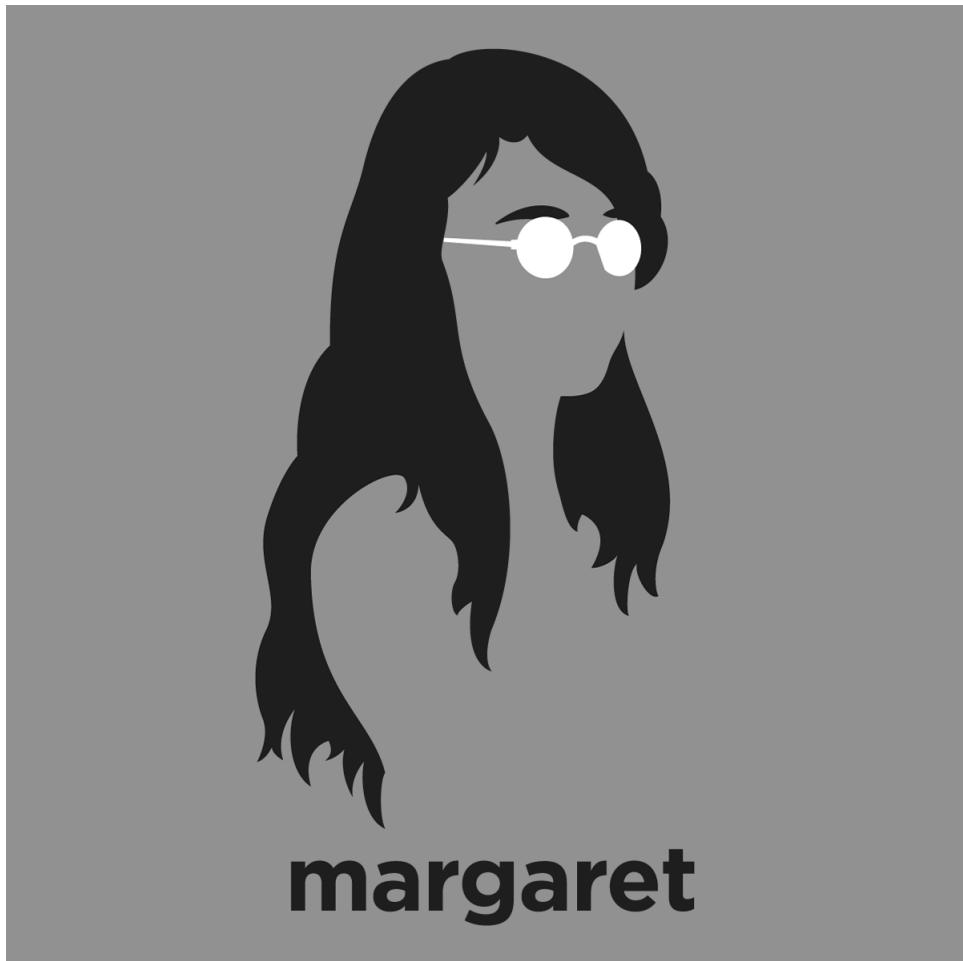


Equipo: MARGARET HAMILTON
Contacto: Julia 716185@unizar.es / Sergio 721057@unizar.es



PROYECTO SOTA Y REY

PLAN DE GESTIÓN, ANÁLISIS, DISEÑO Y MEMORIA DEL PROYECTO

1 de junio de 2018

Índice

| | |
|--|-----------|
| 1. Introducción | 3 |
| 1.1. Estructura del documento | 4 |
| 2. Organización del proyecto | 4 |
| 3. Plan de gestión del proyecto | 6 |
| 3.1. Procesos | 6 |
| 3.1.1. Procesos de inicio del proyecto | 6 |
| 3.1.2. Procesos de ejecución y control del proyecto | 7 |
| 3.1.3. Procesos técnicos | 7 |
| 3.2. Planes | 8 |
| 3.2.1. Plan de gestión de configuraciones | 8 |
| 3.2.2. Plan de construcción y despliegue del software | 9 |
| 3.2.3. Plan de aseguramiento de la calidad | 10 |
| 3.2.4. Calendario del proyecto y división del trabajo | 11 |
| 4. Análisis y diseño del sistema | 14 |
| 4.1. Análisis de requisitos | 14 |
| 4.2. Diseño del sistema | 16 |
| 4.2.1. Interfaz | 18 |
| 4.2.2. Gestor de mensajes (Netcode) | 21 |
| 4.2.3. Emparejamiento (Matchmaking) | 26 |
| 4.2.4. Lógica y BackEnd | 28 |
| 4.2.5. Base de Datos | 29 |
| 4.2.6. Inteligencia Artificial | 37 |
| 4.2.7. Despliegue | 41 |
| 4.3. Tecnologías elegidas | 41 |
| 5. Memoria del proyecto | 43 |
| 5.1. Inicio del proyecto | 44 |
| 5.2. Ejecución y control del proyecto | 45 |
| 5.2.1. Reparto del trabajo | 45 |
| 5.2.2. Comunicación interna | 52 |
| 5.2.3. Adecuación a las herramientas y tecnologías | 52 |
| 5.2.4. Control de versiones | 53 |
| 5.2.5. Pruebas del software | 54 |
| 5.3. Cierre del proyecto | 59 |
| 5.3.1. Planificación | 59 |
| 5.3.2. Comparación estimaciones iniciales | 62 |
| 5.3.3. Lecciones aprendidas sobre gestión | 63 |
| 5.3.4. Lecciones aprendidas sobre herramientas y tecnologías | 64 |
| 6. Conclusiones | 65 |
| Anexo I. Glosario | 78 |

| | |
|-----------------------------------|-----------|
| Anexo II. Actas | 78 |
| Anexo III. Presupuesto | 78 |
| Anexo IV. Diagramas diseño | 78 |

1. Introducción

El presente documento recoge el plan de gestión, análisis, diseño y memoria del proyecto de implementación de una aplicación web para jugar partidas de Guiñote online. La aplicación permite jugar a este conocido juego de cartas contra amigos o adversarios desconocidos de su mismo nivel ya sea en uno contra uno o por parejas. Además, si se prefiere, la aplicación ofrece una Inteligencia Artificial a la que poder retar para entrenarse sin necesidad de un contrincante real.

La aplicación no se limita a ser una plataforma para jugar partidas de Guiñote, sino que añade componentes de red social y funcionalidades para aumentar el atractivo del juego. El sistema utiliza un registro de perfil de usuario en el que se almacena un historial completo de cada jugador. Además, existe la posibilidad de visualizar partidas que se están jugando en ese momento en modo espectador. Las funcionalidades que incrementan la jugabilidad de la aplicación se basan en añadir competitividad. Existe un sistema de ligas al que se pertenece en función de las habilidades del usuario, medidas según el ratio de partidas ganadas y perdidas, y un sistema de torneos de eliminación directa a los que los jugadores pueden inscribirse. Como recompensa al desempeño realizado en los torneos y ligas existe una divisa virtual que los jugadores ganan y que se puede utilizar para comprar items en la tienda del juego, que les permiten personalizar sus partidas.

El sistema es accesible desde el navegador (al menos Google Chrome 64). Para que se pueda jugar desde distintos dispositivos la aplicación tiene una interfaz responsive, adaptándose al tamaño de la pantalla desde la que se juega. Este cambio de dispositivo puede hacerse incluso durante una partida en curso, siempre que no se agote el tiempo de turno correspondiente.

El sistema está alojado en Amazon AWS como un servidor ejecutado en una instancia virtual de EC2 mientras que todo lo referente al almacenamiento de información (datos jugadores, partidas, elementos de la tienda...) está almacenado en Amazon Aurora.

El desarrollo del proyecto está dividido en dos fases, lo que permite que pueda ser validado y contrastado con el cliente. La primera fase incluye un prototipo funcional de la aplicación que permite jugar partidas individuales, o en modo espectador e incluye las pantallas de login y perfil de usuario, aunque todavía no tenga implementada una Inteligencia Artificial funcional y competente. Esta versión del programa es mostrada al cliente el 9 de Abril de 2018.

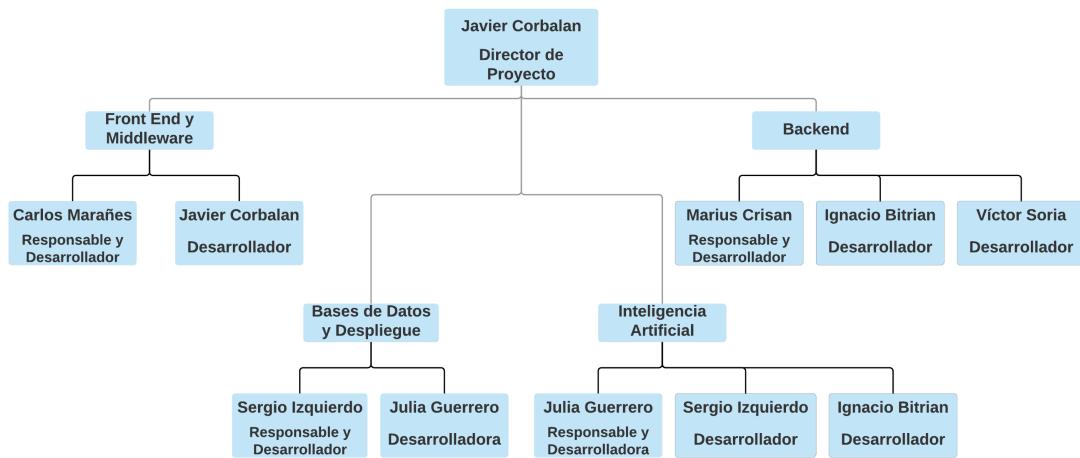
Como resultado final del proyecto el cliente recibe la aplicación desplegada en AWS, el código fuente y el resultado de la compilación para la plataforma objetivo. Además, el cliente recibe un manual técnico completo y una memoria resumen del proceso de desarrollo. La fecha de entrega del producto terminado es el 1 de Junio de 2018.

1.1. Estructura del documento

En la sección 2 se describe la organización del equipo de trabajo, incluyendo un organigrama de la empresa. A continuación, en la sección 3 se detalla el plan de gestión del proyecto. Esta sección incluye una primera parte relativa a cómo se llevarán a cabo las distintas tareas a lo largo del proyecto (subsección 3.1) y una segunda referente a los planes (subsección 3.2), en la que se detallan el plan de gestión de configuraciones, el de despliegue, aseguramiento de la calidad y el plan de trabajo especificado en un diagrama de Gantt. A continuación, se incluye en la sección 4 una descripción detallada de los requisitos funcionales y no funcionales del sistema, así como el diseño del sistema, detallado mediante una serie de diagramas de diseño. La sección 5 describe detalladamente cómo se han llevado a cabo los planes de la sección 3. Se divide en tres bloques: inicio del proyecto (subsección 5.1), donde se explican las fases iniciales de este, ejecución y control (subsección 5.2), en el que se detalla todo el desarrollo del proyecto, y cierre (subsección 5.3), donde se comparan las estimaciones iniciales con las finales y se reflexiona sobre las lecciones aprendidas. Finalmente, se incluye una sección 6 de conclusiones finales. En los anexos se encuentra el glosario (6), donde se explica en detalle el juego del guión, las actas de todas las reuniones con los profesores (6), el presupuesto y estimación de horas inicial (6) y diagramas de diseño más detallados (6).

2. Organización del proyecto

Para incentivar el trabajo en paralelo en proyecto, se han creado diferentes grupos de trabajo identificados por las tareas que conllevan:



1. Desarrollo del Frontend y Middleware: Incluye el desarrollo de la interfaz y las vistas de la aplicación, además de la gestión de la comunicación entre los jugadores y servidor durante el desarrollo la partida.
2. Despliegue e implementación de la base de datos: Incluye el análisis y diseño de la base de datos, la capa de acceso a datos, despliegue de la aplicación y mantenimiento.

3. Desarrollo de la inteligencia artificial: Incluye el análisis del juego, diseño e implementación de un agente de inteligencia artificial.
4. Desarrollo del Backend: Implementación de los servicios web, tratamiento de peticiones, generación de páginas dinámicas (servlets) y la lógica del juego.

La elección del director se ha llevado a cabo a través de una votación de mayoría simple.

3. Plan de gestión del proyecto

3.1. Procesos

El proyecto está dividido en dos iteraciones, cada una de ellas posee cinco fases: requisitos, análisis, diseño, implementación y pruebas. En la primera iteración se va a desarrollar el funcionamiento de una partida tanto individual como por parejas y la página web con las pantallas de login y perfil de usuario. En la segunda iteración se va a desarrollar el resto de funcionalidades del sistema, que incluyen la inteligencia artificial, torneos, tienda, entre otros. Cada fase de cada iteración tiene una duración máxima de una semana, excepto la fase de recogida de requisitos que tiene una duración inferior y que solo se lleva a cabo al inicio del proyecto. A lo largo del proyecto el equipo se divide en los grupos de desarrollo: bases de datos y despliegue, backend, frontend y middleware e IA. En cada uno de ellos la participación es flexible pero se mantiene de forma fija a un responsable.

3.1.1. Procesos de inicio del proyecto

Al iniciar el proyecto, cada grupo de desarrollo adquiere las herramientas necesarias para el desarrollo del proyecto. Las principales herramientas son el entorno de desarrollo IntelliJ, creando una cuenta gratis de estudiante y una cuenta de GitHub para poder llevar a cabo el control de versiones del código y documentación del proyecto. Además, se adquiere un certificado TLS gratuito obtenido através del servicio "Let's Encrypt".

Además, el equipo de bases de datos y despliegue debe registrarse en Amazon AWS con la cuenta de estudiante para la contratación de un servicio cloud en Amazon AWS y realizar un estudio para finalmente adquirir un dominio web. Al finalizar el proyecto estos recursos son traspasados al cliente.

Si aparece algún otro recurso necesario cuya adquisición necesite una aportación económica debe ser notificado al director y aprobado por este antes de contratar dicho recurso.

Dado que el proyecto integra un gran número de tecnologías y capas que conforman la arquitectura web, es necesario realizar un estudio de viabilidad de las tecnologías escogidas antes de realizar la adquisición del software de desarrollo, este estudio requiere 5 horas de trabajo entre todos los grupos. En un principio el desarrollo del proyecto se realiza sobre JavaEE, JSP, WebSockets, JavaScript, Bootstrap y MySQL. Como una parte del equipo de desarrollo posee cierta experiencia en el desarrollo web, no tanto en la utilización de estas herramientas, es necesario que se lleve a cabo un proceso de aprendizaje inicial. Este proceso requiere un número mínimo de horas para permitir al equipo comenzar con el desarrollo del proyecto. No obstante, a lo largo del proyecto la formación y la adquisición de experiencia son fundamentales.

En aquellas tecnologías donde el equipo de desarrollo no posee ninguna experiencia, como WebSockets, JavaScript , JSP, entre otros, se requiere un esfuerzo extra por parte del equipo para auto-formarse mediante la lectura de libros, tutoriales y la documentación pertinente.

- Tanto Carlos Marañés como Javier Corbalán se tienen que formar en Javascript, Websockets y Phaser.io a través del uso de ejemplos y documentación en línea.

- Julia Guerrero y Sergio Izquierdo deben formarse en HTML/CSS y JSP con documentación en línea y con la ayuda de los miembros del equipo que tienen experiencia en esa tecnología.
- El equipo entero se va a formar en la dinámica de trabajo con AWS gracias a la experiencia de Sergio Izquierdo y tutoriales en línea.

3.1.2. Procesos de ejecución y control del proyecto

Una de las funciones más importantes del director del proyecto es gestionar la comunicación dentro del equipo. Para garantizar que esta máxima se cumple, el director tiene la capacidad de convocar reuniones que incluyan a todo el equipo o solo a los responsables de cada grupo de desarrollo. En cada reunión se realiza un acta que recoge todos los aspectos y decisiones importantes acaecidas en la reunión. Las decisiones tomadas en estas reuniones deben trasladarse a la documentación del proyecto y finalmente a la implementación.

Los grupos de desarrollo se coordinan de forma autónoma para no sobrecargar la figura de director, para ello existe la figura de responsable, ya avanzada anteriormente. Si el equipo de trabajo sigue un correcto funcionamiento el director solo debe reunirse con los responsables de cada grupo, sin embargo en ocasiones extraordinarias puede reunirse con todo el grupo para tomar decisiones que engloben a todo el proyecto, o para corregir posibles funcionamientos incorrectos. Los objetivos globales de cada grupo son supervisados semana a semana por el director del proyecto. Mientras que dentro de cada grupo el responsable asigna día a día las tareas necesarias para cumplir con los objetivos. Los objetivos que semanalmente cada grupo de desarrollo se marca deben ser sencillos, concretos y trazables, de forma que permitan medir semanalmente el progreso del proyecto. Si una semana un grupo de desarrollo no cumple con las tareas asignadas el responsable puede reforzar el grupo con nuevos miembros y enfocando las tareas de forma diferente para completar esos objetivos antes de la semana siguiente.

En aquellas situaciones donde se requiera mediación, ya sean disputa o bajo rendimiento, el responsable del grupo de desarrollo debe intervenir realizando aquellas acciones que considere necesarias. En caso de disputa si su resolución no satisface a ambas partes, el director del proyecto junto con el responsable del grupo y las dos partes de la disputa se reunen para tomar las decisiones y resoluciones necesarias para finalizar la disputa.

El proyecto se almacena en un repositorio central donde se lleva el control de versiones. Al finalizar el proyecto se otorga al cliente el repositorio, el código fuente y el control de la aplicación ya desplegada. Al final de cada semana se debe estudiar el progreso del proyecto según las pautas marcadas en el diagrama de Gantt, pudiendo revisar y actualizar dicho diagrama.

