre nunca, evidentemente no se ejecutará.

6.3 Código relevante a destacar

Con las características comentadas, es considerable y significativo resaltar algunos aspectos del código, haciendo hincapié en muchas de las peculiaridades explicadas en el apartado referente al diseño del proyecto.

- **Orientación al evento**

Quizá la característica más importante del lenguaje Javascript es que su arquitectura está orientada al evento, como ya se ha aclarado. Y esa peculiaridad es mucho más significativa en el lado del servidor, porque es la principal propiedad que '*Nodejs*' ha remarcado desde sus inicios.

```
var events = require('events');

exports.Game = Game;

function Game() {
	//It's needed to launch events.
	events.EventEmitter.call(this);

}

//inherit EventEmitter to run events.
sys.inherits(Game, events.EventEmitter);

Game.prototype.concreteFunction = function() {

	if (!this.gameStarted && !this.gameOver) {
		this.emit('message');
	} else {
		this.emit('messageError', 'message');
	}
}
```

Código 6-1. Eventos.

En el ejemplo que se ilustra anteriormente, se puede observar cómo funciona en '*Nodejs*' el uso de los eventos. En este caso con la primera sentencia se obtienen las librerías necesarias para poder emitir los eventos.

La segunda sentencia sirve para exportar los objetos que se desean a otros programas y así poder utilizarse por quien quiera. Esto es una manera de encapsular la información que se quiera, dejando visible las partes que interesen.

Para utilizar los eventos, o hacer que nuestro objeto pueda emitir eventos y que alguien los use, se hace necesario heredar e inicializar el evento, como se ven las siguientes líneas de código. Por último, los mensajes que se quieran lanzar se hacen usando la sentencia '*emit*'.

Ahora alguien debe hacer uso de esta librería y suscribirse al evento correspondiente, como se muestra a continuación:

```
var Game = require('game').Game,
    game = new Game();

game.on('message', messageRecieved);
function messageRecieved() {
    console.log('Ok');
}

game.on('messageError', messageRecievedError);
function messageRecievedError(messageError) {
    console.log('error: ' + messageError);
}

game.concreteFunction();
```

Código 6-2. Recibir eventos.

Lo primero que se hace es obtener la librería que nos interesa y como es una función constructor, se construye un nuevo objeto. Lo siguiente y esto es lo más importante, es suscribirse al evento, diciendo que debe ejecutarse cuando ese evento salte.

Una vez realizadas las suscripciones necesarias, se lanza el método o función del objeto para que éste realice las operaciones necesarias y en cuanto ocurra algo, realizará las pertinentes llamadas.

- **Controladores**

Otro de los elementos más importantes es cómo actúan los controladores. Aunque en cada caso concreto la implementación puede variar, por ejemplo por las diferentes formas que llegan las peticiones o la interactuación con las diferentes librerías, el código básico no cambia:

```
var fs = require('fs');

function Controller() {
	this.jsonEvents = {};
	this.events = {};
	this.initFile();
}
exports.Controller = Controller;

Controller.prototype.initFile = function() {

	var i,
		objectDef,
		objectEvent;

	this.jsonEvents = JSON.parse(fs.readFileSync('./config-events.json', 'UTF-8'));

	for (i in this.jsonEvents.events) {
		objectDef = require('./events/' +
this.jsonEvents.events[i].eventName)[this.jsonEvents.events[i].objectName];
		objectEvent = new objectDef();
		this.events[this.jsonEvents.events[i].eventName] = objectEvent;
	}
}

Controller.prototype.handleRequest = function(request) {
	if (this.events.hasOwnProperty(request.eventName) &&
	    typeof this.events[request.eventName] === "object") {
		this.events[request.eventName].eventConcrete(request);
	}
}
```

Código 6-3. Controlador.

Las partes más importantes ocurren en los métodos o funciones de la función constructor '*Controller*'. En la función '*initFile*', se lee el archivo de configuración y con cada elemento que aparece en el fichero de configuración, se crea un nuevo objeto y se almacenan en un array que contiene todos los eventos.

En un caso general, la función '*handleRequest*', recogerá la petición y buscará si el evento que se solicita existe. En caso de que exista lo lanzará automáticamente.

6.4 Sumario

En el capítulo se ha realizado una especificación para seguir a la hora de la implementación del diseño realizado. También se han destacado las características más importantes del lenguaje de programación y se ha destacado los aspectos más importantes del código que se ha generado durante la implementación.

7 Interfaz gráfica de usuario

7.1 Importancia de la interfaz gráfica en los navegadores

Es quizá de los aspectos más relevantes en un desarrollo web, porque en definitiva es lo que el usuario va a ver. En muchos desarrollos probablemente no hace falta una interfaz gráfica muy elaborada, pero actualmente se requieren cada vez más mejoras que permitan al usuario interactuar con un arco más grande de posibilidades.

Con los estándares que se han ido elaborando a lo largo del tiempo han mejorado el aspecto visual sin tener que dedicar un tiempo que a veces es muy precioso. Por ejemplo, gracias a las hojas de estilo es más sencillo aplicar los estilos a toda una web, y que a veces no tengas que retocar ningún otro elemento.

En concreto las hojas de estilo hacen que el desarrollo web sea completamente diferente a otro tipo de desarrollo, ya que independizan muchos aspectos visuales. Gran parte de las aplicaciones que se desarrollan para escritorio no existe esa independencia de los elementos visuales.

HTML5 agrega nuevas características sobre los estilos, que permiten controlar de una forma más sencilla aspectos que anteriormente no lo eran. Por ejemplo es más fácil ahora hacer los bordes redondeados sobre los elementos que se representan en una página web.

Aunque muchas de las mejoras que se consiguen, son sencillas o relativamente fáciles de aplicar, no siempre para un desarrollo web sea abarcable aplicarlo todo sin tener una plantilla inicial o alguna librería que haga ese trabajo más sencillo. Para este desarrollo en concreto se hace necesario un control mayor de los aspectos visuales y de muchos elementos de la interfaz gráfica, porque habrá que hacer muchos cambios en tiempo de ejecución cuando se van recibiendo o enviando peticiones.

7.2 Librerías utilizadas en el desarrollo

Las librerías a utilizar no solo abarcan a los aspectos visuales, sino más bien a hacer más fácil el desarrollo. Cuando un programador se tiene que enfrentar al desarrollo con Javascript en el navegador, se hace completamente inabarcable el realizar las pruebas en todos y cada uno de los navegadores existentes. Y más aún ahora que existen nuevos dispositivos.

Por eso las librerías permiten un control y una forma de hacer las cosas que te ayudan a la compatibilidad de diferentes tipos de navegadores y terminales, cosa que sin esas librerías es muy difícil de conseguir.

Por otro lado, a veces el exceso de uso de librerías no es la mejor opción, ya que te obliga a hacer las cosas del modo que se hacen con la librería, obligándote a aprender sus peculiaridades e incluso los defectos que pudiesen existir.

Por tanto, las librerías utilizadas se han intentado reducir al menor grupo posible, pero que a la vez permitan controlar el mayor número de aspectos, sin tener que dedicar un tiempo excesivo en su aprendizaje o uso.

- **jQuery**

  Es posiblemente la librería que más se usa en la actualidad en el desarrollo web. Es incluida también en muchas plataformas de desarrollo como elemento para desarrollar. Como su eslogan dice de haz más con menos, intenta simplificar el desarrollo con Javascript haciendo llamadas simples para poder realizar cualquier acción, ayudando a que sea más sencillo entenderlo y aprenderlo.

  Permite de manera sencilla seleccionar, interactuar y modificar cualquier elemento disponible en los documentos HTML, incluso la manipulación de los estilos y las hojas de estilo.

  Además permite manejar eventos de manera sencilla, con acciones y comandos comunes. También te permite realizar animaciones y efectos para producir páginas web más profesionales y más visuales al usuario.

  Es también muy extensible, ya que se pueden agregar plugins o extensiones de terceros, para poder realizar acciones más complejas, como por ejemplo galerías y efectos compuestos.

También, y esto es algo fundamental, permite hacer las acciones con total independencia del navegador. Cuando un desarrollador tiene que realizar alguna funcionalidad, por experiencia sabe que tendrá que hacer ciertas llamadas para unos navegadores y otras para otros. Pues con esta librería se intenta evitar que se tenga que realizar esas diferenciaciones, haciendo que todas sus acciones sean uniformes, sin tener que depender de ninguna forma del navegador.

- **socket.io**

Es una librería que actúa tanto en el lado del servidor como en el navegador, para permitir la comunicación en ambos sentidos, evitando las diferencias entre los distintos mecanismos de comunicaciones existentes.

Como la librería anterior, está pensada para que no dependa del navegador, siempre haciendo uso del mejor tipo de comunicación existente en el navegador. Por ejemplo si se hace uso de un navegador actual de escritorio, usará Websockets para comunicar el servidor y el cliente. Por el contrario, si usa un navegador antiguo, o quizá algún navegador de móvil con limitaciones, es posible hacer uso de otro tipo de comunicación existente, como por ejemplo el long polling.

Independientemente del mecanismo utilizado, al final el uso seguirá siendo el mismo, haciendo el mismo tipo de llamadas y sin depender de ajustes concretos en el navegador, haciendo que sea muy versátil y de un fácil manejo.

- **jQuery mobile**

Es una librería que lo desarrollan las mismas personas del proyecto jQuery. Su objetivo es el mismo que jQuery, hacer más con menos, unificando y optimizando el desarrollo para dispositivos táctiles.

Hace uso de HTML5 para manejar la interfaz gráfica en el mayor número de dispositivos móviles y táctiles existentes, aunque siempre manteniendo la compatibilidad con los navegadores de escritorio convencionales.

Está construido sobre jQuery, por lo tanto su curva de aprendizaje no es excesiva si se está familiarizado con esta librería. También hace uso de temas y estilos a la hora de visualizar los elementos HTML que se creen, permitiendo personalizarlos todos.

7.3 Aspectos destacables de la interfaz gráfica

Las librerías utilizadas permiten que la interfaz gráfica sea más fácil de mantener y manejar, pero eso no quita que sea necesario mantener una relación homogénea de los elementos que son necesarios visualizarse.

Lo positivo de usar estas librerías en concreto es que permiten que se independicen muchos aspectos que por otro lado son necesarios adaptarse a los diferentes navegadores. Y con la aparición de dispositivos táctiles y su alta proliferación, se hace necesario mantener una interfaz que pueda ser operativa en el mayor número de elementos sin tener que hacer ningún cambio relevante.

Para ello, los componentes que se visualizarán serán lo más sencillos posibles, realizando todas las acciones lo más simples y reducidas que se puedan. Aunque no es un objetivo principal, es preferible mantener una interfaz con pocos elementos a llenarla de información.

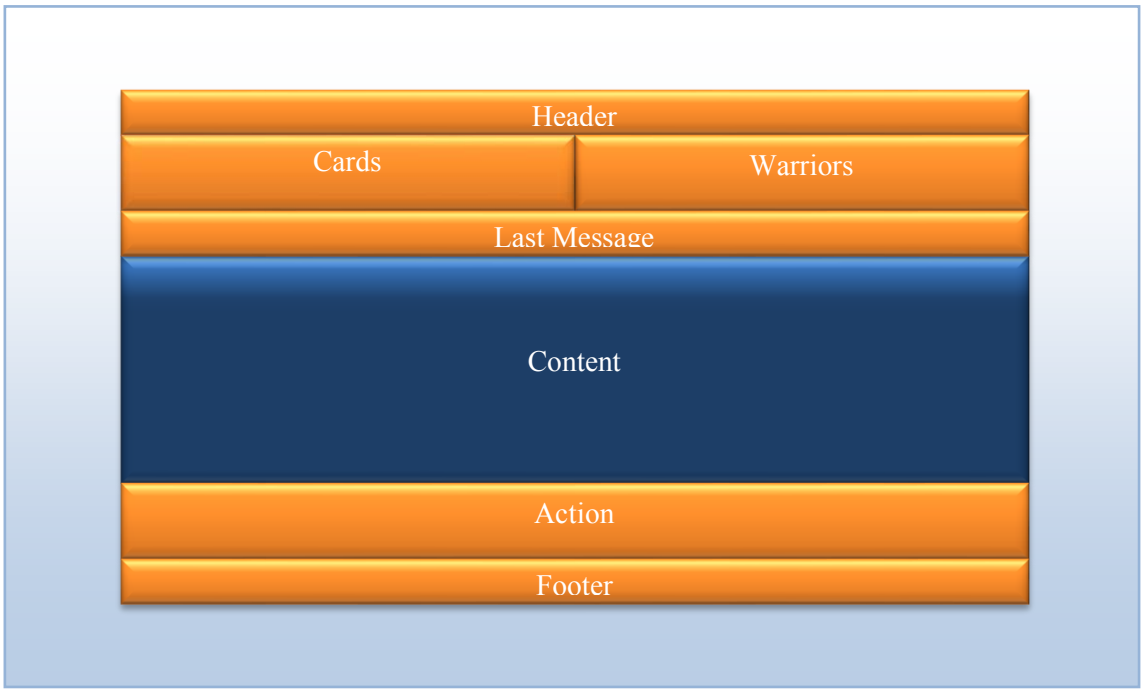

Ilustración 7-1. Boceto Interfaz Gráfica.

Como se muestra la imagen, habrá una cabecera y un pie de página, que estarán visibles en todo momento porque son los que contendrán las acciones que pueda realizar el usuario. Las acciones que a priori son más utilizadas, que son las de visualizar las cartas y los gladiadores, se encontrarán en la barra superior. También se mostrará el último mensaje recibido en la parte superior, pudiendo visualizar todos los mensajes recibidos una vez que se pulse por esta acción.

Todas estas acciones no afectan sobre la partida y sirven simplemente para visualizar el estado actual de la partida. No obstante, en la parte inferior se tendrán las acciones que hacen avanzar la partida y que se activará una vez se reciban los pertinentes mensajes. Las acciones que se pueden realizar son jugar una carta y jugar una apuesta. Estas acciones son excluyentes entre sí, con lo que sólo puede tenerse una acción activa cada vez, por eso solo aparece un único botón en la interfaz.

Gracias a la librería jQuery mobile es sencillo mostrar esta estructura con las etiquetas típicas, como se puede ver en el siguiente esquema:

```
<div data-role="page" data-theme="a" id="page1">
  <div data-theme="a" data-role="header" data-position="fixed">
    <h3>
      <a data-role="button" data-transition="fade" data-theme="a" href="page1" data-icon="plus" data-iconpos="right">
        Last Message
      </a>
    </h3>
    <div data-role="navbar" data-iconpos="top">
      <ul>
        <li>
          <a href="#page1" data-theme="a" data-icon="">
            Cards
          </a>
        </li>
        <li>
          <a href="#page1" data-theme="a" data-icon="">
            Warriors
          </a>
        </li>
      </ul>
    </div>
  </div>
  <div data-role="content"></div>
  <div data-theme="a" data-role="footer" data-position="fixed">
    <h3>
      <a data-role="button" data-transition="fade" data-theme="a" href="page1" data-icon="arrow-r" data-iconpos="right">
      Action
      </a>
    </h3>
  </div>
</div>
```

Código 7-1. Ejemplo página web jQuery mobile.

Esta estructura cargará en el navegador una página con los elementos deseados. El primer elemento con el atributo "*data-role=page*", se considera como una página y en su interior contendrá todos los elementos necesarios para mostrarse en el navegador. Si se tienen uno o más elementos con esa estructura, mostrará el correspondiente al ir navegando por cada uno de ellos, ya que en los enlaces se irá a cada elemento usando el identificador de cada página.