3.1.3. Procesos técnicos

La generación de documentación del sistema se llevará a cabo por el propio equipo de desarrollo durante la implementación de la aplicación. Esta documentación será almacenada y actualizada dentro del repositorio central. Para generar la documentación general se seguirá el estándar UML utilizando la herramienta StarUML en su versión de prueba. En el caso de las diferentes clases y paquetes que componen el sistema se utilizarán las herramientas disponibles para generar la documentación automáticamente como Javadoc y JSdoc.

Para el desarrollo de los diferentes paquetes y componentes que componen el sistema se sigue la metodología de programación en parejas. Donde dos personas construyen una misma clase, uno escribe el código fuente mientras que el otro supervisa la corrección del código. A lo largo del proyecto cada equipo de desarrollo utiliza diferentes herramientas de desarrollo para facilitar la implementación del proyecto, la gestión de versiones y la compilación o interpretación de cada lenguaje utilizado. En general se utilizan los IDEs de JetBrains. IntelliJ IDEA para Java, WebStorm para JS, DataGrip para SQL y modelado de la base de datos.

El despliegue de la aplicación se lleva a cabo utilizando los servicios de Amazon AWS que facilitan un despliegue rápido y sencillo.

3.2. Planes

3.2.1. Plan de gestión de configuraciones

- La documentación está centralizada en un repositorio online. Se divide en varios archivos fuente de L^AT_EX en función de las secciones del documento. De este modo existen los siguientes archivos:

- *main.tex*: Contiene las macros de la documentación, la introducción y la estructura del resto del documento.
- *2-organización.tex*: Contenido de la Sección *Organización del Proyecto*.
- *3-gestión.tex*: Contenido de la Sección *Plan de Gestión del Proyecto*.
- *4-análisis-diseno.tex*: Contenido de la Sección *Análisis y Diseño del Sistema*.
- *5-memoria.tex*: Contenido de la Sección *Memoria del Proyecto*.
- *data.bib*: Base de Datos con la bibliografía usada.

Las imágenes, figuras y diagramas que se utilizan desde esos archivos están guardadas en la carpeta *figuras* con un nombre descriptivo. Además existe una versión compilada (PDF) de la última versión de los documentos así como los documentos anteriormente entregados con el nombre *main-aaaa-mm-dd.pdf*.

La *Propuesta Técnica y Económica* está disponible en formato Word en el fichero *propuesta.docx* al igual que su documento compilado (PDF) y los documentos anteriormente entregados con el nombre *propuesta-aaaa-mm-dd.pdf*

Las actas de reuniones están en la carpeta *actas* y sus nombres siguen el patrón *acta-aaaa-mm-dd*.

La contabilidad de las horas de trabajo empleadas por cada miembro del equipos se encuentran en un fichero de cálculo llamado *contabilidad-horas*.

- Para llevar un control de las versiones del código se crea un repositorio central en la misma organización de GitHub que la documentación y se utiliza el gestor de incidencias integrado en GitHub para resolver errores.
- Cada responsable de equipo designa las tareas de los miembros de su equipo utilizando el gestor de tareas propio de GitHub.

- Se deben realizar commits frecuentes siempre que el código compile. Los commits deben representar avances lógicos y atómicos de trabajo. Si el código subido afecta al de los compañeros se debe crear una incidencia para notificarles y recibir aprobación del resto del equipo para realizar el commit. Al programar en parejas, la revisión de los commits por parte de los compañeros de equipo (en parejas o tríos) se produce de forma natural. Los responsables de equipo son los encargados de realizar una copia de seguridad semanal off-site de su parte del repositorio mientras que el director del proyecto la debe realizar del repositorio completo. El director también es el responsable directo del control de las versiones entregadas al cliente.
- Los equipos realizan su trabajo sobre una rama basada en la rama central del repositorio (rama del equipo). Los miembros de los equipos realizan los commits sobre la rama del equipo (aunque pueden tener ramas auxiliares basadas en la rama del equipo). El responsable de cada equipo es quien puede incluir los avances hechos sobre una rama de equipo en la rama central del repositorio.

3.2.2. Plan de construcción y despliegue del software

- La construcción del software se desarrolla con la utilización del framework InteliJ.

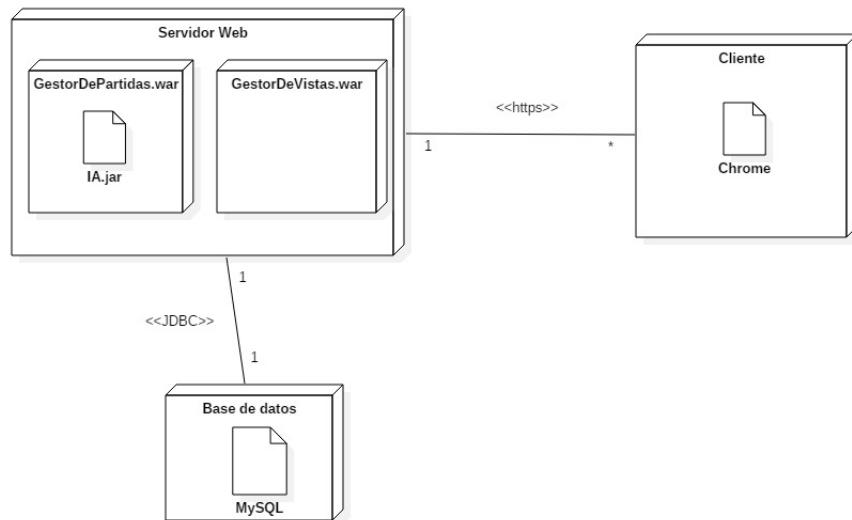


Figura 1: Diagrama de despliegue

- El software se despliega en tres niveles. El cliente, utilizando un navegador Chrome, interactúa con el software desplegado en el servidor web, a través de una conexión https. El servidor web mantiene el puerto 443 abierto para permitir las conexiones https efectuadas por los clientes. El software del servidor web se despliega en Amazon AWS, en una instancia de EC2, y este interactúa con la base de datos desplegada en Amazon Aurora para la creación de partidas o la modificación de los datos de un jugador. La comunicación entre el servidor web y la base

de datos se desarrollara en una intranet utilizando la API JDBC que permite la ejecución de operaciones sobre bases de datos.

3.2.3. Plan de aseguramiento de la calidad

Uno de los pilares del proyecto es el control de la calidad del software. Para ello el equipo intenta automatizar las tareas a este respecto todo lo posible y apoyarse en los siguientes pilares:

- Para garantizar el correcto funcionamiento de los paquetes, clases y funciones generados se realizan diferentes test unitarios. Para las clases cuyos métodos están formados por bucles complejos o varios condicionales se realizan pruebas de caja blanca utilizando la técnica de análisis de caminos. Para el resto de métodos no triviales se realizan pruebas de caja negra con clases de equivalencias y análisis de valores del límite para aquellos métodos que resuelven problemas con valores pertenecientes a unos rangos concretos. Para las pruebas unitarias en Java y Javascript, se utilizan JUnit y unit.js respectivamente. Estas pruebas deben ser satisfactorias antes de cada commit para, adicionalmente, dar un mínimo de garantías de funcionamiento correcto del código que se almacena en el repositorio.

Además, uno de los integrantes del equipo se va a dedicar a realizar pruebas manuales jugando varias partidas en un simulador en las fases iniciales del proyecto y en el sistema real una vez implementado.

- Para la integración de los diferentes módulos entre sí, se realizan test de integración.
- Como guías de estilo, se utilizan las de Google para Java, Javascript, HTML y CSS. En el caso de que a lo largo del desarrollo se introduzca algún lenguaje nuevo, los responsables de equipo y el director consensúan el uso de una guía de uso concreta. En caso de no existir una, se crea un documento con directivas importantes a seguir al utilizar ese lenguaje.
- Para representar y especificar el sistema tanto dentro como fuera de la organización, se utiliza el estándar UML, agilizando y concretando la comunicación entre equipos, evitando errores causados por una mala comprensión de la arquitectura del sistema.
- Se programa por parejas las partes críticas de la lógica del juego y de la aplicación para reducir el número de errores y mejorar la calidad del código en general.
- Cada 30 días se realiza una revisión de requisitos de la aplicación en la que se especifican los requisitos cumplidos y los pendientes.

3.2.4. Calendario del proyecto y división del trabajo



Figura 2: Diagrama de Gantt

Como se observa en el diagrama, el proyecto está dividido en dos iteraciones, con sus correspondientes demostraciones al cliente del avance del proyecto. La primera iteración finaliza la semana del 9 de abril y la segunda, que se corresponde con la entrega final, el 1 de junio.

Primera iteración

En la primera iteración se presenta una partida funcional individual, así como la página web con las vistas de las pantallas de login y perfil de usuario. También esta disponible el modo espectador. La inteligencia artificial está analizada pero no implementada todavía.

En concreto, los requisitos funcionales 2-6, 9-10 y los requisitos no funcionales 1-2 están cubiertos completamente. Además, el requisito funcional 1 está cubierto en cuanto a las partidas individuales, pero queda la implementación de las partidas por parejas. El requisito funcional 13 está cubierto parcialmente, ya que los turnos tienen un periodo de tiempo establecido, acabando la partida si el jugador no realiza movimiento, pero el sistema de puntuaciones no esta implementado. De la misma forma, relativo a los requisitos funcionales 14 y 17, se puede abandonar la partida manualmente pero todavía no hay penalización de puntuaciones. Finalmente, el requisito funcional 24 correspondiente al desarrollo de la inteligencia artificial, queda cubierto solo parcialmente, en lo que respecta a análisis y representación del problema pero no la implementación e integración con el resto del sistema.

Segunda iteración

En la segunda iteración o entrega final, se presenta al cliente el sistema con todas las características especificadas totalmente funcionales. Se añaden a las funcionalidades de la primera iteración todas las correspondientes a las puntuaciones, las ligas y los torneos (así como su programación automática), el sistema de matchmaking, la tienda, el panel de administración y la inteligencia artificial.

En concreto, se cubren por completo los requisitos funcionales 1, 7-8, 11-24 y los no funcionales 3-7.

División del trabajo

A continuación, se detalla una división del proyecto en bloques, con el correspondiente equipo o equipos de los descritos en la sección 2 que los lleva a cabo. Además, se incluye la lista de requisitos (especificados en el apartado 4.1) que quedan cubiertos en cada uno de estos bloques para garantizar que se satisfacen todos ellos.

| Bloque | Equipo | Requisitos |
|---|--------|--|
| Desarrollo de la interfaz del guiñote | 1 | RF: 1,6 RNF: 2 |
| Desarrollo de la lógica de juego del guiñote | 4 | RF: 1 RNF: - |
| Diseño e implementación de vistas web | 1 | RF: 2,4,6,7,8,9,11,15,18,19,20,21 RNF: 2 |
| Implementación de la web dinámica | 4 | RF: 1,6,12,13,14,16,17 RNF: 3,5 |
| Diseño e implementación de las comunicaciones | 1 | RF: 1,3,4,12 RNF: 1,2 |
| Desarrollo de la base de datos | 2 | RF: 2,4,7,9,11,13,14,15,16,17,18,19,20,22,23 RNF: 1,3,4,5,6 |
| Diseño e implementación de la Inteligencia Artificial | 3 | RF: 24 RNF: - |
| Despliegue | 2 | RF: 10 RNF: 1 |

Cuadro 1: Tabla de división del trabajo

4. Análisis y diseño del sistema

4.1. Análisis de requisitos

| GUIÑOTE | |
|---------|--|
| RF-1 | El sistema permite a los usuarios jugar al guiñote en modo uno contra uno y dos contra dos. |
| RF-2 | El sistema almacena un historial de partidas jugadas por el usuario y permite visualizar estadísticas de las partidas jugadas. |
| RF-3 | El jugador, en mitad de una partida, puede desconectarse y volver a conectarse desde cualquier dispositivo siempre que no sea su turno o, en caso de serlo, no agote su tiempo de turno. |
| RF-4 | El sistema requiere que los usuarios se registren con su correo electrónico o Facebook para poder acceder a los servicios que ofrece. |
| RF-5 | El jugador puede seleccionar si desea jugar una partida pública o privada. |
| RF-6 | Los usuarios pueden elegir una partida pública en curso y unirse a ella como espectadores. |
| RF-7 | El sistema cuenta con una divisa virtual que se consigue al iniciar sesión por primera vez, ganando torneos, partidas, ascensos a otra liga, etc. |
| RF-8 | El sistema consta de un panel de administración al cual se pueden conectar solamente los usuarios definidos como administradores con anterioridad. |
| RF-9 | El sistema permite al usuario administrar sus datos personales almacenados en el sistema: nombre de usuario, avatar, correo electrónico. |
| RF-10 | El usuario debe conectarse utilizando el navegador web Google Chrome para garantizar el correcto funcionamiento del sistema. |
| RF-24 | El usuario puede jugar contra un agente de inteligencia artificial, únicamente en el modo uno contra uno. |
| RNF-1 | El sistema garantiza la seguridad de la información de los usuarios. |
| RNF-2 | El sistema permite la conexión desde diferentes dispositivos. La aplicación es responsive por lo que se muestra de forma diferente para cada uno de los diferentes tamaños de pantalla. |
| RNF-3 | El sistema tarda menos de 20 segundos en encontrar partida aleatoria en caso de que haya un número de jugadores suficientes esperando para iniciar partida con las mismas características. |

Cuadro 2: Requisitos relacionados con la jugabilidad y el usuario

| LIGAS | |
|-------|--|
| RF-11 | Los jugadores tienen asociada una puntuación que varía en función de las partidas ganadas o perdidas. Dependiendo de ésta, pertenecerán a una u otra liga. |
| RF-12 | El sistema primero intenta emparejar a los jugadores de la misma liga. Si no es posible, los empareja con los de la liga más cercana a la suya. |
| RF-13 | Si el usuario no realiza ninguna acción durante su turno, la partida se termina y él recibe una penalización de puntuación. El turno es un periodo de tiempo preestablecido. |
| RF-14 | El sistema permite que los jugadores abandonen manualmente una partida de liga con una penalización asociada a su puntuación. |
| RNF-4 | Para ascender a una liga superior el jugador debe haber ganado muchas partidas. |

Cuadro 3: Requisitos de las ligas

| TORNEOS | |
|---------|--|
| RF-15 | El sistema permite a los usuarios inscribirse y participar en torneos. Los torneos son competiciones con eliminatorias directas a una partida única, en las cuales se pasa a la siguiente ronda ganando la partida. |
| RF-16 | Por cada ronda del torneo ganada el jugador recibe una puntuación proporcional a la ronda del torneo siendo la mayor bonificación la del ganador del torneo. |
| RF-17 | Los jugadores que abandonen una partida de un torneo son descalificados con una penalización asociada a su puntuación. |
| RF-18 | El administrador puede programar la creación automática de torneos ya sean puntuales o periódicos, especificando un momento de inicio y unos premios determinados. |
| RF-19 | El administrador puede modificar y eliminar torneos que aún no estén en curso. |
| RNF-5 | La puntuación recibida en cada fase por ganar una partida es mucho mayor que la recibida en una partida de liga y el doble que la de la fase anterior del mismo torneo. La puntuación para el campeón y el subcampeón es mucho más grande que la de los otros participantes. |
| RNF-6 | Los torneos programados inicialmente son semanales. |

Cuadro 4: Requisitos de los torneos

| TIENDA | |
|--------|--|
| RF-20 | El sistema posee una tienda para personalizar el tablero de juego, las barajas y el avatar, que inicialmente consta de 3 barajas, 3 tableros y 20 avatares diferentes. |
| RF-21 | El administrador de la aplicación puede añadir artículos nuevos a la tienda y modificar el precio de los existentes |
| RF-22 | Los artículos de la tienda se desbloquean en función de la liga más alta en la que ha participado el usuario en algún momento. |
| RF-23 | Los artículos se compran con la divisa virtual que el usuario tiene acumuladas. |
| RNF-7 | El valor en divisas virtuales de las barajas es más mayor que el de los tableros. El precio de los avatares varía para cada uno de ellos y para conseguirlos el jugador debe haber ganado muchas partidas. |

Cuadro 5: Requisitos de la tienda

4.2. Diseño del sistema

Se ha decidido implementar una aplicación web de 3 capas porque como se accede desde el navegador para hacer un cambio en la interfaz no hace falta una actualización en cada nodo cliente, siempre que la nueva interfaz sea compatible con Google Chrome 64 o posteriores. Además, como el modelo se aloja en un servidor distinto al de la base datos cualquier cambio en una de las partes no afecta al resto del sistema ya que se comunica mediante una interfaz bien definida al comienzo del proyecto. Otra ventaja de las aplicaciones en tres capas es que es muy fácil encontrar documentación y ejemplos de otras aplicaciones parecidas en internet ya que hoy en día son las más utilizadas. No se ha elegido un modelo de aplicación de 4 capas porque resulta un inconveniente que la interfaz se ejecute en un servidor central ya que sería un cuello de botella importante para el sistema dado que gran parte del tiempo de ejecución del sistema corresponde a la interfaz.