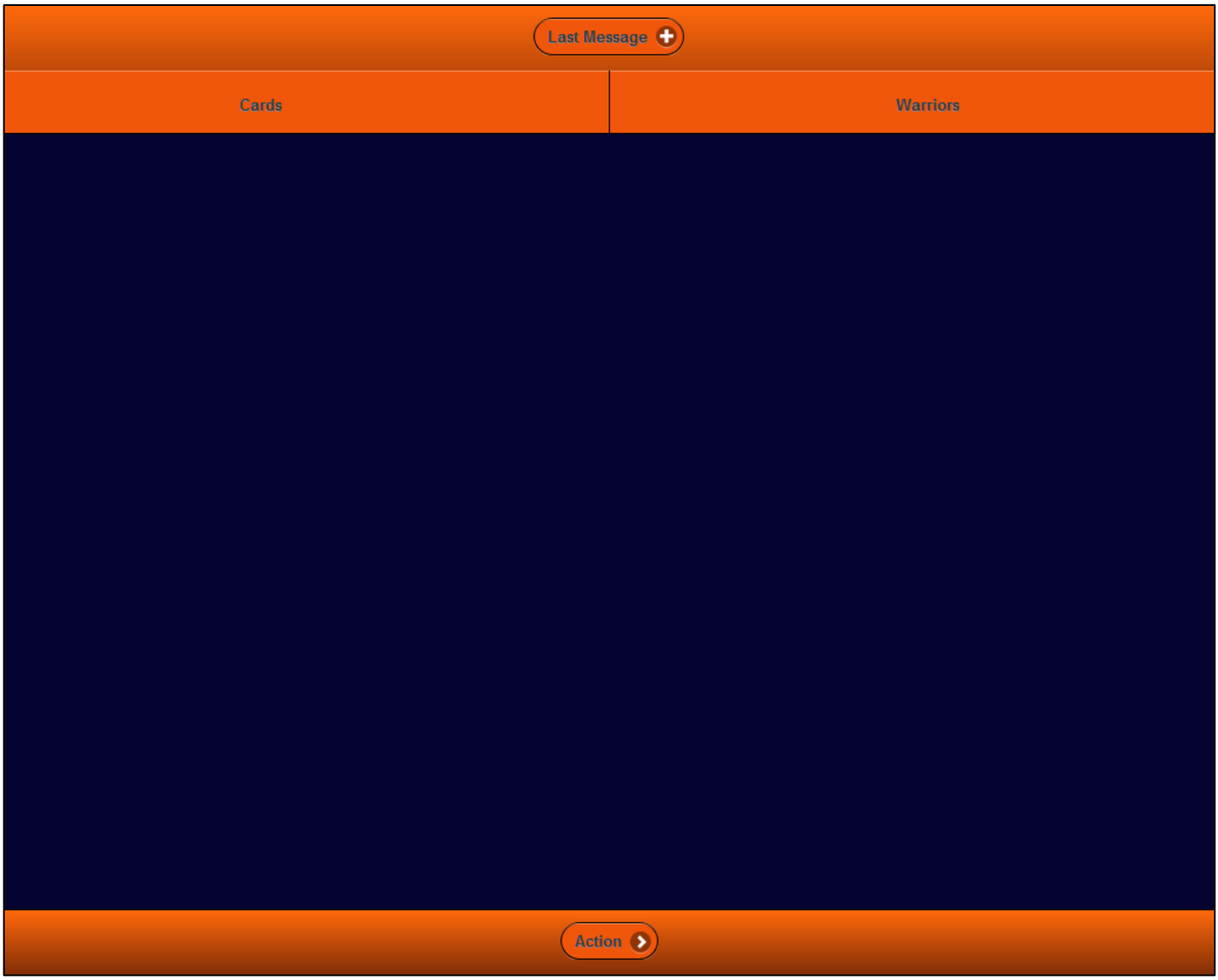

Ilustración 7-2. Ejemplo Interfaz Gráfica.

En la imagen anterior se puede observar un ejemplo de carga en un navegador web con la estructura anteriormente citada y con las librerías necesarias para mostrarlo.

7.4 Sumario

Se ha visto la importancia y relevancia del navegador y de los aspectos que le hacen importante de cara al usuario. Se han descrito las librerías utilizadas para la visualización, comunicación y compatibilización de los diferentes navegadores existentes en el mercado.

8 Evaluación

8.1 Pruebas realizadas en diferentes dispositivos

Probablemente uno de los apartados en los que más dificultades pueden existir. Los motivos de las dificultades principales son promovidos por la falta de fondos para adquirir cada uno de los diferentes dispositivos existentes que disponen de un navegador. Aunque también existe otro motivo muy importante, que es la disposición temporal de hacer todas y cada una de las pruebas de funcionamiento.

Conseguir el objetivo de que funcione un desarrollo web en el mayor número de dispositivos o terminales, o por lo menos de los navegadores más populares, es difícil. Al menos se hace verdaderamente difícil determinar un alcance claro para ese objetivo. Es difícil porque el alcance viene dado por tres preguntas que se deben realizar al abordar un desarrollo web que pueda funcionar en diferentes dispositivos, que están relacionados con las dificultades de este apartado:

- ¿Qué plataformas y navegadores son los más utilizados?
- ¿Qué navegadores son capaces de soportar las nuevas funcionalidades del nuevo estándar?
- ¿Qué dispositivos adquirir y qué simuladores utilizar para realizar pruebas?

Para responder se hace necesario disponer de datos lo más fidedignos posibles. Lo cual hace aún más difícil discernir una respuesta correcta, porque muchas estadísticas pueden venir de plataformas específicas, o de datos que pueden venir de las propias compañías que desarrollan el navegador.

Posiblemente mantener esos datos o esa información se puede considerar una ventaja competitiva, porque eso puede hacer que desarrollar para un dispositivo con poco uso no sea viable, y no solo afecta al desarrollo web, sino a cualquier tipo de aplicación. Esas preguntas al final se basan en tener los mejores datos posibles, que muchas veces y en este caso en concreto lo disponen las grandes compañías que ofrecen servicios por internet: Los grandes buscadores, servicios de correo o cualquier otro tipo de servicio usado globalmente.

En definitiva y resumiendo, como esa información no resulta pública o de fácil acceso, el objetivo puede quedar un poco en el aire. Pero por suerte, hay mucha información referente a

las librerías utilizadas que permiten no tener que realizar tantas pruebas en simuladores o la adquisición de dispositivos.

Todas ellas provienen de equipos de desarrollo y de testeo muy elaborados. Ofrecen sus características con listas de compatibilidades de navegadores y plataformas, facilitando en definitiva su uso.

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
	Navegadores de Escritorio					Navegadores Móviles			
jQuery	6+	1+	1+	1+	1+				
jQuery mobile	7+	4+	11+	4+	10+	3.1+	5+	11+	2.2+
socket.io	5.5+	3+	4+	3+	10+	3.1+			1.5+

Tabla 8-1. Compatibilidad librerías.

Como se muestra en la tabla anterior, las librerías funcionan en la mayoría de navegadores existentes y en versiones bastante tempranas. Se nota que en los de escritorio tienen una mayor compatibilidad, donde existe compatibilidad con versiones que tienen ya bastantes años de antigüedad.

Sin embargo, en los navegadores móviles, no existe tanta compatibilidad. El caso de jQuery es especial porque no se ofrece ningún test sobre esas plataformas, aunque es cierto que sí que hay funcionalidad total en todos los navegadores móviles reflejados. Para apoyar esa afirmación, jQuery mobile está destinado a esos navegadores y al ser una librería que se añade a jQuery, hace que todas sus funcionalidades sean operativas. No obstante, la tabla no muestra todos los navegadores existentes, aunque si se muestran los más comunes hoy en día.

En la siguiente tabla se muestra el porcentaje de uso existente de todos los navegadores seleccionados. Los datos utilizados provienen de "*StatCounter GlobalStats*", hasta el 12 de Abril de 2012 y aunque existan muchos más navegadores, los porcentajes serían ínfimos al respecto.

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
15ª Versión Anterior			4: 0.05%						
14ª			5: 0.06%						
13ª			6: 0.07%						
12ª		2: 0.08%	7: 0.05%						
11ª		3: 0.32%	8: 0.08%						

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
10ª		3.5: 0.36%	9: 0.06%						
9ª		3.6: 2.36%	10: 0.14%						
8ª		4: 0.45%	11: 0.14%		9: 0%				
7ª		5: 0.38%	12: 0.19%		9.5-9.6: 0.04%				
6ª		6: 0.36%	13: 0.16%		10.0-10.1: 0.02%				
5ª		7: 0.34%	14: 0.24%		10.5: 0.01%				
4ª	5.5: 0.01%	8: 0.65%	15: 0.23%	3.1: 0.01%	10.6: 0.03%			10: 0.01%	
3ª	6: 1.06%	9: 0.91%	16: 0.52%	3.2: 0.01%	11: 0.03%	3.2: 0.08%		11: 0.00%	2.1: 0.14%
2ª	7: 2.30%	10: 1.27%	17: 2.16%	4: 0.14%	11.1: 0.03%	4.0-4.1: 0.13%		11.1: 0.01%	2.2: 0.52%
Versión Anterior	8: 13.28%	11: 13.45%	18: 23.64%	5: 1.02%	11.5: 0.07%	4.2-4.3: 0.51%		11.5: 0.02%	2.3: 1.28%	3: 0.07%
Actual	9: 14.14%	12: 0%	19: 0%	5.1: 3.22%	11.6: 1.26%	5: 1.56%	5.0-6.0: 2.00%	12: 0.03%	4: 0.03%
Inmediata Siguiente	10: 0.03%	13: 0%	20: 0%	5.2: 0%	12: 0.02%				
Futuras		14: 0%	21: 0%						
Total:	30.82%	20.94%	27.8%	4.4%	1.49%	2.27%	2%	0.06%	2.04%

Tabla 8-2. Uso de navegadores según versión.

La elección de estos datos está motivada por ser uno de los portales de estadísticas más importantes del sector, recogiendo billones de valores al mes. Además ofrece mucha información pública sin necesidad de realizar nada al respecto. Pero el inconveniente, como ya se ha mencionado anteriormente, es que los datos quizá no sean totalmente reales. No obstante, se pueden considerar bastante cercanos a ellos.

A esta lista de compatibilidades con las librerías escogidas, se le tiene que añadir los elementos utilizados del nuevo estándar HTML5 que se utilizarán. Como enumerar cada uno de ellos puede ser demasiado exhaustivo, sólo se verán los realmente relevantes.

Las librerías utilizadas están pensadas para que en el caso de que no exista un elemento concreto disponible, use las características disponibles del navegador utilizado. Por tanto, de los elementos más importantes el que quizá pueda sufrir algún problema importante es el de uso de etiquetas de los documentos SVG.

El nuevo estándar permite incluir las etiquetas SVG en un documento HTML. SVG son las siglas para definir un lenguaje de gráficos vectoriales y escalables (Scalable Vector Graphics). De esta forma se pueden aprovechar muchos de los elementos que se han creado para realizar las pruebas en las fases preliminares iniciales al crear las reglas del juego. Casi todos los elementos generados son gráficos vectoriales, con lo que para la implementación final su inclusión podría ahorrar un tiempo en pensar las formas de añadir la información.

Pero el posible inconveniente es la compatibilidad existente en los diferentes navegadores:

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
	Navegadores de Escritorio					Navegadores Móviles			
		3.6						10.0	2.1
	6.0	9.0				3.2		11.0	2.2
	7.0	10.0	17.0			4.0-4.1		11.1	2.3
	8.0	11.0	18.0	5.0		4.2-4.3		11.5	3.0
Actual	9.0	12.0	19.0	5.1	11.6	5.0	5.0-6.0	12.0	4.0
Inmediata Siguiente	10.0	13.0	20.0	5.2	12.0				
Futuras		14.0	21.0						

Tabla 8-3. Compatibilidad SVG.

La tabla anterior muestra cómo han ido añadiendo la nueva funcionalidad en los navegadores: Para escritorio desde versiones más antiguas y poco a poco en las versiones móviles. Aunque no entre en la dinámica del proyecto, sería interesante el poder ofrecer una alternativa para las versiones en las que no esté soportado, pero eso ya estaría fuera del ámbito, a no ser que fuese realmente un elemento que no estuviese implementado en la mayoría de ellos. Cosa que no ocurre, ya que casi el 70% de los usuarios, tienen soportado este elemento, sin tener en cuenta que con el paso del tiempo este porcentaje será aún mayor al actualizarse a las nuevas versiones que lo soportan.

8.2 Pruebas en la comunicación entre navegador y servidor

Está claro que esta característica es primordial en un proyecto web. Y más aún para una aplicación de estas características, en el que se hace necesario mantener una comunicación bidireccional entre el usuario y en consecuencia el navegador, con el servidor. Más importante aún se hace para transmitir tanta información de un lado a otro.

La web ha estado largamente construida sobre el paradigma llamado petición/respuesta del protocolo HTTP. Un navegador carga una página web y no ocurre nada hasta que un usuario entra en una nueva página. Esto cambió con la aparición de AJAX, haciendo que la web fuese más dinámica, pero aún así se requiere que el usuario siga interactuando para la carga de nuevos datos. O eso, o hacer que la aplicación realice periódicamente peticiones para cargar nuevos datos, técnica conocida como "*Polling*".

Con el tiempo la tecnología permitió que los servidores pudiesen mandar datos en el momento en que estuviesen listos, que es una forma de mantener la conexión abierta entre el cliente y el servidor, engañando al servidor de que ha iniciado la comunicación con el cliente. Esto se conoce como "*long polling*", donde el cliente abre una conexión HTTP con el servidor el cual la mantiene abierta un tiempo o hasta que tenga una respuesta. En cuanto tiene una respuesta el servidor la manda, y se inicia una nueva conexión tanto si se ha enviado algún dato como respuesta o se ha acabado el tiempo.

El inconveniente de estas técnicas es que se sobresatura el protocolo HTTP, que no está pensado para aplicaciones que necesitan una baja latencia, necesario en los desarrollos de tiempo real. Para ello, en el nuevo estándar se inició una especificación para mantener y establecer una conexión entre el navegador web y el servidor, donde ambos puedan mandarse información en cualquier momento. A esto se le conoce como "*Web Socket*" y está destinado a no saturar el protocolo, sino más bien a ser un protocolo más de comunicación que servirá para las aplicaciones con necesidad de baja latencia.

Una primera diferencia importante es que se reduce la información redundante que se envía. Por ejemplo, la cabecera de una petición y una respuesta con HTTP puede tener alrededor de unos 800 bytes, y eso sin enviar ni un solo dato.

```
GET /Polling HTTP/1.1
  Host: localhost:8080
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5)
        Gecko/20091102 Firefox/3.5.5
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Language: en-us
  Accept-Encoding: gzip,deflate
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
  Keep-Alive: 300
  Connection: keep-alive
  Referer: http://www.example.com/PollingStock/
  Cookie: showInheritedConstant=false; showInheritedProtectedConstant=false;
  showInheritedProperty=false; showInheritedProtectedProperty=false;
  showInheritedMethod=false; showInheritedProtectedMethod=false;
  showInheritedEvent=false; showInheritedStyle=false; showInheritedEffect=false
```

Código 8-1. Cabecera HTML de una petición.

```
HTTP/1.x 200 OK
Server: Node JS 6.17
Content-Type: text/html;charset=UTF-8
Content-Length: 21
Date: Sat, 19 May 2012 17:32:46 GMT
```

Código 8-2. Cabecera HTML de una respuesta.

Por supuesto es un ejemplo, puede que las peticiones contengan menos información, aunque también pueden tener más, incluso casos en los que puedan existir peticiones y respuestas que exceden los 2.000 bytes. Además si atendemos a que la información que se envía como respuesta, que sea de un pequeño texto, de por ejemplo 20 caracteres, no tiene sentido que la cabecera exceda de esta forma la información que se quiera enviar.

Extrapolando el ejemplo, si hay 1.000 usuarios realizando estas peticiones por segundo, o en vez de 1.000, que haya 10.000, las cifras pueden ser algo más importantes.

	Petición/Respuesta media	Bytes por segundo	bps	Mbps
1.000 clientes	800	800.000	6.400.000	6,10
10.000 clientes	800	8.000.000	64.000.000	61,04
100.000 clientes	800	80.000.000	640.000.000	610,35

Tabla 8-4. Ejemplo de peticiones HTML por número de usuarios.

En la tabla anterior se ve que las cifras son algo más importantes a medida que hay una interacción más importante y donde la información que se envía es totalmente innecesaria, pero un mismo ejemplo donde la información se envía con una solicitud por medio de "*Web Socket*" donde el cliente ya no necesita realizar una petición cada vez que se necesita alguna información, los resultados ya son diferentes al ser la cabecera de tan solo 2 bytes.

	Petición/Respuesta media	Bytes por segundo	bps	Mbps
1.000 clientes	2	2.000	16.000	0,015
10.000 clientes	2	20.000	160.000	0,153
100.000 clientes	2	200.000	1.600.000	1,526

Tabla 8-5. Ejemplo de peticiones Websocket por número de usuarios.

La tabla muestra cómo se reduce sustancialmente el envío de datos innecesarios a través de las redes, siendo la reducción en torno al orden 500:1, dependiendo del tamaño de la cabecera esto puede ser mayor o menor, incluso del orden de 1000:1 en reducción.

En el caso de la latencia, la diferencia no será tan inmensamente grande, pero sí que se notará cierta mejora. Usando el ejemplo anterior, la petición que se realiza por medio de HTTP necesita de enviar la petición y obtener una respuesta. Si ya existe un retardo entre el inicio del envío de una solicitud y la llegada de la respuesta, esto quiere decir que tardará normalmente el doble de tiempo en esperar por la petición que una petición con el otro mecanismo.

No obstante, con las últimas actualizaciones del protocolo HTTP que permite que las respuestas se puedan enviar en varias partes, se puede mantener una conexión abierta engañando de esta forma al navegador y consiguiendo que el servidor envíe respuestas sin una petición previa. Este mecanismo no quita la información de la cabecera que puede sobresaturar las peticiones, pero si las reduce en gran parte al igual que la latencia.