La distribución de las diferentes partes se puede apreciar en el diagrama de componentes de la figura 3

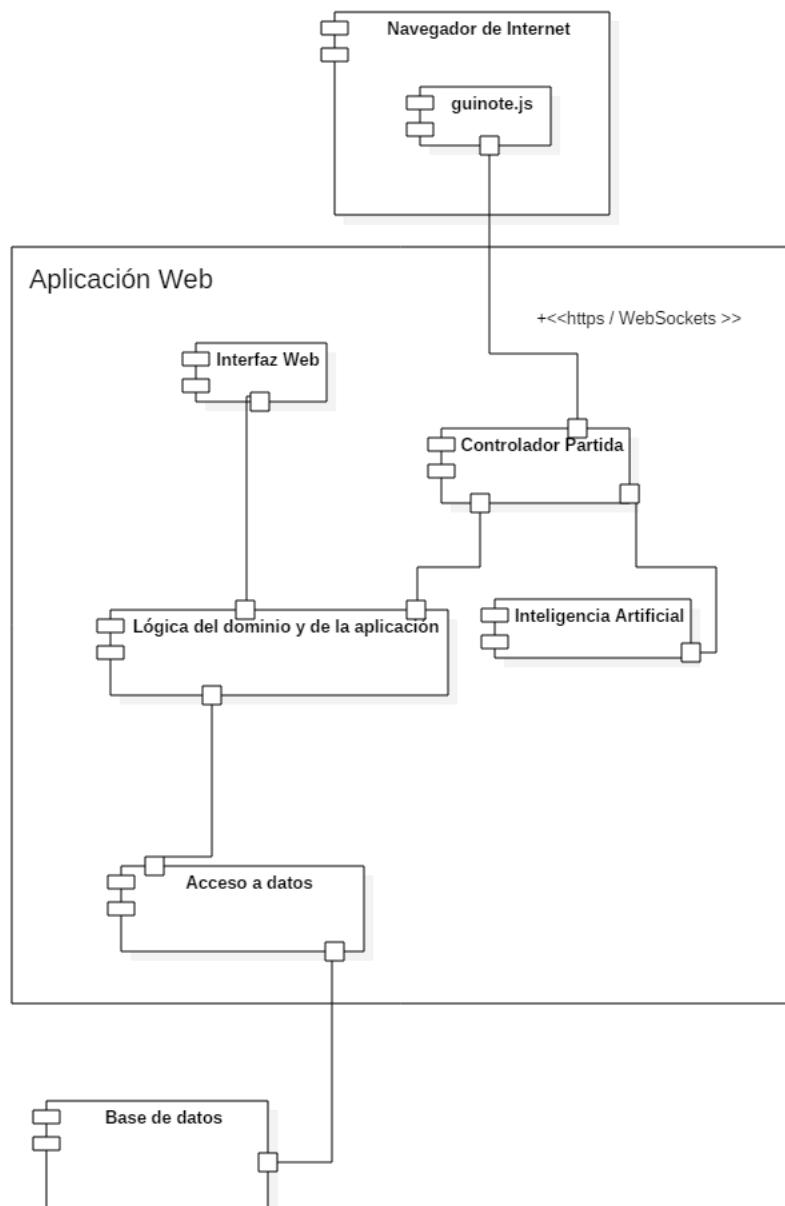


Figura 3: Diagrama de componentes donde se refleja la distribución del sistema

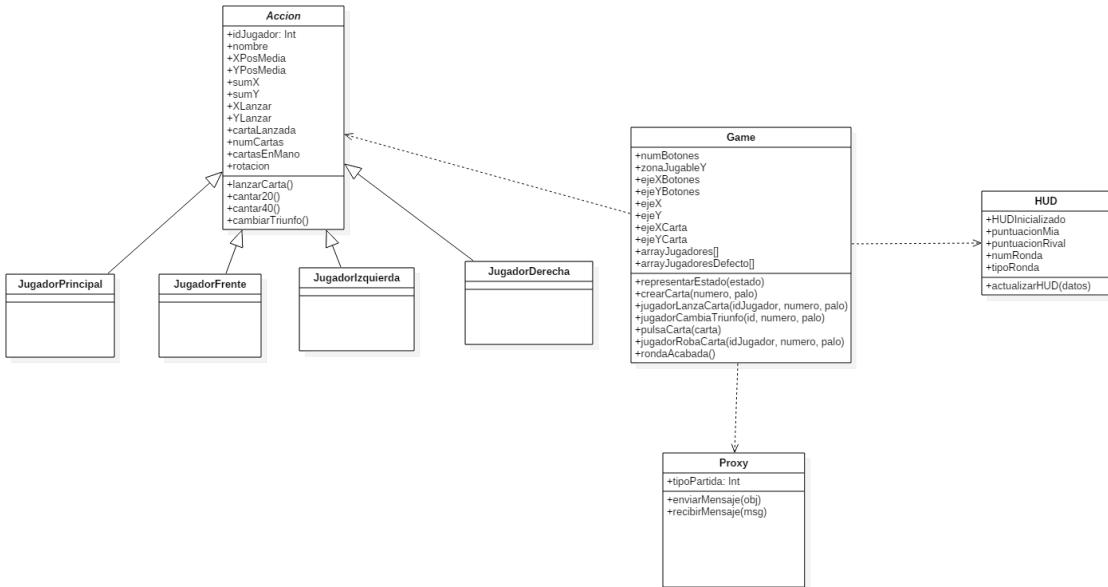


Figura 4: Diagrama de clases de la interfaz de juego

4.2.1. Interfaz

La interfaz se ha desarrollado utilizando la herramienta Phaser, basada en JavaScript. Se ha implementado de manera que sea fácilmente modificable ya que a lo largo de la producción del software se han de introducir cambios, por lo que se han parametrizado la mayoría de las funciones en medida de lo posible.

Una de las principales decisiones de diseño que se han tenido que llevar a cabo es como representar a cada jugador. Se ha hecho de manera que cada jugador tiene su posición como referencia y la interfaz es capaz de representar al resto en función de éste. Como hay un máximo de 4 jugadores, la interfaz toma cada posición como un rol, teniendo todos los roles los mismos parámetros simulando un patrón de diseño Strategy, para facilitar la integración. El diagrama de clases de la figura 4 muestra la implementación de la interfaz. Aunque en JavaScript no haya clases como tal, se ha aprovechado la sintaxis UML para visualizar las diferentes partes.

Otra decisión de diseño que sobre la que se ha reflexionado ha sido acerca de la comunicación entre la interfaz y el controlador de la partida. La tecnología que se ha escogido han sido WebSockets como se ha dicho previamente. Dicha tecnología permite únicamente comunicarse a través del paso de mensajes, por lo que se ha decidido que los mensajes sean en formato JSON para que sean fácilmente legibles para ambos extremos de la comunicación.

La aplicación web que no se corresponda con la parte jugable, es decir, el perfil del usuario, la vista del historial de partidas, ligas, torneos, etc. se implementa con JSP y servlets. Se puede observar el mapa de navegación inicial de la aplicación en la figura 5.

Una vez se ha desarrollado el sistema, con todas las vistas ya implementadas, el mapa de navegación resultante se observa en la figura 6.

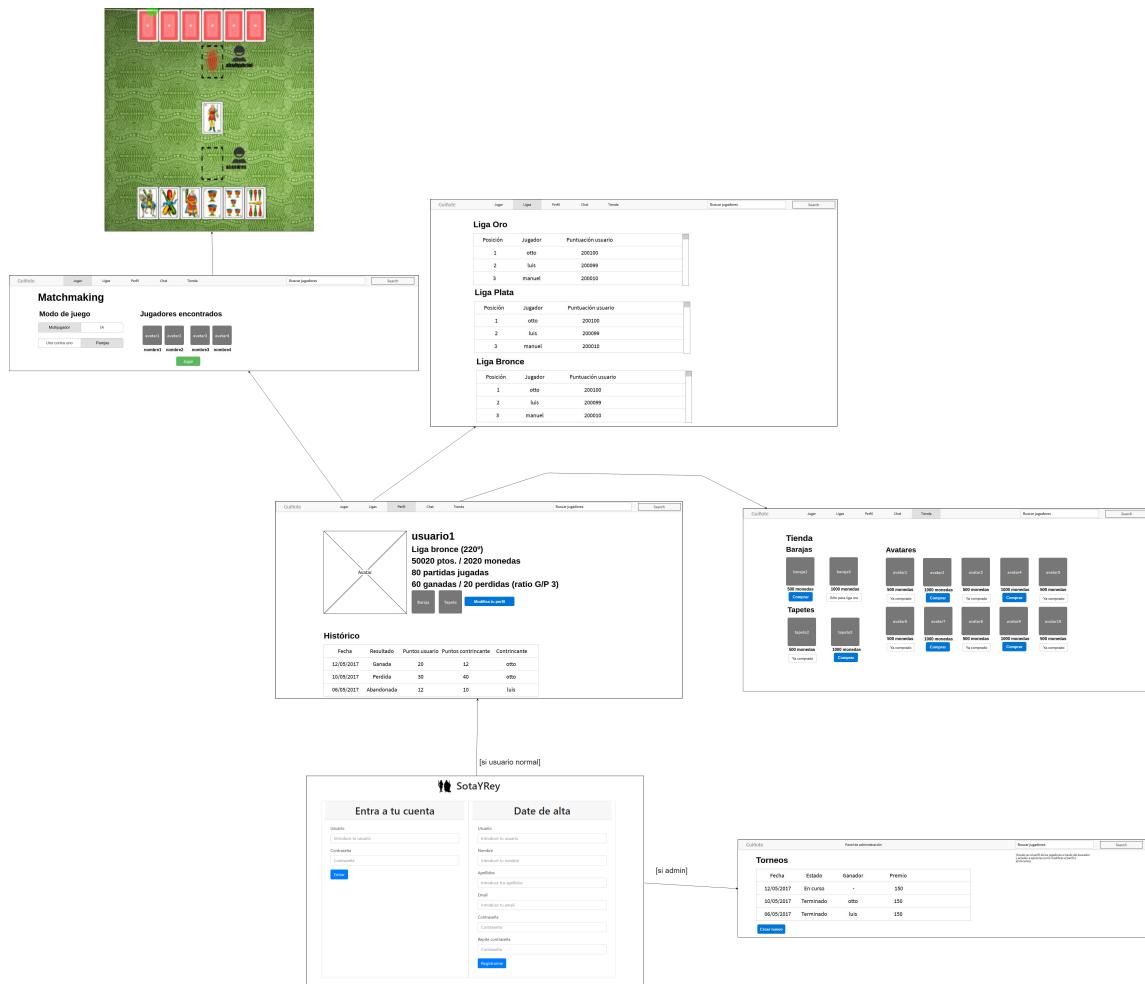


Figura 5: Mapa de navegación

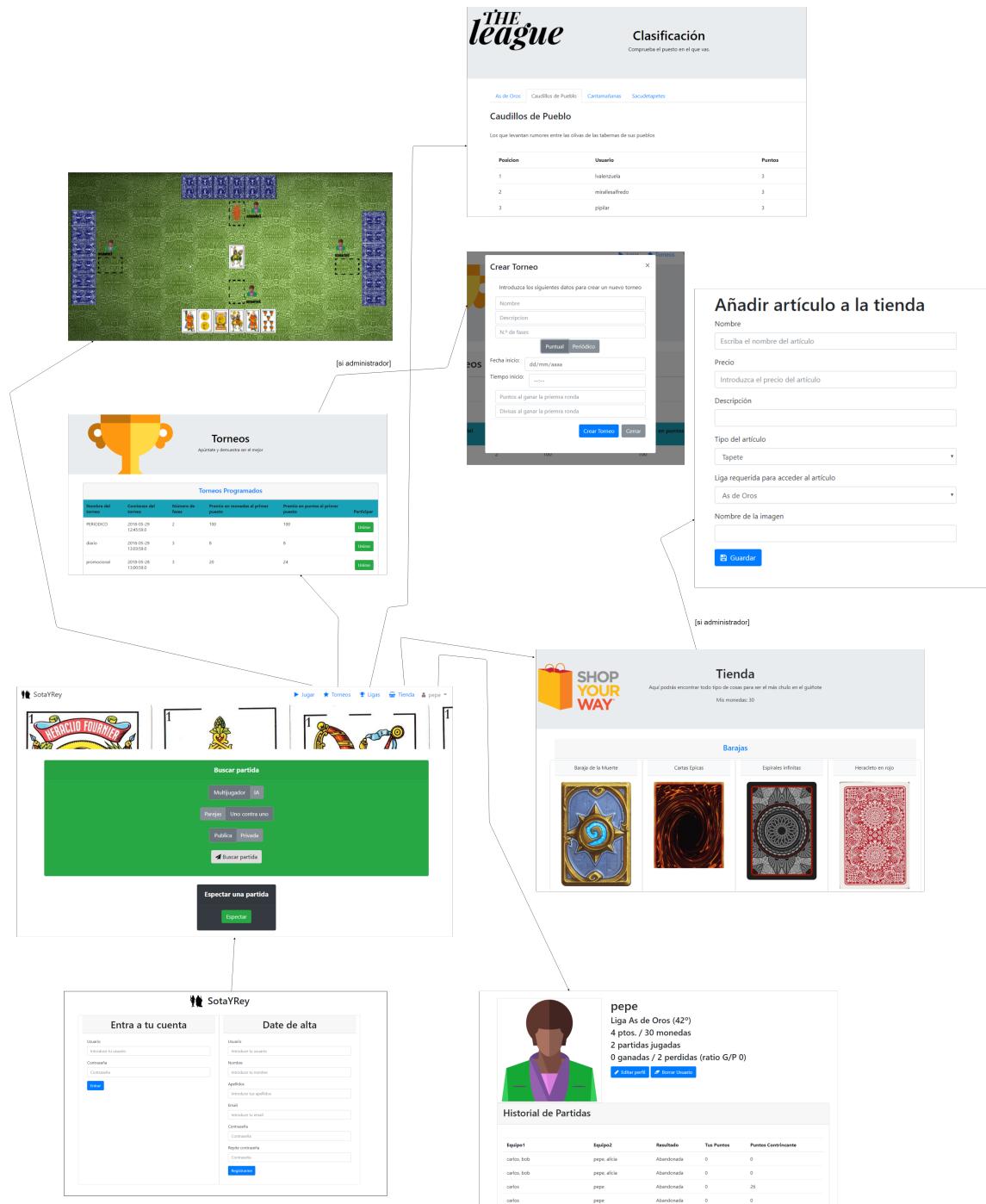


Figura 6: Mapa de navegación final

4.2.2. Gestor de mensajes (Netcode)

Para poder sincronizar los eventos que suceden en una partida entre el servidor y todos los clientes se usan websockets, siendo el gestor de mensajes un endpoint de websockets seguros en la URL "wss://[raíz]/gm/endpoint". El ciclo de vida de una partida comienza recibiendo el gestor los mensajes de `listoJugador`. Una vez se reciben los de todos, se les envía un estado inicial de la partida con toda la información necesaria. Después, cada jugador informa de los eventos sucedidos en su cliente al gestor, el cual se encarga de verificar su coherencia y retransmitirlo al resto de clientes. Al informar de los eventos de forma incremental, se evita el overhead de tener que enviar el estado completo cada vez. El único momento en el que es imposible evitar el mandar el estado completo es en la transición de idas a vueltas.