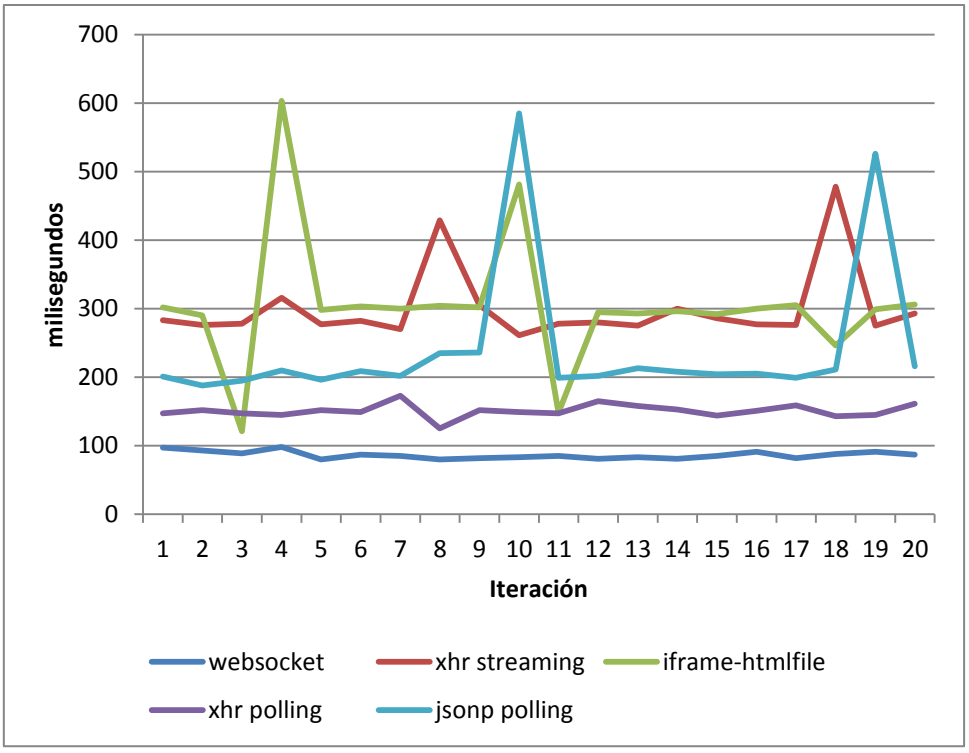

Ilustración 8-1. Tiempos de respuesta.

El gráfico anterior muestra los tiempos en unas tomas realizadas y usando los diferentes protocolos existentes. A continuación se muestra el promedio obtenido:

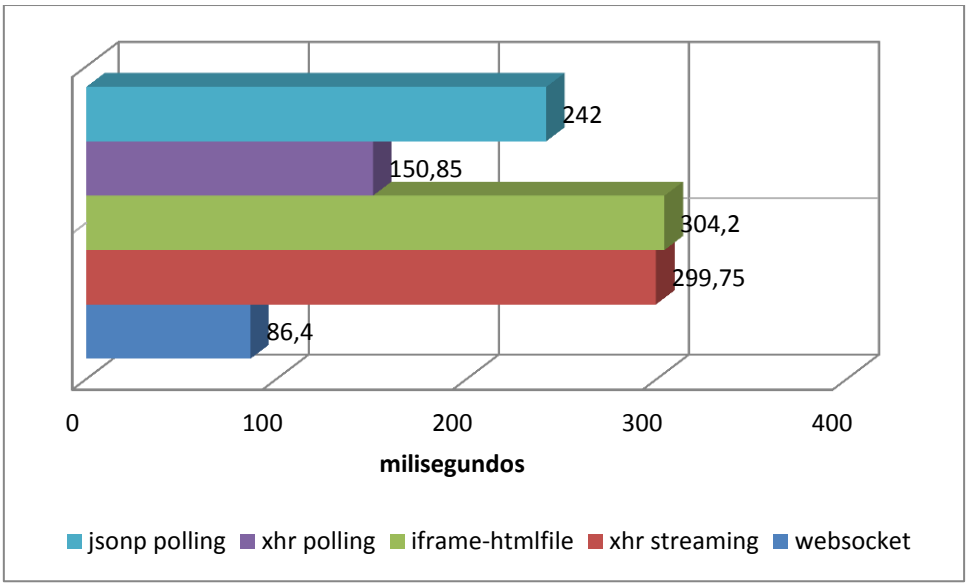

Ilustración 8-2. Tiempo medio de respuesta.

Se puede observar en ambos gráficos que el nuevo protocolo mejora las latencias que se obtienen con los otros mecanismos, del orden de al menos la mitad de tiempo ahorrado. Para una aplicación crítica en cuanto al tiempo de respuesta esto es muy importante y hace que el mecanismo de comunicación a utilizar sea muy importante.

Para este caso en concreto, el tiempo de respuesta no es esencialmente importante, aunque sin duda, cuanto mejor sea las respuestas al usuario serán más propicias. Pero el problema vuelve a pasar en este caso al ser HTML5 un estándar en preparación y que no todos los navegadores implementan cada una de las nuevas funcionalidades.

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
	Navegadores de Escritorio					Navegadores Móviles			
		3.6						10.0	2.1
	6.0	9.0				3.2		11.0	2.2
	7.0	10.0	17.0			4.0-4.1		11.1	2.3
	8.0	11.0	18.0	5.0		4.2-4.3		11.5	3.0
Actual	9.0	12.0	19.0	5.1	11.6	5.0	5.0-6.0	12.0	4.0
Inmediata Siguiente	10.0	13.0	20.0	5.2	12.0				
Futuras		14.0	21.0						

Tabla 8-6. Compatibilidad Websockets.

La tabla anterior muestra claramente que no todos los navegadores soportan este nuevo protocolo, no llegando a tener el 50% de los usuarios este protocolo incluido en su navegador. Lo que hace muy necesario el tener la posibilidad de implementar algún mecanismo que permita usar la mejor tecnología que disponga el navegador donde se ejecute la aplicación. La librería socket.io hace exactamente esto: Si no se le especifica nada, escogerá de una lista ordenada, el mecanismo de comunicación. Si se le especifica la lista, cogerá según el orden establecido, el protocolo a utilizar si está disponible.

Por último, existe un último inconveniente con la nueva comunicación: Puede que la comunicación por el puerto utilizado no esté permitida, ya sea por un firewall, un antivirus o un HTTP Proxy o cualquier otro software que impida la comunicación. En este caso, aunque exista el nuevo mecanismo, si alguno de estos elementos no permite utilizarlo, gracias al poder usar anteriores tecnologías, no surgirá tal inconveniente.

8.3 Sumario

En este capítulo, se han realizado diferentes pruebas para ver el funcionamiento en diferentes dispositivos y analizando las librerías utilizadas. También se ha analizado el aspecto de la comunicación del navegador con el servidor, viendo las diferentes tecnologías que son implantadas en los navegadores, como los tiempos y la latencia existente con cada uno de ellos.

9 Conclusiones, futuros trabajos y valoraciones

9.1 Conclusiones

Durante el desarrollo de este proyecto hubo varios momentos clave que determinaron la evolución final del proyecto. A parte, la situación actual de muchas características no ayudaba para tomar las mejores decisiones.

El primer momento clave fue seleccionar exactamente las tecnologías a utilizar. Fue un verdadero quebradero de cabeza seleccionar alguna, ya que cada una intenta vender su propuesta de la mejor manera posible y a veces no es fácil hacer pruebas con unas y con otras. Quizá es un aspecto que deberían mejorar los productos ofertados, un caso especial ocurrió con la tecnología Java.

Un servidor muy popular en Java es Tomcat, el cual no ofrece de ningún modo una forma de comunicarte bidireccionalmente con el cliente, a no ser que se haga uso de terceras librerías. El uso de estas terceras librerías no es para nada sencillo y la inclusión con Tomcat es muy costosa de entender si es que alguna de las librerías ofrece tal compatibilidad. Y esto mismo ocurre con la mayoría de servidores. Al ser tan nuevo, van ofreciendo las funcionalidades a cuenta gotas sin tener nada claro cuándo se podrá tener.

Cuando empezó el proyecto había pocas cosas que te ayudasen a decidir, porque al final tendrías que dedicar un gran tiempo a investigar y ver si algún elemento llega a tener todas las funcionalidades que se requieren. Fue un trabajo muy difícil y que en algún momento se veían inviables muchos avances que se realizaban. Un día parecía que avanzabas un paso, pero al día siguiente retrocedías al principio. Algo que sinceramente parecía muy frustrante.

El segundo momento clave fue seleccionar una lógica de juego bastante elaborada que requiriese no ser ni sencilla ni excesivamente compleja. En una primera toma de contacto se podían seleccionar muchas ideas pero que al juntarlas no parecía fácil dar con la tecla. Ya sea por querer ser demasiado perfeccionista, o por querer todo detallado, no se conseguía de ningún modo un juego que satisficiese todas las expectativas.

Se requiere mucho tiempo para diseñar un buen sistema, y más aún hacer las pruebas y cuadrar los números. Al final no se conseguía un resultado óptimo, pero cambio tras cambio se conseguía algo más cercano a ello. También el diseño gráfico para realizar las pruebas era muy grande, quizá por inexperiencia, se intentaba hacer para las pruebas una versión muy detallada y con ello se perdía mucho tiempo, para que luego en un par de pruebas se decidiese otro sistema

que te volvía a hacer de nuevo todo el diseño gráfico. Como en el caso anterior, un día dabas un paso y al siguiente volvías al principio.

El tercer momento clave podría decirse que era familiarizarse con los aspectos concretos del lenguaje de programación. Uno puede pensar que conoce en parte un lenguaje de programación, pero a veces te puedes equivocar y darte cuenta que no lo conoces para nada. En muchos aspectos el lenguaje seleccionado puede tener vicisitudes con cualquier lenguaje orientado a objetos y hace que sea extraño, pero comprendiendo los dos o tres aspectos que los diferencian, puedes entenderlo completamente.

También es muy diferente programar para un navegador que para un servidor. Hacerlo con el mismo lenguaje puede a veces crear inconvenientes que al principio no se veían. La lectura tanto de libros como de los tutoriales de los expertos puede ayudar a discernir la mayoría de aspectos que te pueden ayudar a solventar estos problemas.

Quizá el último momento clave podría ser la interfaz gráfica. Es algo que no concibes hasta muy avanzadas las cosas, pero cuando tienes que plantearte cómo mostrarlas, es cuando te das cuenta que no es tan sencillo el cómo debería quedar. Más aún en el mundo de la web que ya tiene un lenguaje y sus hojas de estilo muy avanzados, que sin mucha ayuda puede ser complicado entenderlo.

Además, con la aparición de navegadores móviles y sus resoluciones reducidas, han cambiado completamente como mostrar la información, teniendo que hacer las cosas para dos tipos de pantallas: Escritorios y móviles. Esto ya es una opinión personal, pero creo que deberían aunarse esas dos nociones y hacer una interfaz lo más amigable posible, por ello creo que hacer sólo una es una mejor solución. Está claro que esto no siempre se puede conseguir y dependerá del tipo de proyecto que se desea hacer.

En definitiva, se ha conseguido discernir cómo funciona una aplicación desde su inicio hasta su finalización, viendo todos los apartados importantes. Al final no es lo fácil y sencillo que podría parecer una aplicación para un juego. Requiere mucho esfuerzo combinado, que entre varias personas se podría llevar siempre que el trabajo en equipo funcione.

9.2 Futuros trabajos

Una de las características a mejorar es la capacidad de guardar más información de las partidas, para realizar estadísticas más personalizadas. También dar la posibilidad de conversar entre todos los usuarios, tanto en una partida en concreto como en un foro más global.

Probablemente los futuros trabajos más interesantes serían poder ofrecer más tipos de juegos diferentes. Aunque la infraestructura no esté totalmente preparada para ofrecer más juegos, sí que está lista para ello, haciendo que muchos de los módulos y componentes tengan la posibilidad de añadir más funcionalidades sin tener que cambiar partes de código.

Una posibilidad sería ofrecer algún tipo de juego que necesite de una comunicación más rápida, para poder comprobar si el nuevo protocolo de comunicación puede soportar aplicaciones en tiempo real.

Las alternativas son muy variadas a partir de este momento, con lo que poder sacar más partido a los nuevos elementos ofrecidos por el nuevo estándar está aún por probar su total utilidad.

9.3 Valoración personal

En particular, creo que el trabajo desde el principio ha sido bastante frustrante. En algunos momentos he visto que no había manera de avanzar o de encontrar una solución factible, que con el paso del tiempo no ha mejorado sustancialmente.

Personalmente creo que en este caso en concreto el trabajo en solitario es muy difícil y agotador. Realizar a la vez una parte de investigación con un proyecto tan amplio como un juego, sin tener además un equipo de trabajo continuo no ha ayudado.

Una de las cosas positivas que se pueden sacar de un proyecto tan largo es de poner en práctica muchos conocimientos adquiridos durante todo este tiempo. Y por encima de todo, la capacidad de adaptarte a nuevos entornos en los que poder analizar y diseñar con más facilidad que en un principio uno tenía pensado.

Otro aspecto destacable es la capacidad de englobar un número de tareas y acotarlas temporalmente. Aunque con las capacidades obtenidas esto debería ser más sencillo no siempre es así, porque se nota mucho que al principio estos cálculos ni llegan a acercarse a la realidad, pero al ir finalizando partes, conoces el entorno en el que te mueves y las características de cada uno. Dónde obtiene uno mismo una mayor rentabilidad al tiempo y dónde la puede perder. He de decir que al final un proyecto de este estilo, me ha hecho más realista y con una mayor fortaleza para solventar los problemas que vengan más adelante.

Anexo I. Planificación y Presupuesto

Planificación

La planificación se ha realizado teniendo en cuenta que fue necesario realizar diferentes tipos de trabajos y que no es fácil diferenciar a qué categoría o tipo equivale a cada uno de ellos. También se ha realizado teniendo en cuenta las prioridades y objetivos fijados en su inicio.

Id	Nombre de la tarea	Duración	Comienzo	Fin	Predecesora
1	**Estudio inicial**	**35 días**	**01/06/11**	**19/07/11**	
2	Definir objetivos y características	8 días	01/06/11	10/06/11	
3	Investigar nuevas tecnologías Web	16 días	13/06/11	04/07/11	2
4	Características juegos online	11 días	05/07/11	19/07/11	3
5	**Logica del juego**	**80 días**	**13/06/11**	**30/09/11**	**1**
6	Definición inicial de reglas	15 días	13/06/11	01/07/11	2
7	Consultas históricas	5 días	04/07/11	08/07/11	6
8	Diseño gráfico de reglas	20 días	11/07/11	05/08/11	7
9	Evaluaciones iniciales	5 días	08/08/11	12/08/11	8
10	Pruebas de campo	10 días	08/08/11	19/08/11	8
11	Rediseño de reglas	10 días	22/08/11	02/09/11	10
12	Rediseño gráfico	5 días	22/08/11	26/08/11	10
13	Pruebas de campo	5 días	29/08/11	02/09/11	12
14	Documentación de reglas	20 días	05/09/11	30/09/11	13
15	**Análisis de sistema**	**15 días**	**03/10/11**	**21/10/11**	**5**
16	Análisis inicial	5 días	03/10/11	07/10/11	
17	Definir alcance	10 días	10/10/11	21/10/11	16
18	**Estudio de viabilidad**	**52 días**	**21/10/11**	**03/01/12**	**15**
19	Definir alternativas	12 días	21/10/11	08/11/11	
20	Pruebas con alternativas	25 días	09/11/11	13/12/11	19
21	Descartar posibilidades	10 días	14/12/11	27/12/11	20
22	Seleccionar alternativa	5 días	28/12/11	03/01/12	21
23	**Diseño de la solución**	**33 días**	**21/10/11**	**07/12/11**	**15**
24	Analizar arquitecturas	10 días	21/10/11	04/11/11	
25	Seleccionar arquitectura	5 días	07/11/11	11/11/11	24
26	Definir descomposición	3 días	14/11/11	16/11/11	25
27	Especificar componentes	15 días	17/11/11	07/12/11	26
28	**Desarrollo**	**88 días**	**03/01/12**	**04/05/12**	**23;18**
29	Pruebas iniciales con alternativa seleccionada	18 días	03/01/12	27/01/12	
30	**Componente Modelo Conceptual**	**30 días**	**27/01/12**	**09/03/12**	**29**
31	Implementar lógica del juego	20 días	27/01/12	24/02/12	
32	Pruebas de la lógica del juego	10 días	27/02/12	09/03/12	31
33	**Componente Persistencia de Datos**	**10 días**	**27/01/12**	**10/02/12**	**29**
34	Analizar y definir alternativas	5 días	27/01/12	03/02/12	
35	Seleccionar e implementar alternativa	3 días	06/02/12	08/02/12	34
36	Pruebas de la persistencia de datos	2 días	09/02/12	10/02/12	35
37	**Componente Modelo de Aplicación**	**12 días**	**09/03/12**	**27/03/12**	**30;33**
38	Implementar controladores	8 días	09/03/12	21/03/12	
39	Pruebas de controladores	4 días	22/03/12	27/03/12	38
40	**Componente Interfaz Gráfica**	**28 días**	**27/03/12**	**04/05/12**	**30;33;37**