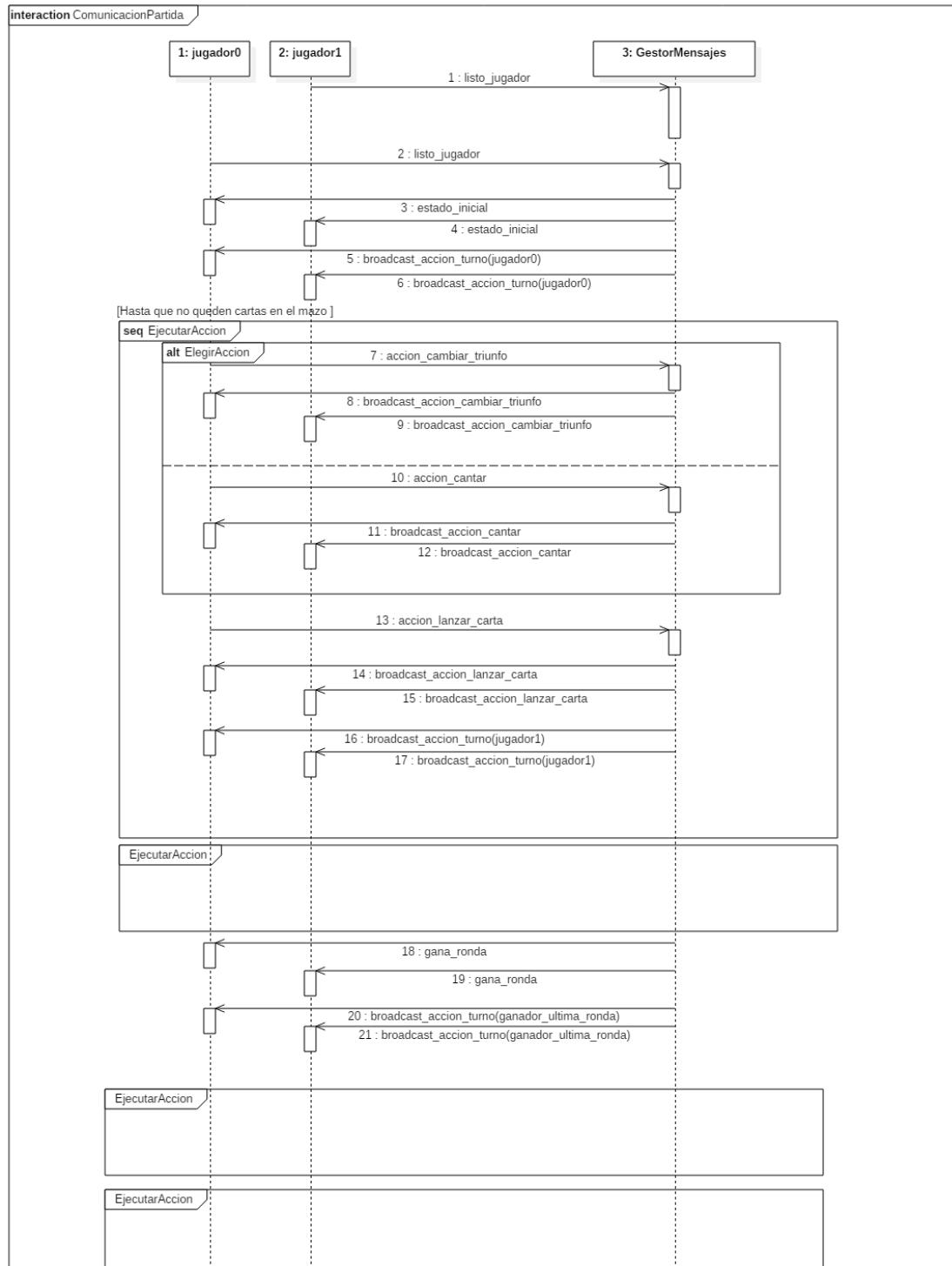


Figura 7: Diagrama de secuencia de una partida completa

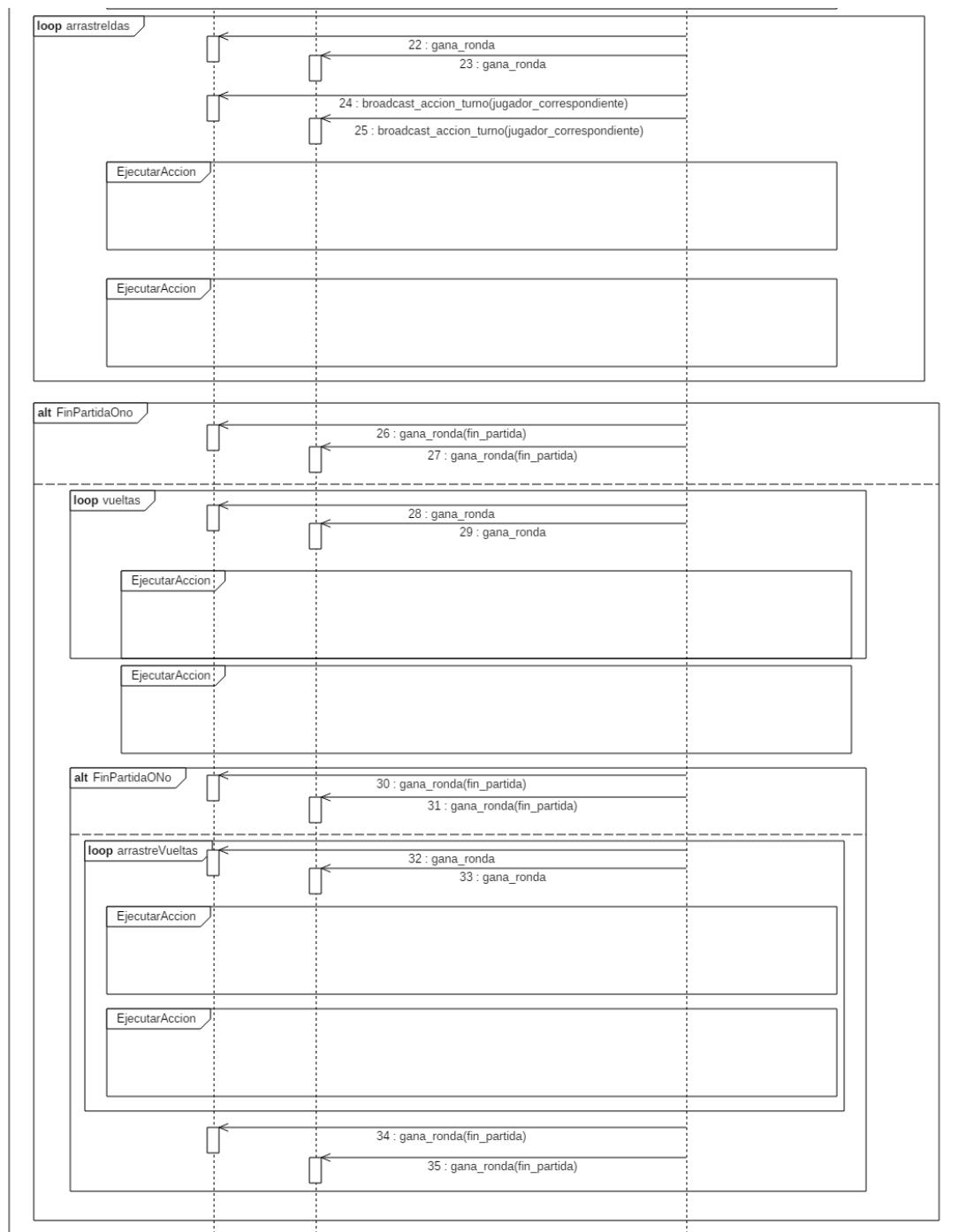


Figura 8: Diagrama de secuencia de una partida completa

Como caso interesantes se encuentra la desconexión/reconexión de jugadores, que se gestiona notificando al resto de jugadores de un timeout para programar una .“alarma”. Si antes de recibir el gestor la .“alarma” de cualquiera de los otros jugadores, recibe un mensaje `listoJugador` del ausente, se entiende que se reconecta y se le envía el estado de la partida en ese momento, notificando también al resto de jugadores. La otra posibilidad es que se reciba el timeout antes de que se reconecte, lo que significa que el jugador ha expirado su tiempo, finalizando la partida y notificando a los jugadores.

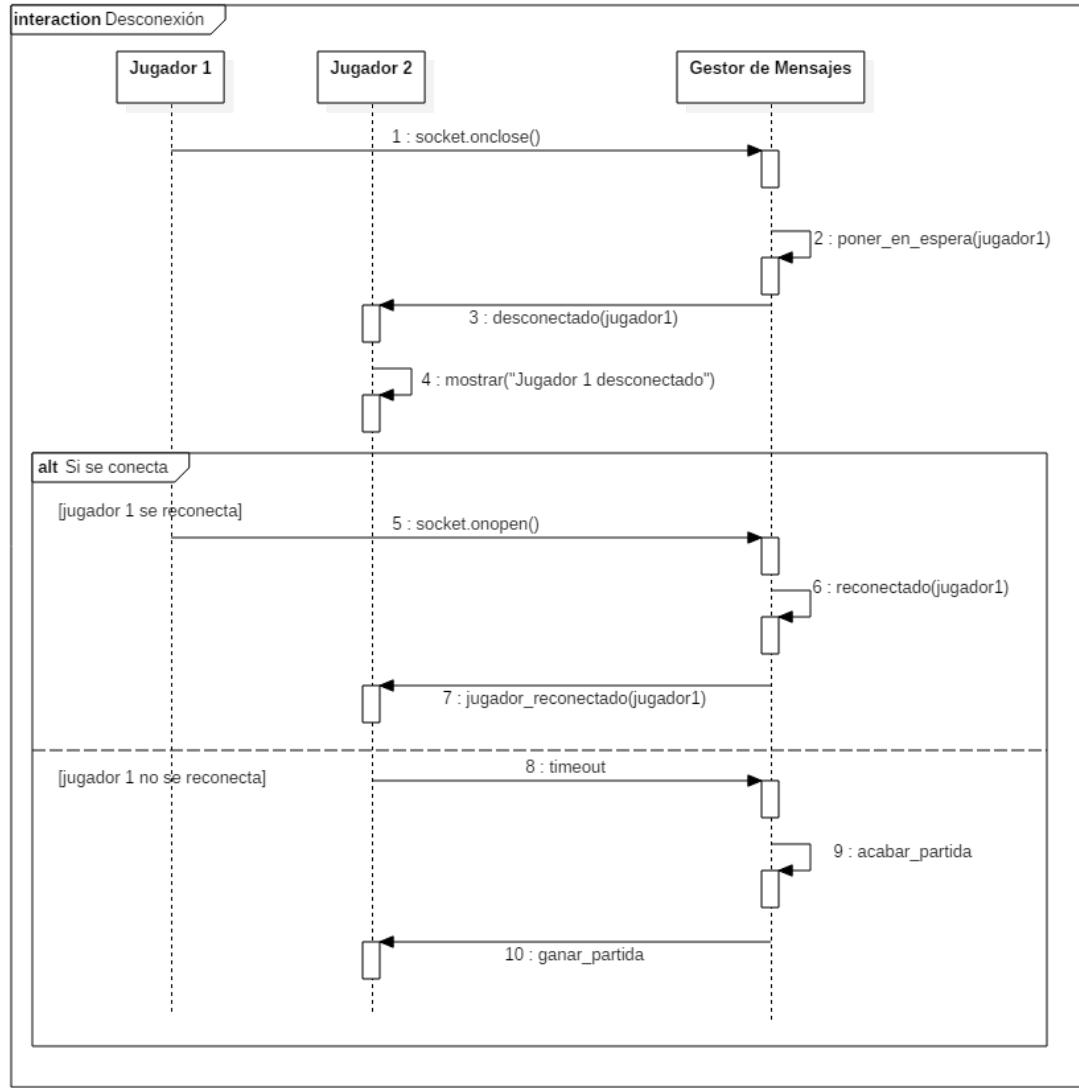


Figura 9: Caso de desconexión/reconexión de jugador

El otro caso interesante es la gestión de los espectadores, que básicamente son tratados como jugadores en el sentido de que son notificados de todos los eventos acontecidos pero no pueden emitir ningún evento. Además, su reconexión tampoco es gestionada ya que no influye para el desarrollo de la partida.

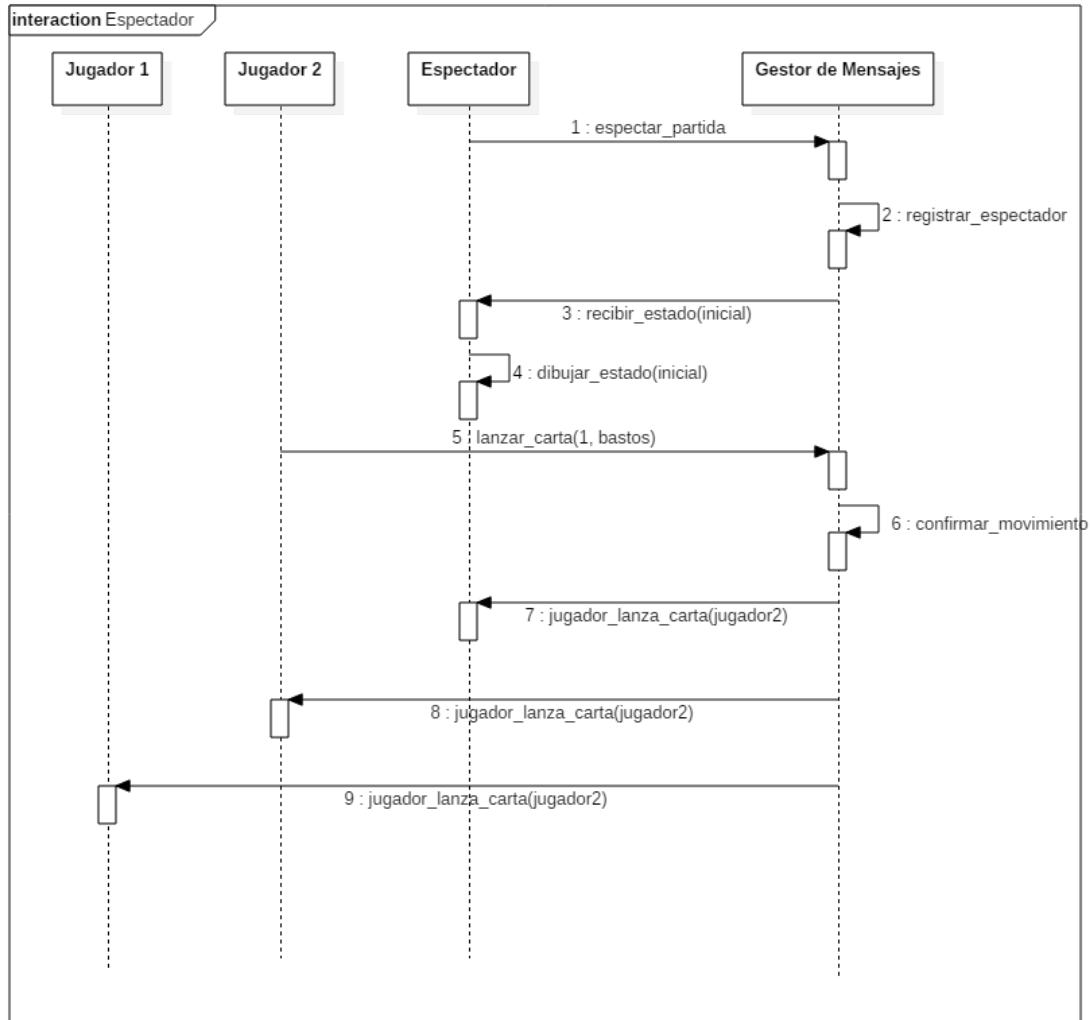


Figura 10: Gestión de espectadores

4.2.3. Emparejamiento (Matchmaking)

Una de las funcionalidades principales de la aplicación consiste en permitir que los jugadores encuentren a otros de su mismo nivel para jugar automáticamente. Esta funcionalidad la gestiona el endpoint de websockets seguros "wss://[raíz]/mm/matchmaking". La secuencia de búsqueda de partida es relativamente sencilla: El jugador que busca partida envía un mensaje con los parámetros de la partida deseada. En caso de que sea con la IA, se crea una partida nueva y se le redirige inmediatamente. En el resto de casos, si hay suficientes jugadores buscando una partida de las mismas características y pertenecen a la misma liga, se les empareja y son redirigidos a su partida. Además,

el jugador envía periódicamente mensajes informando de que sigue buscando, de forma que si excede un número determinado de "sigo buscando", se le empareja con un jugador de cualquier liga ya que se considera que ha esperado demasiado tiempo y jugar con alguien de diferente nivel tiene un impacto menos negativo sobre la experiencia de usuario que estar esperando largos períodos de tiempo.

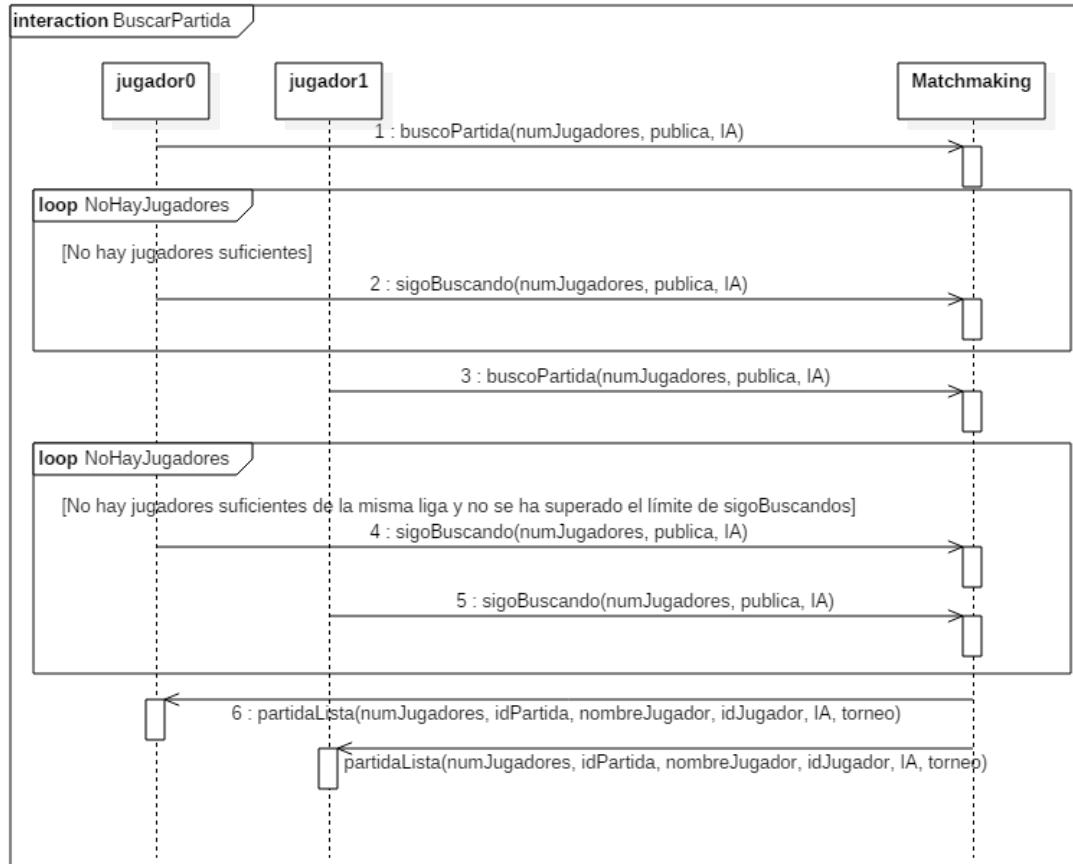


Figura 11: Búsqueda de partida

Además, este endpoint se encarga de gestionar también los torneos, de forma que cuando un jugador busca torneo envía un mensaje `buscoTorneo` con el identificador del torneo. El endpoint le contesta con el tiempo restante hasta la hora límite de comienzo del torneo. En cuanto haya un número suficiente de jugadores, se les empareja en partidas y se les redirige. Sin embargo, si llegada la hora límite no está lleno el torneo, los jugadores esperando envían un mensaje `empezarTorneo` al endpoint, que rellena los huecos del torneo con IAs.

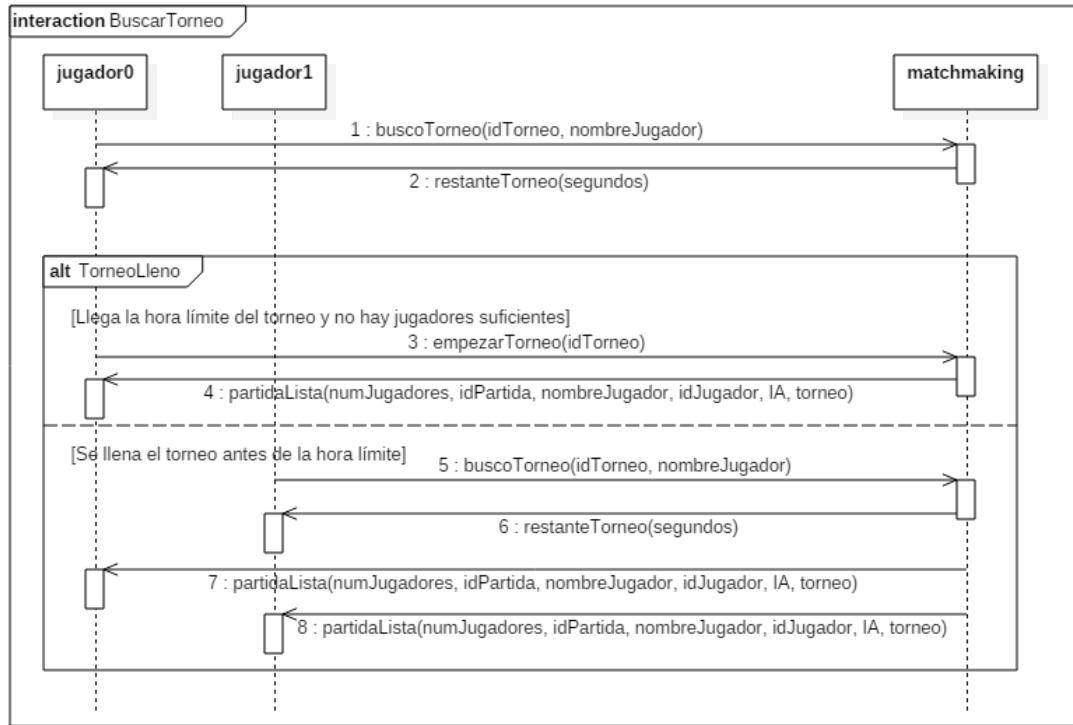


Figura 12: Búsqueda de torneos

4.2.4. Lógica y BackEnd

Para la implementación de la lógica del juego se ha analizado el dominio del problema y se ha realizado un primer diseño de clases de análisis. Posteriormente se ha implementado objeto a objeto el diseño inicial y al mismo tiempo actualizado el diagrama de clases.

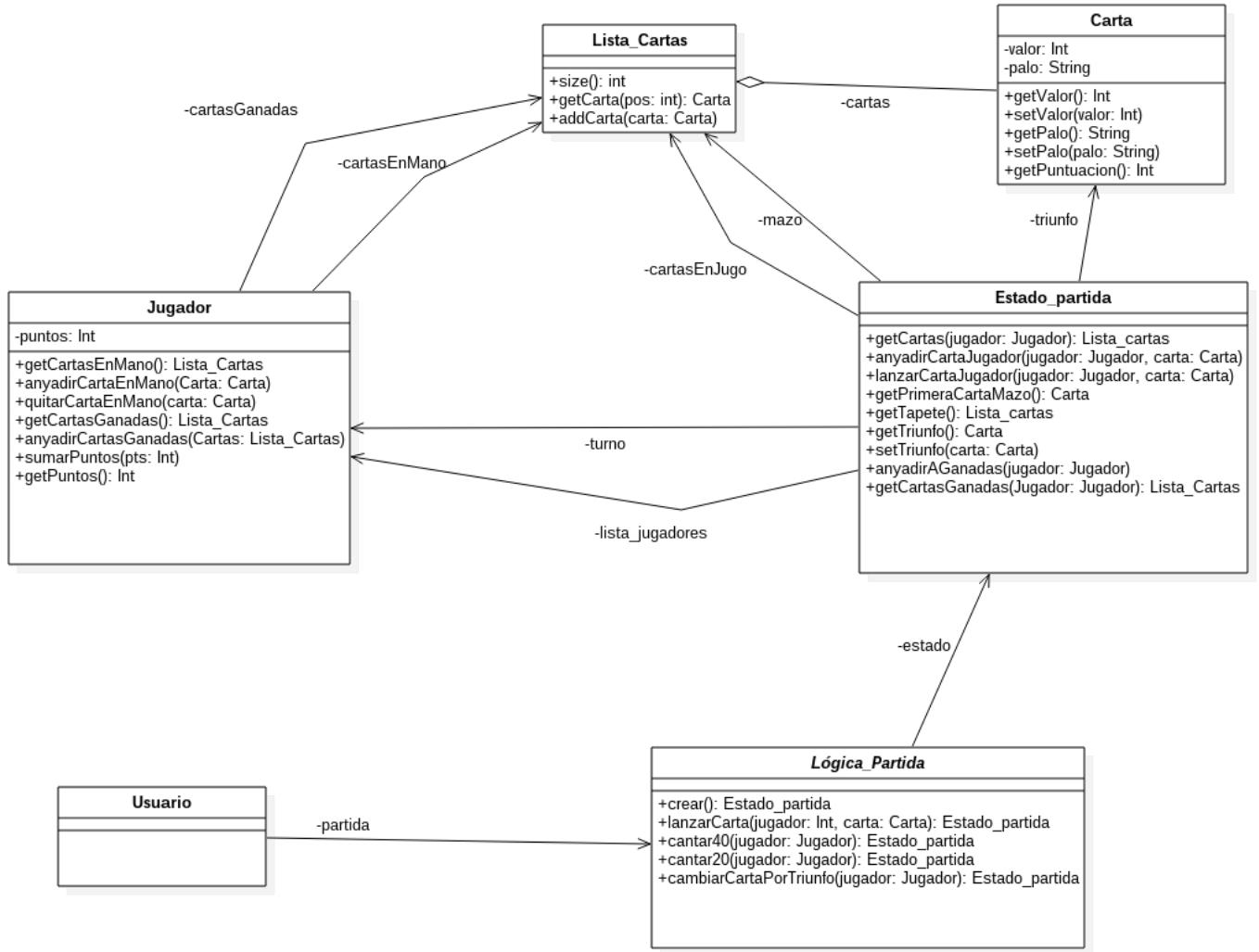


Figura 13: Diagrama de clases en la fase de análisis del problema

4.2.5. Base de Datos

El primer paso en el diseño de la base de datos fue plantear el esquema entidad relación como esquema conceptual del problema. Una vez diseñado el paso a un esquema relacional fue sencillo.

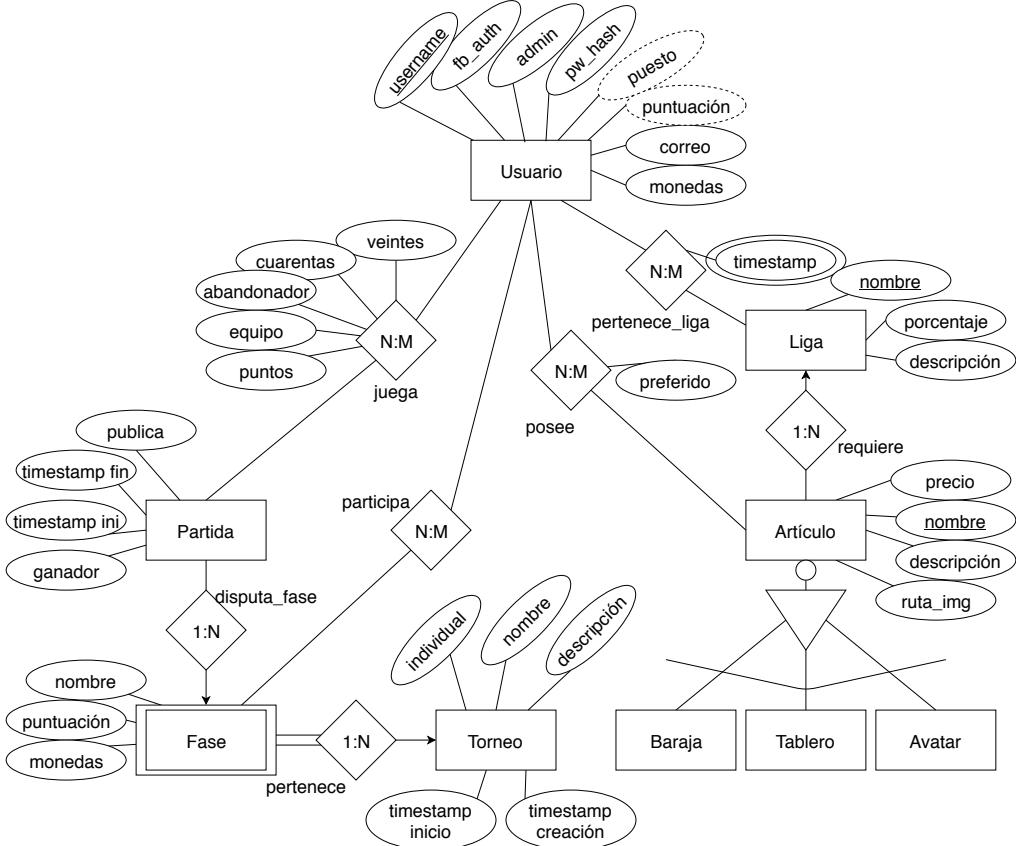


Figura 14: Esquema Conceptual

Antes de llegar al esquema actual se plantearon otras posibilidades, primeramente por una falta de claridad entre los datos que debían ser persistentes y los que no. En un primer momento se creó una entidad Espera, débil respecto a un Usuario que representaba que un usuario estaba esperando para encontrar jugadores. Finalmente la entidad no existe y la información de una espera no es almacenada en la base de datos. Las partidas son introducidas en la base cuando se empieza una partida, para así poder tener la información de las partidas en curso. La parte con más dificultad es la relacionada con los torneos. Para representar los torneos existen las fases, entendidas como octavos, cuartos, semifinales... Una partida puede estar ligada a una fase, y la fase pertenece a un torneo. Finalmente los jugadores están relacionados con una fase para poder emparejar jugadores.

Para la comunicación del sistema con la base de datos se ha utilizado el patrón DAO (Objeto de Acceso a Datos). Para ello se han implementado una serie de objetos VO que representan la información guardada de forma persistente y unos objetos DAO que abstraen las operaciones con el JDBC mediante métodos de Java. Para mayor seguridad de los datos, todos los posibles usos de la base se realizan a través de una interfaz que proporciona los métodos necesarios además de ofrecer un *pool* de conexiones para aumentar la velocidad de la interacción cuando haya múltiples usuarios.

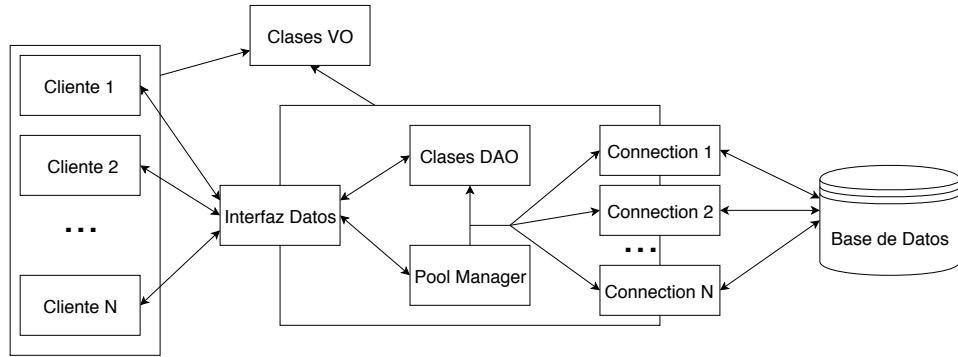
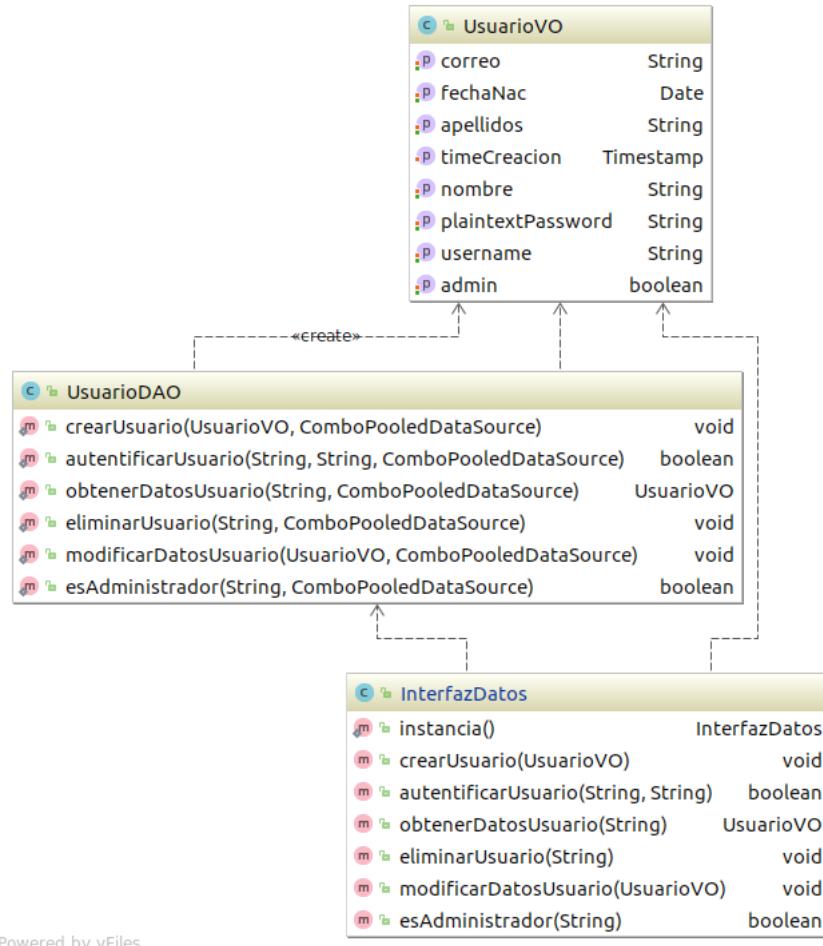


Figura 15: Esquema de la estructura del acceso a datos

Los clientes que usen la base de datos deben realizar todas las operaciones a través de la Interfaz de Datos. La Interfaz utiliza el pool manager para no tener que crear una nueva conexión con cada cliente y poder reutilizarlas. La Interfaz invocará a las clases DAO que utilizarán las conexiones para conectarse finalmente con la base. Las clases VO sirven para representar los objetos de la base en la comunicación entre las clases.

Dado el gran número de clases empleado en la implementación del modelo, su representación se ha realizado mediante diferentes diagramas de clases enfocados en las diferentes funcionalidades de la base .



Powered by yFiles

Figura 16: Diagrama de clases para la funcionalidad de usuario

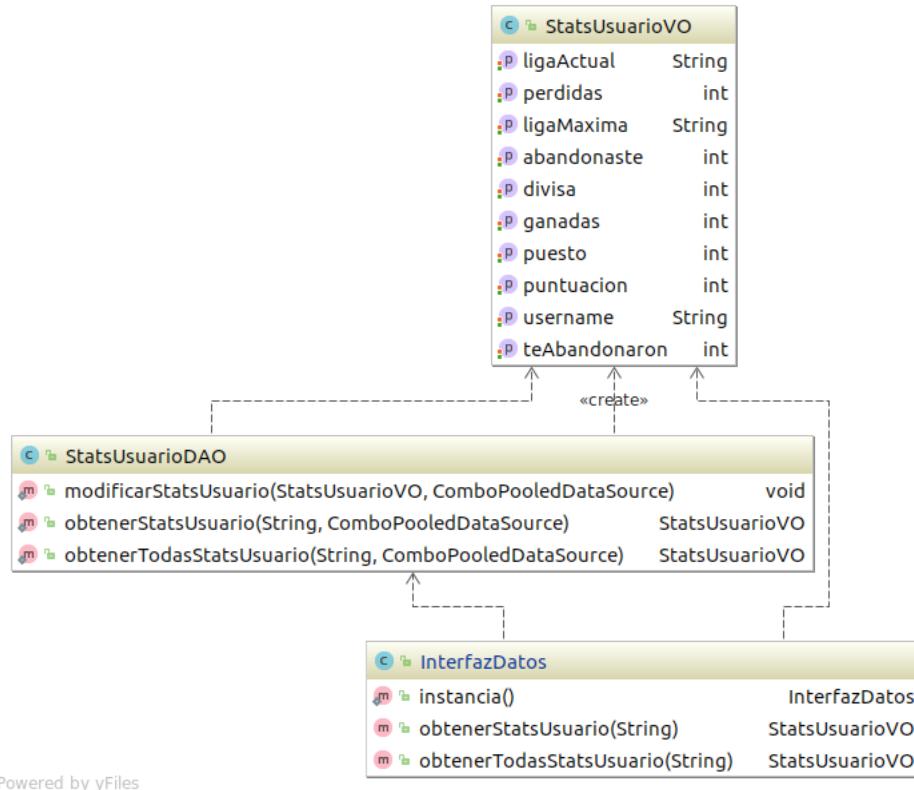


Figura 17: Diagrama de clases para la funcionalidad de las estadísticas del usuario