Id	Nombre de la tarea	Duración	Comienzo	Fin	Predecesora
41	Analizar y definir alternativas	8 días	27/03/12	06/04/12	
42	Seleccionar e implementar alternativa	5 días	09/04/12	13/04/12	41
43	Pruebas de la interfaz gráfica	15 días	16/04/12	04/05/12	42
44	**Evaluación de la solución**	**35 días**	**04/05/12**	**22/06/12**	**28**
45	Pruebas finales de la solución	25 días	04/05/12	08/06/12	
46	Pruebas con diferentes dispositivos	5 días	11/06/12	15/06/12	45
47	Pruebas de comunicación	5 días	18/06/12	22/06/12	46
48	**Documentación**	**200 días**	**30/09/11**	**06/07/12**	**5**
49	Escritura memoria	195 días	30/09/11	29/06/12	
50	Revisión	5 días	02/07/12	06/07/12	49

Tabla 0-1. Listado de tareas.

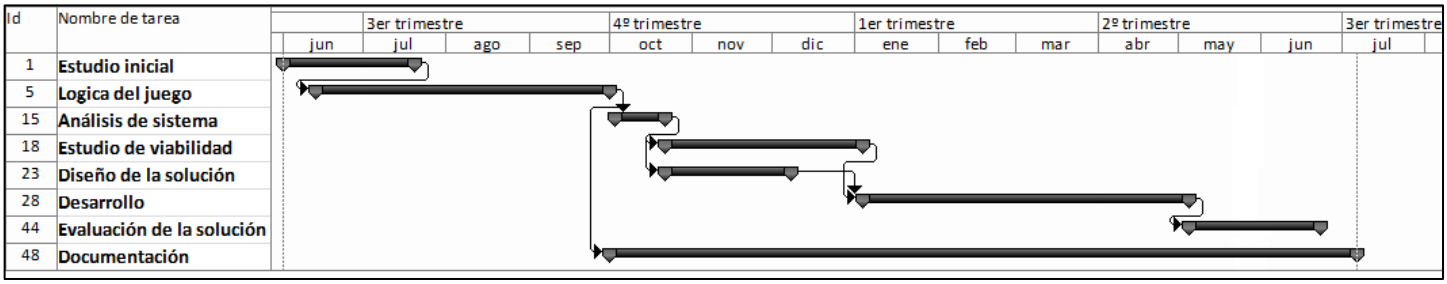

Ilustración 0-1. Diagrama de Gantt.

Se puede observar en la figura y tabla anteriores, las diferentes tareas que exigen el trabajo diverso que requiere unos conocimientos diferenciados. Lo que ocurre que no es claro cuándo se aplica en cada caso en muchas de las tareas, por ejemplo al realizar un diseño gráfico e implantarlo en el desarrollo de la aplicación, que posteriormente requiere alguna modificación para adecuarlo correctamente, entonces entra dentro de varias categorías. En un desarrollo con un equipo de trabajo que tiene bien diferenciadas esas categorías y tareas, no se producirían estos casos.

A continuación se muestra una lista de los roles que se han empleado, pero no se incluyen las horas necesarias para llevar a cabo la actividad relevante, sino el coste de cada rol en términos salariales, que luego se utilizará para calcular el presupuesto.

Cargo	Tarea Principal	Salario Bruto	Coste / Hora
Analista	Análisis y Diseño	35.000	19,15
Diseñador Gráfico	Diseño gráfico de reglas e interfaz gráfica	24.000	13,13
Programador	Desarrollo e implementación	28.000	15,32
Responsable de Documentación	Documentación del proyecto	26.000	14,22
Responsable de Pruebas	Pruebas	26.000	14,22
Expertos (Historiador)	Consultas históricas de lógica del juego	24.000	13,13

Tabla 0-2. Categorías y Salarios.

Para el cálculo de coste por hora se ha utilizado la fórmula para saber las horas que se hacen al año, que es la siguiente:

*[Semanas al año (52) – Semanas de Vacaciones (4,3) – Semanas de Festivos (2)] * Horas Semanales (40) = [52 – 4,3 – 2]*40 = 1828.*

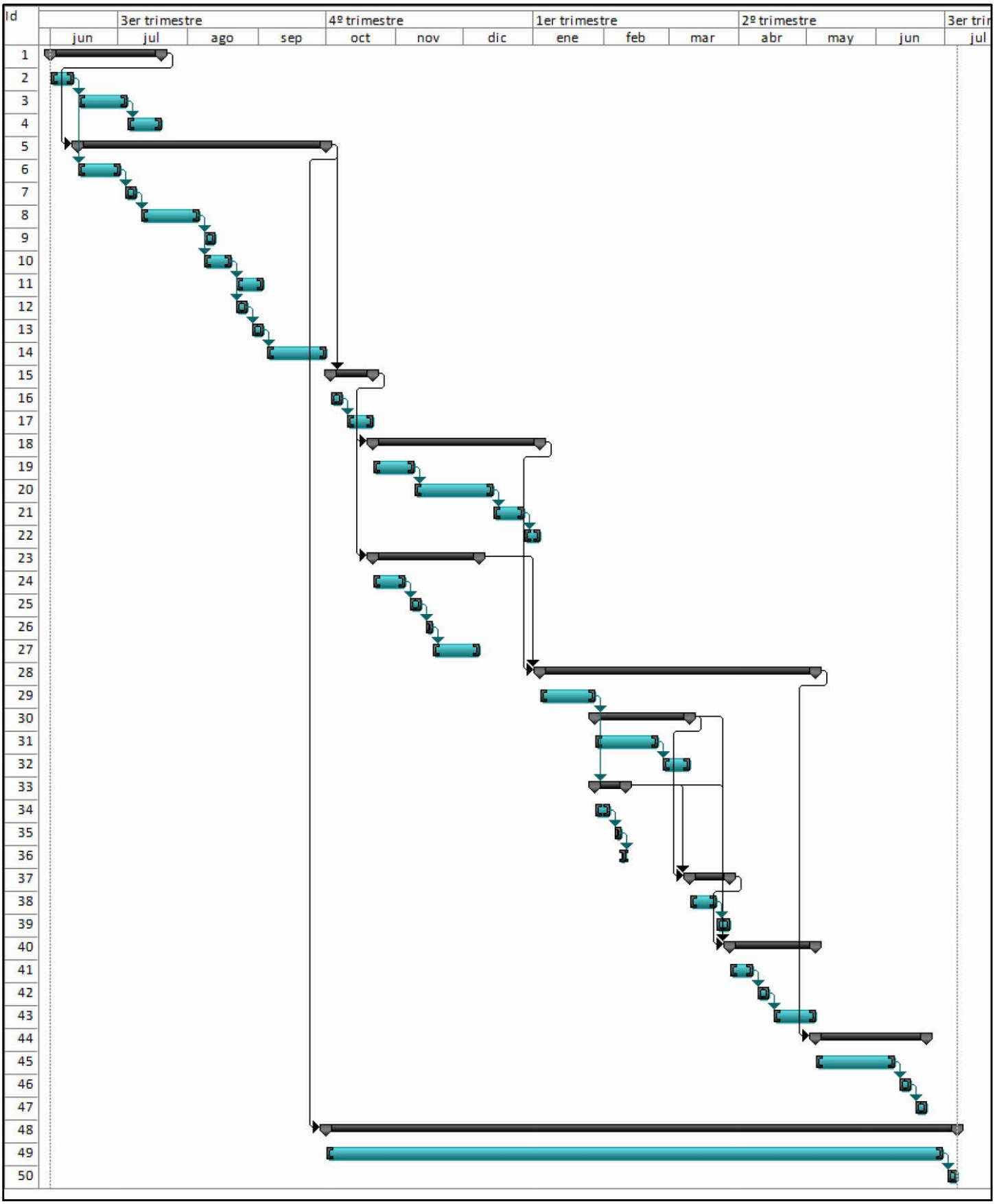

Ilustración 0-2. Diagrama de Gantt detallado.