Powered by yFiles

Figura 18: Diagrama de clases para la funcionalidad de partida

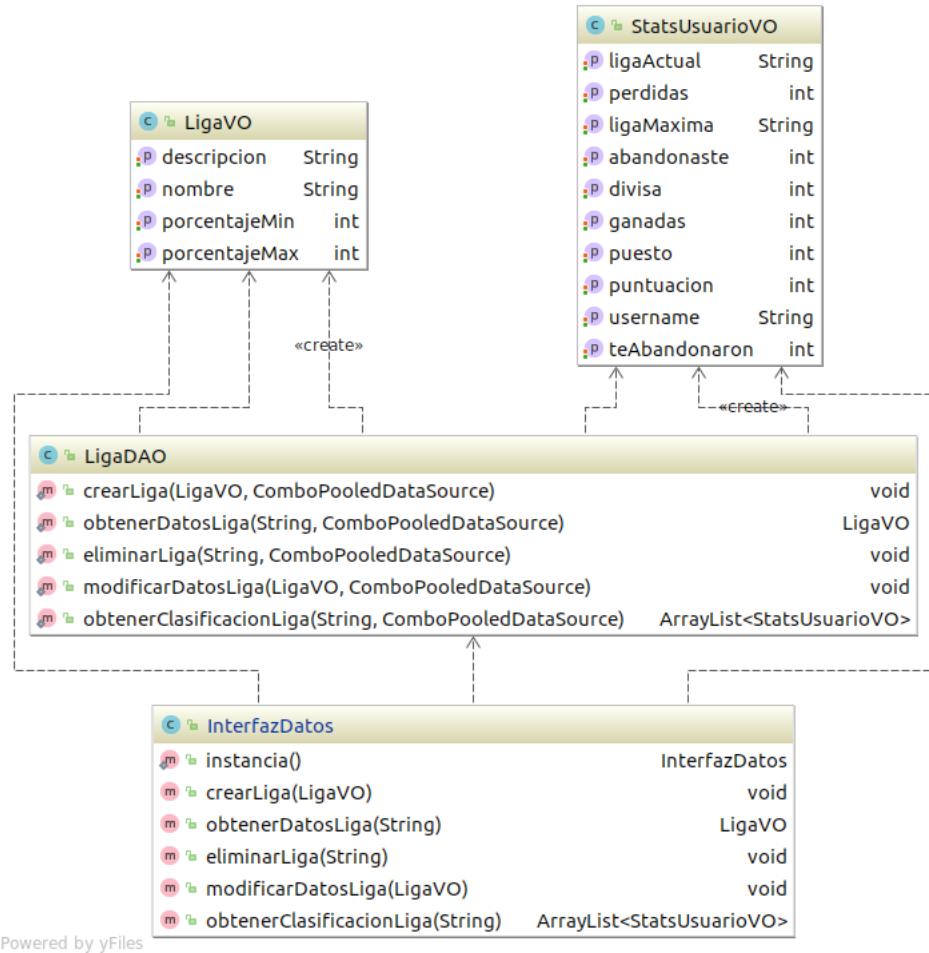
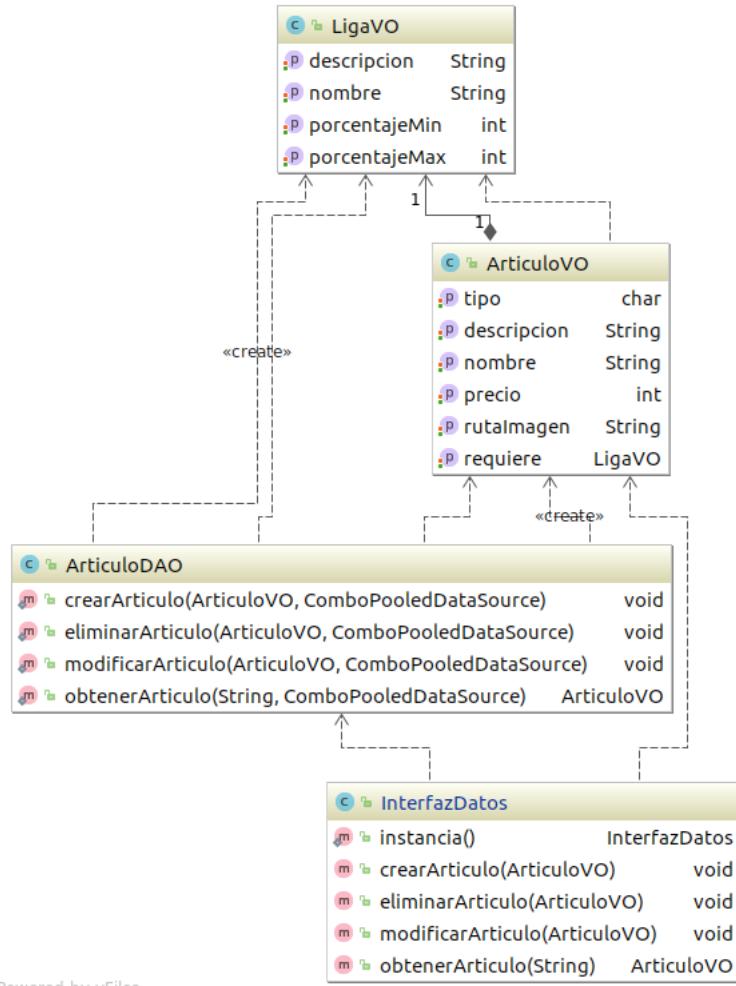


Figura 19: Diagrama de clases para la funcionalidad de liga



Powered by yFiles

Figura 20: Diagrama de clases para la funcionalidad de artículo

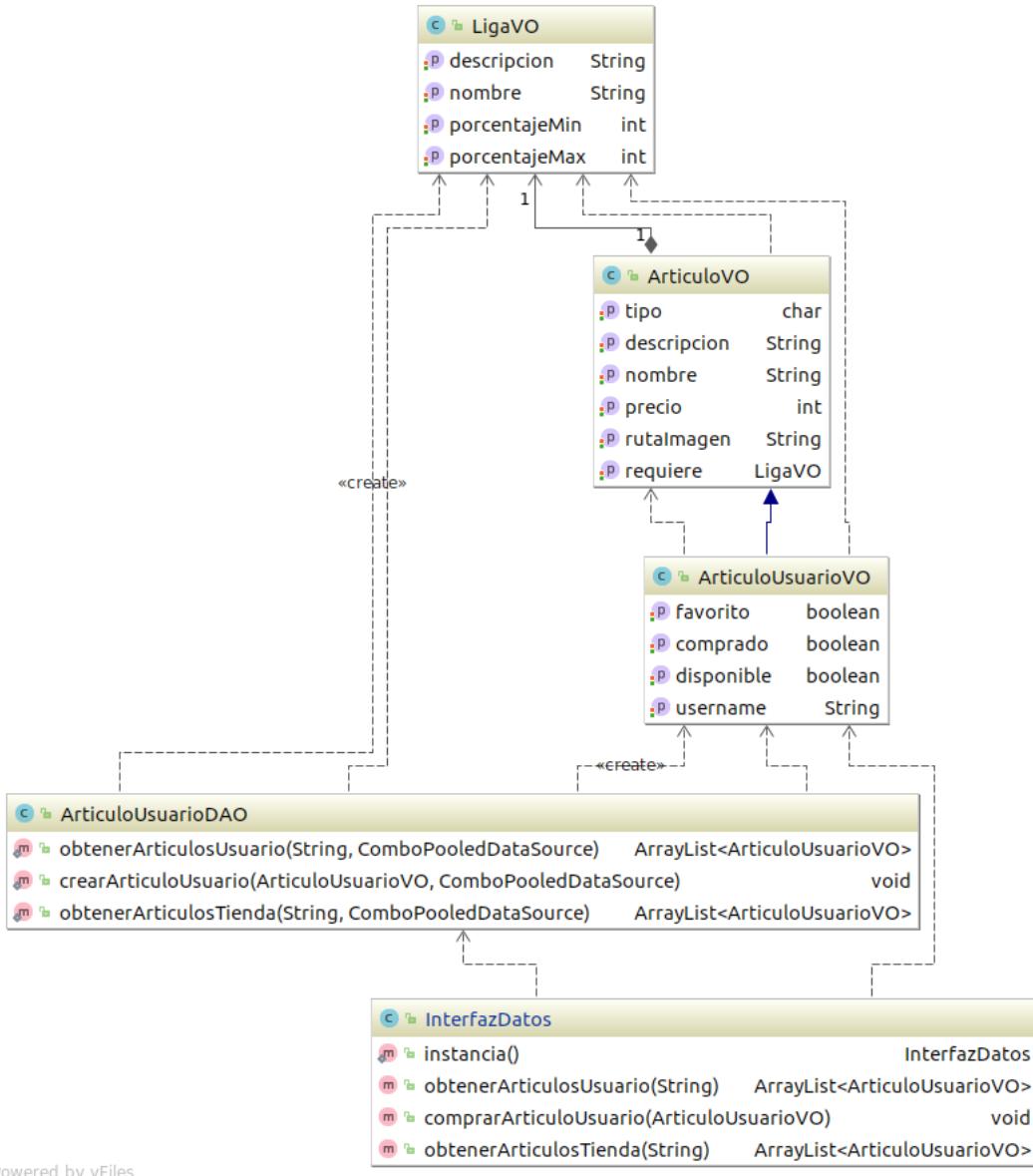


Figura 21: Diagrama de clases para la funcionalidad de los artículos de usuario

4.2.6. Inteligencia Artificial

Crear una inteligencia artificial para el Guiñote dista mucho de ser una tarea sencilla pues la propia lógica del juego hace que sea difícil decidir que cartas son beneficiosas para el jugador a tirar. Las personas cuando juegan aplican unas estrategias personales que pueden variar desde guardar

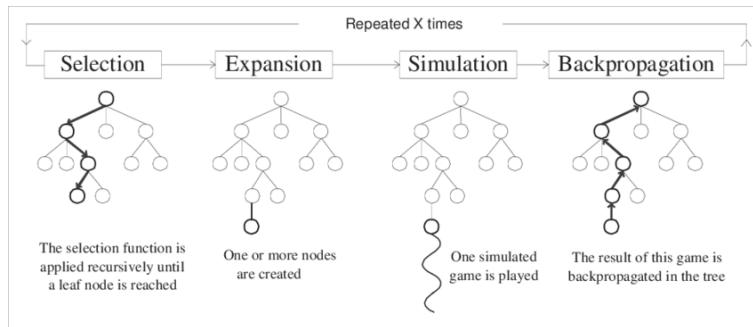
triunfos para el final, maximizar los puntos en cada ronda, descartarse de palos para el arrastre... y ninguna de estas estrategias es superior al resto ni garantiza ganar. Basar la IA en los mecanismos humanos no es una buena estrategia debido a que la forma de jugar de las personas está condicionada por prejuicios que han ido adquiriendo a lo largo del tiempo tras numerosas partidas por los pueblos de Aragón. Además, al tratarse de un juego poco extendido sus estrategias nunca han trascendido a niveles académicos dónde hayan sido estudiadas de forma objetiva. Por ello se ha decidido que la inteligencia a desarrollar debía ser independiente de lo que a priori puedan parecer buenas jugadas. Todo esto quiere decir que el sistema en ningún momento se basa en reglas escritas manualmente por los desarrolladores.

Para elegir el algoritmo más adecuado se ha consultado numerosa literatura buscando juegos similares al Guiñote. Se trata de un juego con información oculta y además no determinista, es decir, no se conoce la carta del rival, y no se sabe el orden de las cartas a robar, que podría ser cualquiera, dando lugar a distintos finales de la partida.

El primer algoritmo que se ha tenido en cuenta es el *expectiminimax*. Se ha descartado debido a que basa sus resultados en función de la probabilidad de cada carta, pero a ojos de un jugador del Guiñote, el rival puede tener cualquier carta, y cualquier carta puede ser la siguiente en ser robada.

El algoritmo final ha sido una variante del *monte carlo tree search*, conocido como *MCTS*. Este algoritmo es ampliamente utilizado para juegos de tablero donde el estado completo de la partida es conocido en cada momento. Basa su éxito en la realización de simulaciones desde un punto concreto de la partida. De esta forma elige el siguiente movimiento comprobando que jugada ofrece unos resultados mas favorecedores en las simulaciones. El problema de este algoritmo es que es necesario conocer el estado completo de la partida para poder utilizarlo (saber las cartas del rival y las del mazo), y hacer trampas no es una opción viable para una inteligencia artificial.

La forma de solucionar la información oculta viene con una variación del algoritmo original, utilizando el *information set monte carlo tree search*, conocido como *ISMCTS* [6]. La variación que incluye este algoritmo se basa en determinizar el estado de la partida. Conseguir el estado de la partida completa, pero debido a que es desconocido, se inventa. En cada iteración del algoritmo se 'barajan' las cartas desconocidas para la IA y se reparten al rival y al mazo de forma que en ese momento se tiene un estado conocido (aunque no tiene por qué ser real). El éxito del *ISMCTS* recae en que al realizar un gran número de simulaciones, la determinización aleatoria del estado de la partida tiende a representar el estado real.

Figura 22: Fases del algoritmo *MCTS* [5]

El esquema del algoritmo *MCTS* se basa en cuatro fases, cada nodo del árbol representa un movimiento, y cada nivel representa el jugador que realiza ese movimiento:

Selección: Se va bajando por los nodos del árbol hasta llegar a una hoja eligiendo el siguiente nodo con los resultados de la fórmula UCT basada en UCB1. Esta fórmula elige nodos que ofrecen un gran beneficio o que han sido poco explorados. El factor con el que realiza la exploración puede ser modificado a favor de mas exploración o más explotación.

Expansión: Cuando se llega a un nodo hoja se generan todos sus posibles hijos, es decir, las respuestas que da el otro jugador al movimiento de esa hoja.

Simulación: Desde el nodo hoja al que se ha llegado se realiza una simulación en la que se van realizando movimientos aleatorios (pero posibles) hasta finalizar la partida.

Propagación: Cuando la simulación ha finalizado se obtiene el resultado de la partida (ganada o perdida) y se va propagando esa información hacia arriba hasta la raíz.

Este proceso se repite multiples veces de forma que las simulaciones modelan el resultado real. Cuando se acaba el tiempo para realizar las iteraciones se elige como movimiento a realizar el hijo con más visitas de la raíz del árbol.

En el *ISMCTS* implementado se añade un primer paso previo a las cuatro fases. En ese paso se determiniza el estado de la partida. Específicamente en el Guiñote se mezclan las cartas desconocidas por la IA y se reparten de forma que ahora se conocen aunque no representan la realidad.

La IA realiza las simulaciones de forma aleatoria pero siguiendo las reglas del juego, de forma que para la implementación sólo ha sido necesario conocer las reglas del Guiñote pero no ninguna estrategia de victoria. El esqueleto de la implementación se ha basado en una implementación realizada en *Python* por quienes publicaron el artículo sobre el *ISMCTS* [7].

En primer lugar se ha realizado una implementación en *Python* para evaluar y validar que el algoritmo juega correctamente. Una vez comprobada la viabilidad se pasó a realizar la implementación definitiva en *Java* que sería conectada con el resto de componentes del sistema. Con la versión definitiva se realizaron una serie de pruebas a fin de determinar el mejor número de iteraciones que no suponga una espera al usuario y el mejor factor de explotación y exploración. A fin poder comparar la IA se le han presentado cuatro distintos rivales:

random: Juega una carta al azar.

greedy: Jugador voraz que siempre que tira en segundo lugar intenta maximizar los puntos obtenidos (con una penalización a gastar triunfo) y cuando lanza primera minimiza los puntos. Con esta estrategia sencilla se consigue una jugada óptima local que puede representar las jugadas de un rival novato.

cheating: Se trata de la misma IA, pero en vez de determinizar aleatoriamente la partida, 'mira' las cartas del rival y del mazo, de forma que hace trampas.

humano: Ignacio Bitrián, integrante del grupo, es un gran jugador del juego y ha participado en numerosos torneos.

El número de iteraciones elegido finalmente ha sido 20.000. De esta forma la IA responde en menos de un segundo pero son suficientes para que suponga una notable dificultad ganarle. Para elegir el factor entre explotación y exploración se han comparado distintos factores contra los distintos rivales, obteniendo que el mejor factor es 0.7, coincidiendo con el factor que aparece en algoritmo original. Con los parámetros ajustados se han enfrentado los rivales a partidas entre ellos obteniendo la siguiente matriz de resultado:

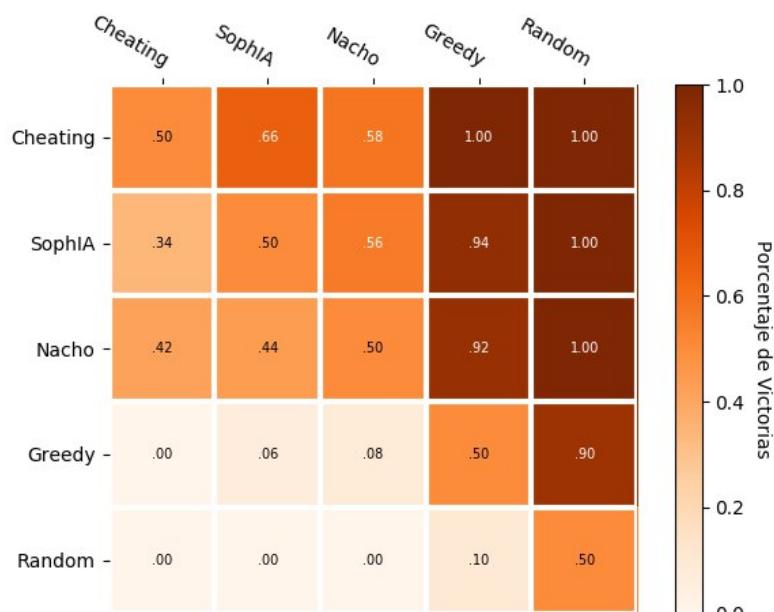


Figura 23: Matriz de resultados del algoritmo

Observando la matriz de resultados se puede concluir que la IA es muy competente, consiguiendo más victorias que derrotas frente a un rival humano experimentado. La IA que hace trampas gana en bastantes ocasiones a la IA normal, pero la IA ha sabido defenderse en el 30 % de las partidas. Jugando contra el jugador voraz y el aleatorio consigue victoria en casi todas las ocasiones.

4.2.7. Despliegue

La aplicación se pone en marcha en un servidor Tomcat, ya que permite la instalación de aplicaciones web en formato .war. Se distinguen dos aplicaciones diferentes: la que da soporte a la lógica de la aplicación y la encargada de coordinar una partida en curso. Ambas se comunican indirectamente a través de la base de datos y están coordinadas.

La base de datos es relacional ya que se necesitan muchas consultas de los datos almacenados y para cada partida hay varias inserciones o actualización de los datos almacenados. El sistema gestor de la base de datos va a ser MySQL porque se ahorran costes al ser un SGBD de código abierto y no tener que adquirir una licencia de pago. Además, el equipo cuentan con experiencia en el diseño e implementación de bases de datos utilizando MySQL.