En la figura anterior se muestra el desglose de la planificación del proyecto. Como ha sido necesario realizar desde un principio muchas pruebas o lectura de mucha documentación para encontrar una solución que fuese apta, la planificación ha sufrido continuos cambios, en especial al exigirse una expansión temporal.

Presupuesto

Para elaborar el presupuesto se hace uso de la planificación planteada en el apartado anterior, más todos los gastos derivados de los elementos necesarios para su elaboración.

- **Recursos materiales: software y hardware**

  Las herramientas y aplicaciones utilizadas para elaborar el proyecto y que han sido necesarias para ayudar y simplificar su desarrollo. Se incluyen todos los recursos tangibles necesarios para elaborar cada una de las partes del proyecto, como puede ser el sistema operativo, el PC o el coste mensual de la línea telefónica.

Concepto / Recurso	Coste
PC desarrollo	380
Microservidor para pruebas	190
Microsoft® Windows 7 Professional	309
Microsoft® Office Professional 2010	499
Microsoft® Project Standard 2010	775
Línea telefónica /ADSL (40€ Mensuales * 13 Meses)	520
Total	2673

Tabla 0-3. Recursos materiales.

El PC para desarrollo se ha considerado un equipo sin demasiadas prestaciones para un desarrollo de estas características, considerando la posibilidad de usar un producto portátil.

- **Gastos indirectos**

  Los gastos indirectos hacen referencia a costes operacionales en referencia por ejemplo a los viajes para celebrar reuniones, material de oficina o impresión de los elementos necesarios para hacer las pruebas de la lógica. Se deben incluir por ejemplo la electricidad, alquiler y amortización inmobiliaria entre todos esos gastos indirectos que afectan en buena medida al presupuesto. Para no hacer una lista demasiado detallada de cada uno de los elementos, se considera suficiente incluir como un 6% sobre el gasto de personal para considerar el gasto indirecto resultante al trabajo realizado durante este tiempo.

- **Gastos de personal**

  Con la explicación del apartado anterior de la planificación, el trabajo ha sido realizado por un Ingeniero Informático realizando diferentes roles en el proyecto. Como no hay una diferencia clara en algunos de los apartados para saber cómo imputar las

horas, se ha considerado realizar una media estimada al esfuerzo necesitado en la duración del proyecto. Por tanto el coste por hora vendrá especificado por las categorías más comunes en el respectivo proyecto:

Cargo	Porcentaje aproximado	Coste / Hora
Analista	30%	5,75
Diseñador Gráfico	5%	0,66
Programador	30%	4,60
Responsable de Documentación	20%	2,84
Responsable de Pruebas	10%	2,84
Expertos (Historiador)	5%	0,66
	Total	17,35

Tabla 0-4. Gastos de personal.

Las horas realizadas al día se consideran 8 y el total aplicado a este proyecto es de 288 días calculado en la planificación, dando como resultado un total de: 2304 Horas (288x8).

Concepto / Recurso	Coste / Hora	Horas
Ingeniero Informático	17,35	2.304
	Total	39.974,4

Tabla 0-5. Gastos totales de personal por horas invertidas.

- **Resumen de Presupuesto**

Concepto / Recurso	Total
Recursos materiales	2.673
Gastos de personal	39.974,40
Gastos indirectos (6% de Gastos de Personal)	2.398,46
Coste Total	45.045,86
Margen de Imprevistos (10% de Coste Total)	4.504,59
Beneficios (15% de Coste Total)	6.756,89
Precio Neto(Coste Total + Margen de Imprevistos + Beneficios)	56.307,34
I.V.A. (18%)[Impuesto al Valor Agregado]	10.135,32
Presupuesto Total (IVA Incluido)	66.442,66

Tabla 0-6. Resumen de presupuesto.

El Presupuesto final es de **66.442,66 €**, IVA incluido.

Bibliografía

[1] Stoyan Stefanov. “Javascript Patterns”. O`REILLY®, 2010.

[2] Daniel Pratt Mannix. “The way of the Gladiator”. Ibooks, Inc. ©, 2001.

[3] Mark Pilgrim. “HTML5: Up and Running”. O`REILLY®, 2010.

[4] Garann Means. “Node for Front-End Developers”. O`REILLY®, 2012.

[5] Manuel Kiessling. “The Node Beginner Book”. Leanpub, 2012.

[6] David Sawyer McFarland. “Javascript & jQuery: the missing manual”. O`REILLY®, 2011.

[7] Ryan Benedetti, Ronan Cranley. “Head First jQuery”. O`REILLY®, 2011.

[8] Brad Broulik. “Pro jQuery Mobile”. Apress ®, 2011.

[9] Ross Hermes, Dustin Diaz. “Pro JavaScript Design Patterns”. Apress ®, 2008.

Agradecimientos

Antes de realizar cualquier tipo de agradecimiento, me gustaría comentar las primeras frases que hago referencia al empezar la narración de este proyecto.

La primera cita es sobre el juramento que realizaban los gladiadores y que tiene su origen a las primeras insurrecciones aparecidas con Espartaco, poniendo en jaque al imperio más grande conocido de su época. Pero a parte de esa referencia, está más enfocado al grado de lo que podían enfrentarse en la arena. Más probablemente que a los gladiadores profesionales, se le puede atribuir a los esclavos y los condenados a muerte que tenían que enfrentarse entre ellos o contra cualquier tipo de animal. Un gladiador entrenado y preparado era muy caro como para dejarlo escapar con tanta facilidad.

Pero aparte de todas esas connotaciones, la lucha de gladiadores es quizá lo que empezó como el primero de los divertimentos para las personas que no tenían otra cosa que hacer. Y claramente, como una de las primeras barbaries que la humanidad ha ido creando a lo largo de su historia. La lectura del libro de Daniel P. Mannix que aparece en la bibliografía, da unas pequeñas explicaciones o ideas de lo que era ese espectáculo (por llamarlo de algún modo), mereciendo la pena su lectura para entender y comprender un poco por encima como era ese mundo.

Las siguientes citas son dos de los autores que más me han influenciado y que no quería dejar escapar la oportunidad de hacer un homenaje a sus obras. No obstante, para mí están relacionadas con el tema de la primera cita, incluso para obras que temporalmente son tan distantes unas de otras, tanto en el momento de su escritura, como en el momento que intentan reflejar sus escritos.

Ahora sí, me gustaría agradecer a tanta gente este trabajo que no me gustaría dejarme a nadie, espero que sepan perdonarme los que no aparezcan. Agradezco a toda la gente que ha compartido una parte de su tiempo conmigo, y que de algún modo han sabido soportarme.

En especial a mis padres, hermanos y cuñadas, porque son los que más tiempo han tenido que aguantarme.

A Óscar “Kibagami” por ayudarme a la hora de sacar un sistema de juego, o darme las suficientes ideas, espero que tenga mucha suerte en el mundo de los juegos de mesa.

A Santi “Txanti” por darme alguna de las ideas para el juego y por siempre ofrecer su ayuda.

A Chus, Piris, Fernao, Vicente, David, Mig, JC, Mario, por hacer las primeras pruebas y comprobar el sistema de juego, que hasta lo que es a día de hoy no tiene nada que ver en su inicio.

A Adrián y Néstor por estar a mi lado todos estos años.

A mis compañeros de Universidad Guillermo, Dani, Abel y Carlos, que han tenido que soportar quizá los peores años de carrera. En especial a Guillermo porque con el equipo de baloncesto uno acaba comprendiendo muchos tipos de frustraciones.

A Ester por ser mi compañera inseparable, que ha tenido que soportar los momentos más difíciles del proceso de elaboración de este proyecto y del día a día con alguien como yo. Es algo impagable.

Quería dedicar el proyecto a todos esos pequeños amigos que tenemos en nuestras casas. Todo el que tiene un amigo de éstos sabe lo importantes que se acaban convirtiendo con el paso de los años. Muchas veces sus pérdidas no son fáciles de llevar, pero es un proceso más en la fase de aprendizaje que debemos tener. Sobre todo para valorar las cosas que tenemos, de valorar el momento y saborearlo.

Da igual como seas, tanto física como mentalmente, o lo que haya ocurrido durante el día, porque cuando llegas a casa, ellos están ahí para estar a tu lado. Incluso si eres tú el que se va, estará aguardando tu llegada día tras día.

El proyecto va dedicado a ellos y en particular a todos los pequeños amigos que he tenido a mi lado en todo este tiempo. En especial a Newton, que este año se marchó dejando un hueco que difícilmente se podrá tapar. Y a Leo, que ojalá esté muchos años con nosotros.