La razón por la que no se ha elegido otro SGBD de código abierto son los problemas que presenta RDS Aurora. Otros SGBD como PostgreSQL son más exigentes en recursos y, por lo tanto, el coste es mayor sin repercutir un beneficio real sobre la aplicación ya que se considera que MySQL es más que suficiente para una aplicación de estas características. Otra alternativa era Oracle pero debido al alto coste de la licencia no se ha escogido.

El patrón de diseño utilizado para la comunicación del sistema con la base de datos es *façade* ya que permite dividir el sistema completo en dos o más subsistemas consiguiendo un alto desacoplamiento de la base de datos y el juego. De esta forma, se puede dividir mejor el trabajo de forma independiente entre los equipos para poder llevar a cabo un diseño e implementación top-down.

4.3. Tecnologías elegidas

- **Interfaz de la partida en el navegador web.** La interfaz del juego consiste en una única pantalla donde los jugadores que participan en la partida en curso se comunican. Se implementa utilizando JavaScript y Phaser. Phaser es un framework para desarrollo de juegos en HTML5 basado en la tecnología JavaScript.

Se ha decidido utilizar JavaScript por la sencillez a la hora de ser visualizado en un navegador y se incorpora a la perfección con HTML5. Se ha barajado la posibilidad de utilizar Flash pero se descarta por estar obsoleta y porque algunos navegadores ya no lo soportan. También se podría haber utilizado Unity pero no se ha llevado a cabo por tener una curva de aprendizaje muy complicada.

- **Capa de comunicación de la partida.** Es el servicio que está por debajo de la partida encargado de notificar las acciones de los jugadores al resto. Además comprueba que el transcurso de la partida es correcto, como si de un coordinador se tratara. Se utiliza lenguaje Java, ya que es un servicio Web e irá desplegado a través de un archivo .war. Además, facilita la comunicación con la tecnología WebSockets, que es la que se ha escogido para comunicar en tiempo real el navegador con el controlador. Se ha decidido WebSockets por tener una curva de aprendizaje sencilla ya que se utilizará para enviar mensajes desde el navegador al controlador. Se integra perfectamente con Java.

Se ha descartado utilizar Sockets.io ya que va ligado a Node.js, que es una tecnología más novedosa pero que el equipo desconoce por completo, aprenderla supone un número de horas extras y se desconoce si es una tecnología viable para la aplicación a desarrollar.

- **Interfaz Web.** Se implementa en HTML 5 y se utiliza JSP y Servlets para la generación de contenido dinámico y procesamiento de formularios, respectivamente. Se trata de una

tecnología poco actual pero de la cuál el equipo de desarrollo tiene cierta experiencia, por lo que se asegura la calidad del servicio.

- **Lógica y dominio de la aplicación.** Implementado en Java para favorecer la interoperabilidad con la interfaz web.
- **Acceso a los datos.** Se utilizan objetos de tipo implementados en Java. De esta manera se cumple el patrón "Modelo - Vista - Controlador".
- **Base de datos.** Se utiliza un Sistema Gestor de Bases de Datos relacional, ya que las principales consultas que se hacen son de tipo JOIN. Se ha decidido utilizar MySQL ya que es un sistema que el equipo domina. La desventaja es que es poco eficiente en comparación con otros como Oracle, pero para el número de usuarios que tendrá la aplicación es suficiente con dicho gestor.

5. Memoria del proyecto

A continuación se describen los objetivos principales de cada uno de los equipos de trabajo así como un pequeño resumen de cómo se ha desarrollado el trabajo de cada equipo a lo largo de todo el proyecto. En el resto de la sección se describe más detalladamente el desarrollo del proyecto.

FrontEnd y Middleware

El principal objetivo del Frontend eran desarrollar las interfaces con las que los usuarios interaccionarían con la aplicación. Se distinguen dos interfaces, las de navegación web y la parte jugable del guiñote. La parte jugable se ha podido implementar en su mayoría, a falta del modo espectador y la parte de personalización. Sin embargo, la parte de la interfaz de navegación se completará para la segunda iteración, ya que únicamente se tienen las pantallas de login y la navegación.

En cuanto al Middleware, el principal objetivo era comunicar la interfaz de juego para que los usuarios puedan jugar entre ellos, que se ha conseguido con éxito. Aun así, es necesario completar con la funcionalidad de que un usuario puede cambiar de dispositivo en una misma partida.

BackEnd

El objetivo del equipo de backend era desarrollar un paquete de java que representase la lógica del juego del guiñote, al mismo tiempo que desarrollaba gran parte de la página web dinámica del proyecto. En la primera parte del proyecto el equipo se centro en el análisis, diseño, implementación y pruebas de la lógica del guiñote. Esta parte era fundamental para el correcto avance del proyecto ya que el equipo del FrontEnd y la IA dependía de este módulo para desarrollar al completo sus tareas. Por este motivo, se hizo mucho énfasis en las pruebas de la lógica ya que una vez integrado con los demás componentes del sistema son muy difíciles de detectar los errores porque al tratarse de un juego de carta es muy difícil reporducir de nuevo el mismo contexto que ha producido el error. A mitad del proyecto, tras reducirse el equipo a dos miembros, se empieza a trabajar en la página web. Primero se tuvieron que diseñar algunos de los jsp de la web ya que el equipo de FrontEnd llevaba mucha carga de trabajo en la segunda iteración. Para ello, se dividió el trabajo en 2 partes, donde cada miembro del equipo se encargaba de realizar la mitad de los jsp de la web y los servlets correspondientes a dichas vistas.

Al finalizar la implementación de la web dinámica, el equipo se dedicó a realizar más pruebas en la lógica y a resolver los errores que el resto de equipos habían detectado en la lógica del juego.

Base de datos

Los principales objetivos del equipo de bases de datos eran por una parte diseñar e implementar la base de datos para el sistema (esquema conceptual, lógico y físico) y por otra desarrollar la capa de acceso a datos proporcionando una interfaz para el Backend. Ambos objetivos se cumplieron en su totalidad y según el plazo establecido, a excepción de los torneos, que debido a su complejidad (no estimada en un primer momento) se alargó su desarrollo varias semanas. Además, el equipo desarrolló los eventos periódicos de la base de datos: torneos periódicos y actualización de ligas. Exceptuando las pequeñas dificultades iniciales relacionadas con un tardío diseño de la interfaz de acceso a datos, tanto la comunicación entre los miembros del equipo como con el resto de equipos se ha desarrollado sin problemas. En la parte final de integración, se detectaron algunos errores en el

acceso a datos, aunque la depuración de esta parte de la aplicación fue una de las menos complejas y fueron solucionados todos los errores sin mayores problemas. El trabajo del equipo en general ha sido muy satisfactorio.

Inteligencia Artificial

El objetivo del equipo de inteligencia artificial ha sido cumplido con un claro éxito: desarrollar una IA competente para jugar al guiñote 1 vs. 1. El trabajo de este equipo ha consistido en primer lugar en investigar en la literatura para aprender sobre algoritmos de inteligencia artificial utilizados en juegos similares, la decisión del algoritmo a utilizar, el desarrollo de un prototipo funcional para estudiar su viabilidad, y finalmente su implementación final y la integración de este con el resto de la aplicación. El equipo que formaba la IA fue reconfigurado antes de empezar a trabajar añadiendo a los miembros del equipo dedicado a la base de datos dado que iban avanzados y tenían una menor carga de trabajo. El trabajo se ha desarrollado sin problemas destacables, a excepción de que la fase de análisis e investigación fue algo más costosa de lo planificado. Los resultados de SophIA han sido mucho mejores de lo esperado, suponiendo esta una de las características más distintivas de la aplicación final.

5.1. Inicio del proyecto

Tal y como se describió, tras un diseño preliminar del sistema en general, cada subequipo se puso a analizar sus necesidades concretas y a buscar ejemplos para comenzar a implementar con cierto contexto. Cabe destacar que entre las búsquedas de esos ejemplos, se estableció contacto con Iván López Asín, desarrollador de la app [guiñotepro.es](#). Fue una sorpresa que usase generalmente las mismas tecnologías que las propuestas preliminarmente (Phaser.io y WebView para móviles), lo que resalta un buen trabajo de documentación previo del equipo. Además, mostró su interés en integrar nuestra inteligencia artificial, pero habría que portarla a Javascript y adaptarla a su implementación.

Posteriormente, se configuró el repositorio en Github debidamente y como denominador común se ha utilizado la suite de IDEs de JetBrains: WebStorm para Javascript, IDEA para Java, JSP y Servlets y, por último, DataGrip para el diseño y la interacción con la base de datos.

Por último, antes de ponerse a implementar, cada equipo de trabajo debió pasar una fase de formación. Sin embargo, se ha intentado centrar la atención exclusivamente en las tecnologías que afectasen directamente a cada equipo para reducir el tiempo invertido en esto. Por suerte, el backend del proyecto está desarrollado en Java, tecnología en la que todo el mundo tiene cierta experiencia, haciendo así más fácil la interacción entre los diferentes componentes y la depuración de errores.

Respecto al aprendizaje de las nuevas tecnologías aplicadas en el proyecto, la experiencia ha sido generalmente positiva. En el caso de la GUI, se ha tenido que aprender a utilizar tanto Javascript como el framework de videojuegos Phaser. Para ello, se han utilizado tutoriales tanto oficiales como de terceros. Además, entre la GUI y el gestor de mensajes la comunicación se realiza a través de websockets, por lo que ha habido que formarse tanto en la perspectiva de websockets para Javascript como para JavaEE con la API de Tomcat. En el backend, los principales problemas de integración se han debido a falta de experiencia con el flujo de trabajo del IDE, lo que era un coste temporal que no estaba calculado a priori. Por último, según lo decidido, se configuró la base de datos en MySQL dentro de Amazon Aurora (AWS), tratándose de una tecnología también novedosa para el equipo.

Cabe destacar que el despliegue transcurrió sin mayores incidencias siguiendo las instrucciones provistas por Amazon. Las principales características de la base de datos utilizada para el proyecto son:

- Versión Software: Aurora MySQL 5.7.12
- Hardware: 1 CPU, 2GB RAM
- Identificador de la Intancia: sotayrey-aurora
- Identificador del Cluster: sotayrey-cluster
- Identificador de la base de datos: sotayrey_db
- Copia de Seguridad: Cada 30 días

Éstas se deben al requerimiento de cumplir con las limitaciones de la versión gratuita para estudiantes. Además, se ha habilitado el acceso externo para facilitar la fase de desarrollo, aunque se cerrará cuando se pase a producción. Respecto al software relacionado con la base de datos, hizo falta formación en el funcionamiento de la API JDBC, así como en la utilización de pools de conexiones con c3p0 para mejorar el rendimiento, cosa que no se había especificado anteriormente en los procesos de inicio del proyecto.

5.2. Ejecución y control del proyecto

Los procesos de control no se han llevado a cabo tal y como estipulaba el primer plan de gestión. Ya que el director del proyecto no se ha coordinado con los responsables de grupo. Sin embargo a nivel local, los responsables de grupo si han dirigido y organizado sus respectivos grupos. Además no se ha producido ninguna situación que requiera de mediación ni necesidad de intervención extraordinaria por parte de los responsables. Los procesos técnicos se han llevado a cabo tal y como estaban especificados en la primera versión del plan de gestión. Tanto en el apartado de pruebas como documentación. Aplicar los procesos acordados ha costado bastante por la falta de costumbre, sin embargo los ficheros fuente, pruebas y documentación obtenidos nos han permitido trabajar de forma más eficiente.

5.2.1. Reparto del trabajo

En un primer momento, el reparto de trabajo fue definido únicamente en relación a los equipos creados, como se detalla en la sección 3.2.4. Sin embargo, se detectó que este reparto, aunque necesario, era demasiado genérico y no permitía medir el progreso de forma clara. Por ello se decidió que dentro de cada equipo de trabajo se llevaría a cabo una división del trabajo en tareas o paquetes de trabajo, siguiendo la Estructura de Descomposición del Trabajo (WBS [8]). Cada paquete de trabajo corresponde a una tarea específica y claramente delimitada, como puede ser el desarrollo de una clase, y especifica un único responsable de este (no significa que sea la única persona que trabaje en él pero sí la responsable de su desarrollo). Para el control del progreso en relación a esta división del trabajo se ha utilizado el apartado Projects de GitHub, en el que cada paquete se marca con un tick cuando está finalizado y depurado.

A continuación se muestran los diagramas de paquetes desarrollados:

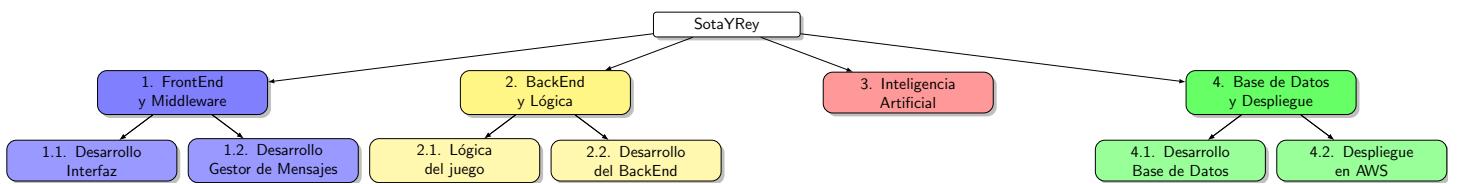


Figura 24: Diagrama general de reparto del trabajo

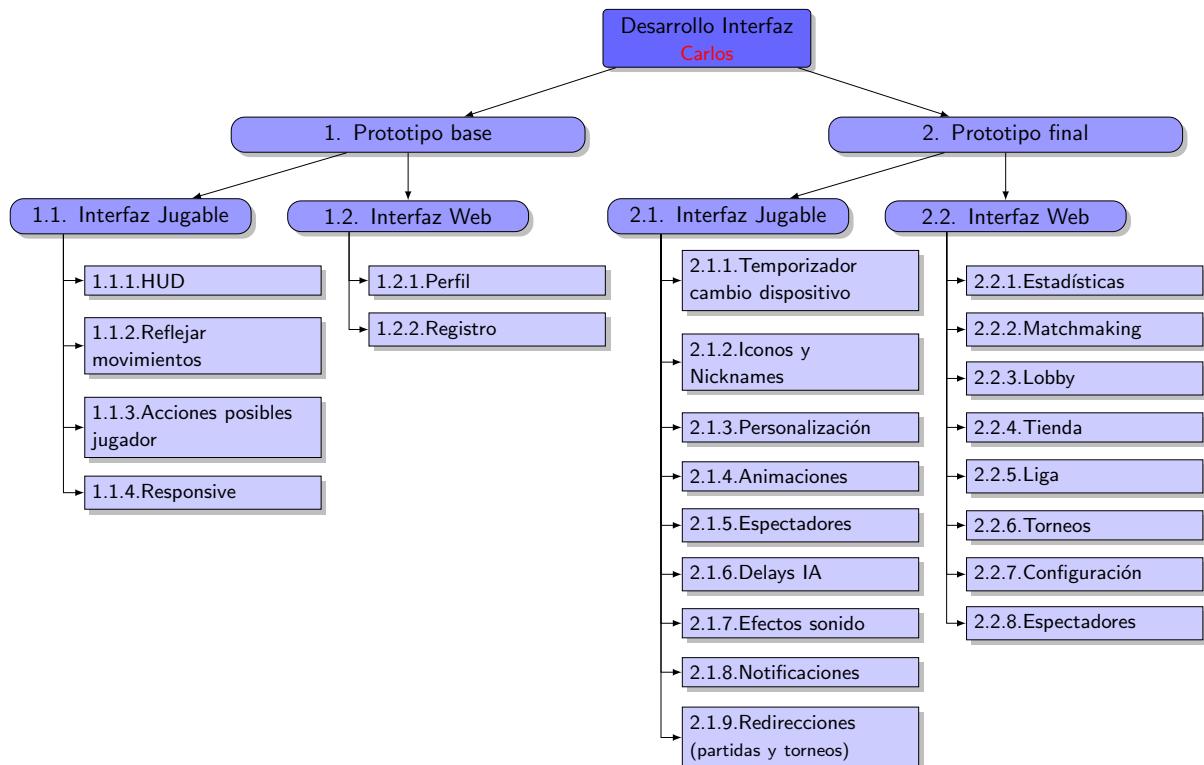


Figura 25: Diagrama de Paquetes de Trabajo Interfaz

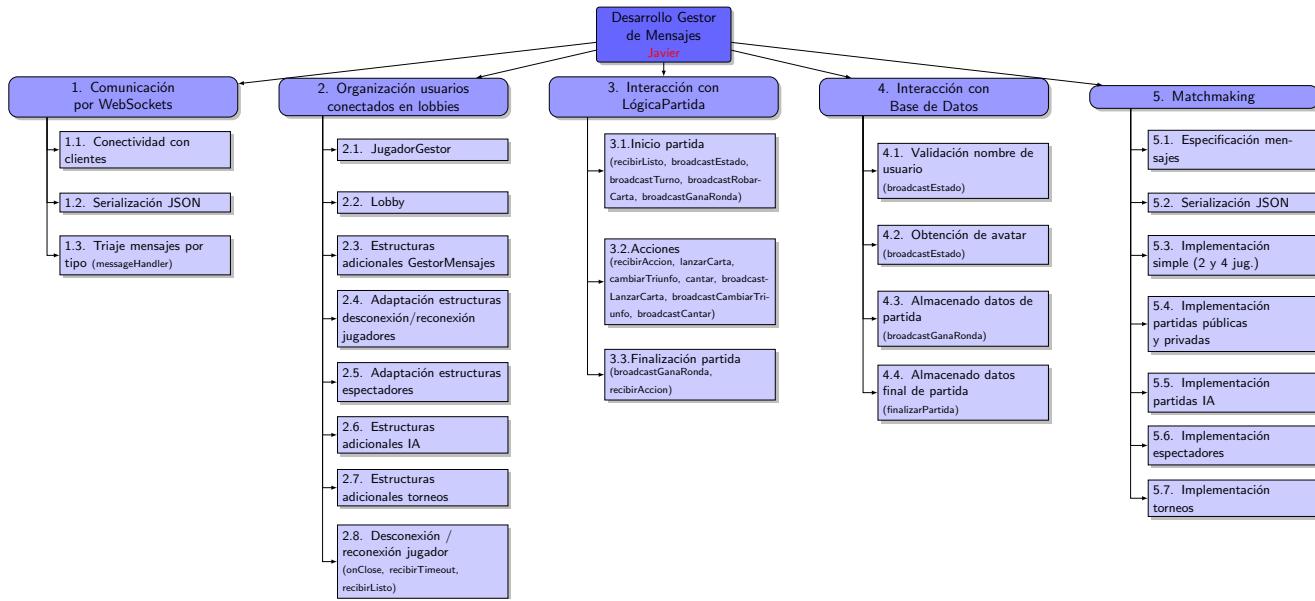


Figura 26: Diagrama de Paquetes de Trabajo Gestor de Mensajes

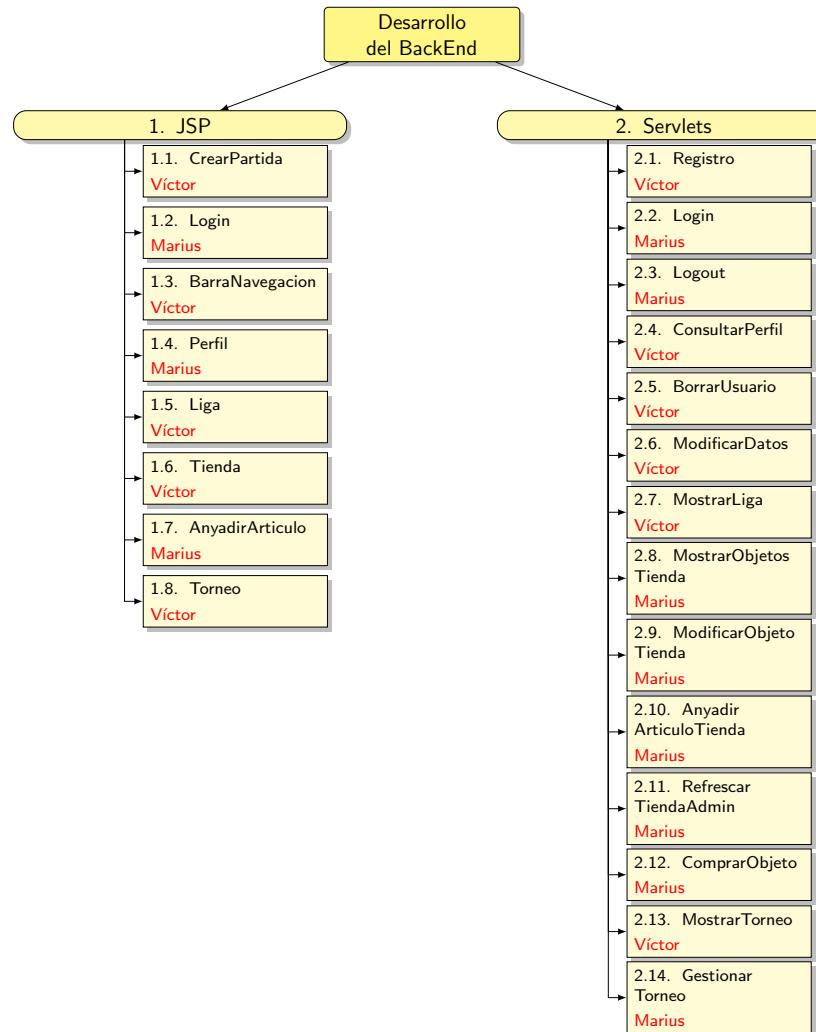


Figura 27: Diagrama de Paquetes de Trabajo BackEnd

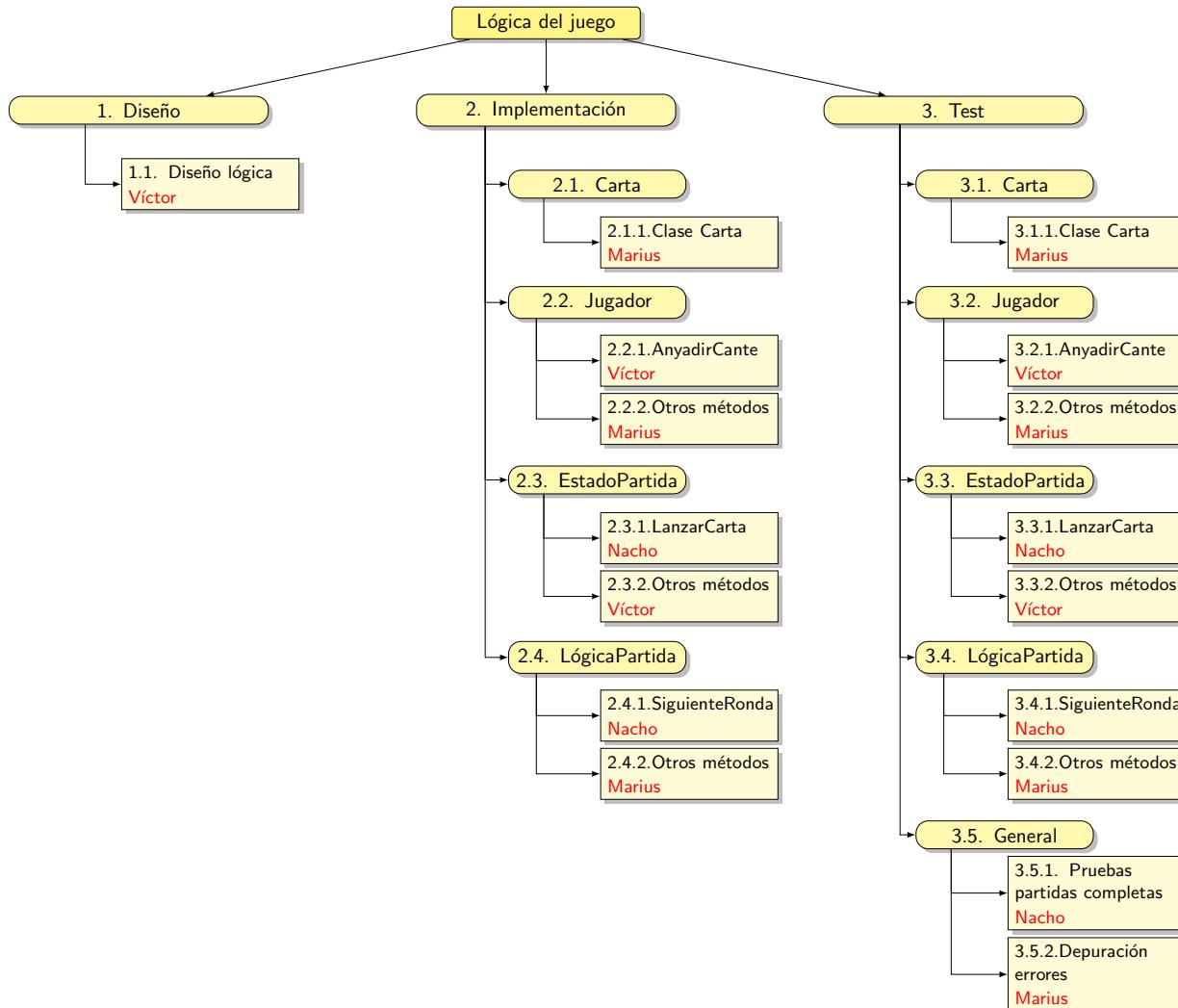


Figura 28: Diagrama de Paquetes de Trabajo Lógica

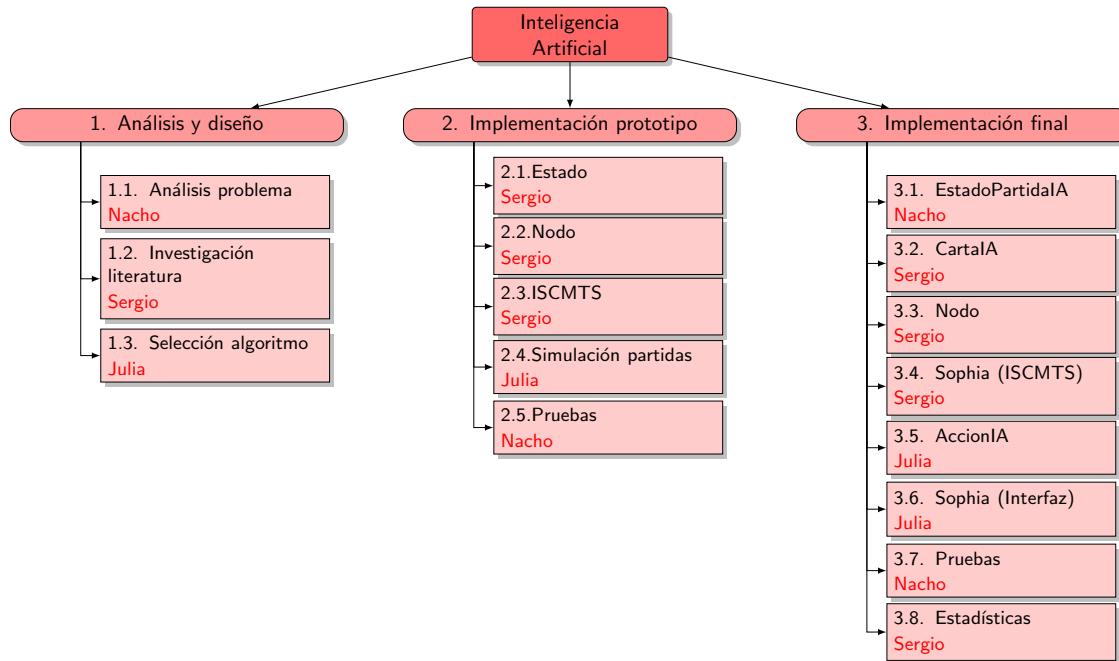


Figura 29: Diagrama de Paquetes de Trabajo Inteligencia Artificial

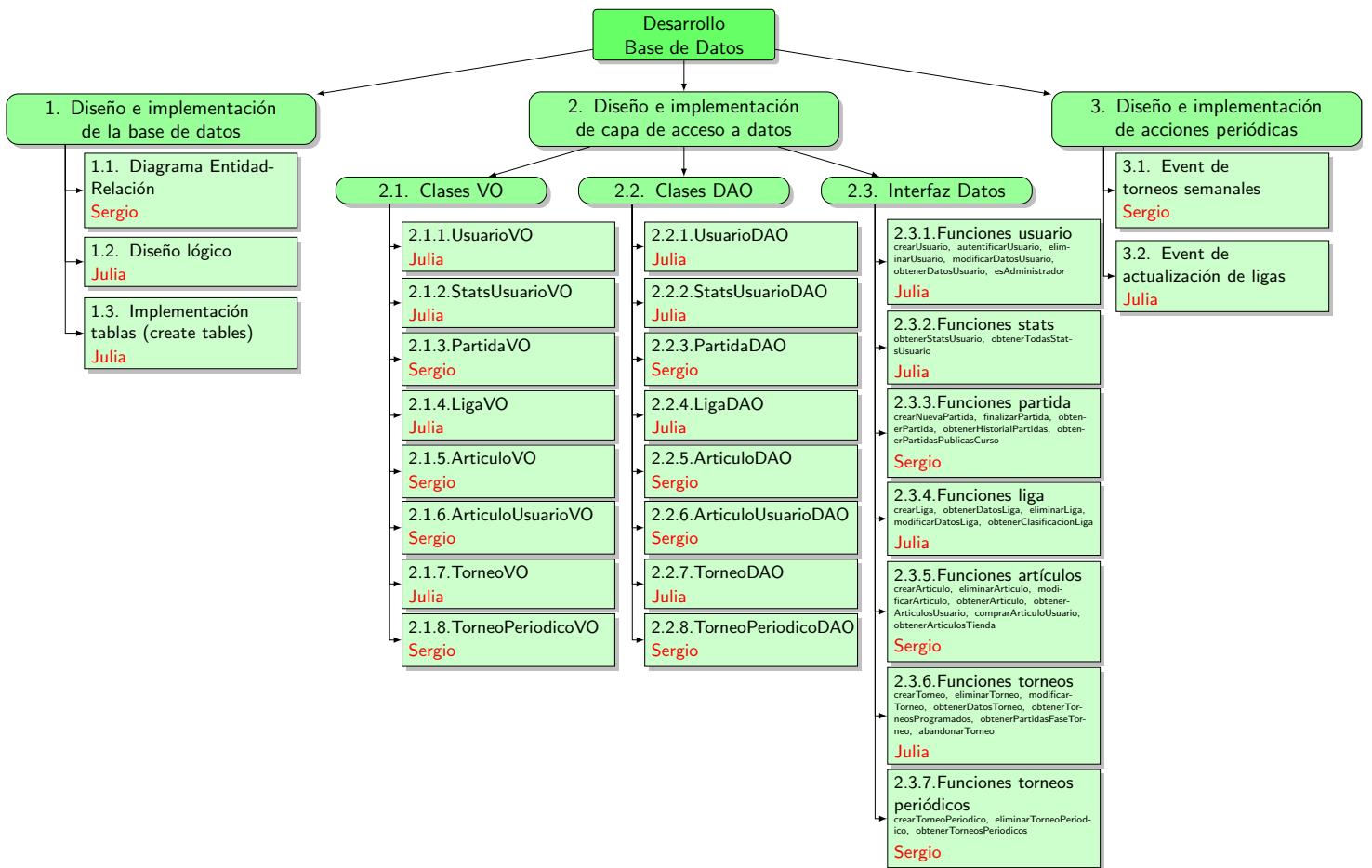


Figura 30: Diagrama de Paquetes de Trabajo Bases de Datos

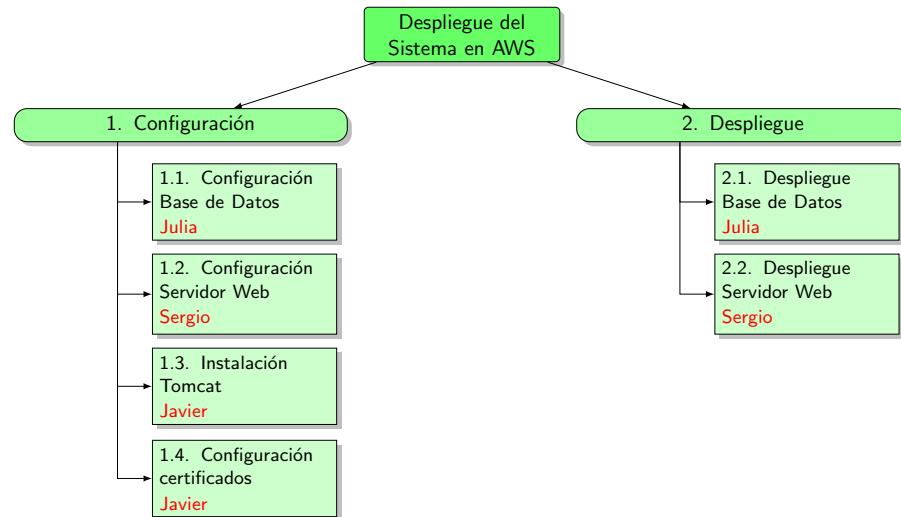


Figura 31: Diagrama de Paquetes de Trabajo Despliegue

Es importante añadir que el trabajo de integración de los diferentes paquetes de trabajo no está reflejado en estos diagramas pero también ha sido necesario repartirlo y realizarlo.

5.2.2. Comunicación interna

La decisión de la nueva estructura de grupos se realizó mediante una reunión entre los miembros afectados por el cambio. En cuanto a la parte de BackEnd, al comienzo del proyecto la comunicación ha sido bastante pobre, ya que en algunos momentos los desarrolladores han realizado la misma actividad por duplicado. En el desarrollo de los diferentes componentes cada miembro tenía su propio diseño en mente y ha sido necesario ponerse de acuerdo y plasmar el diseño en un diagrama UML de clases, que todos pudieran consultar y hacer referencia. Hacia el final de la primera iteración el equipo ha mejorado su comunicación en gran medida.

La comunicación entre los diferentes equipos se ha llevado a través del servicio de terceros Skype para, principalmente, resolver dudas acerca de como comunicar e integrar las distintas partes de la aplicación. Además, se utiliza el servicio de mensajería WhatsApp para reportar bugs, incidencias y comunicaciones asíncronas. También se han llevado a cabo varias de tipo presencial reuniones para especificar interfaces comunes para facilitar la integración.

Se ha utilizado Floobits para la programación por parejas para las partes de la aplicación más complejas.

5.2.3. Adecuación a las herramientas y tecnologías

La base de datos se ha desplegado en el gestor elegido en un primer momento, MySQL en Amazon Aurora. Para la implementación se eligió Java como lenguaje de programación lo que ha permitido la utilización de librerías para la conexión de la base de datos (JDBC y c3p0). El desarrollo en Java fue acompañado del software IntelliJ que permitía el soporte para las librerías y el control de versiones. Además permite programar en Java, por lo que el uso de la herramienta es una ventaja.

En cuanto al desarrollo de la interfaz de juego se ha utilizado la herramienta WebStorm, en un primer momento, ya que permite de forma fácil hacer cambios en la interfaz y verlos reflejados de manera inmediata en el navegador. Sin embargo, al integrar esta parte con el resto del proyecto se ha seguido su desarrollo con el software IntelliJ. De esta manera, se consigue probar el funcionamiento del juego gracias al despliegue con el servidor Tomcat para poder probar las comunicaciones entre jugadores.

Una vez se despliega el sistema en Amazon, para depurar las comunicaciones basta con modificar aquel fichero que se necesitaba ajustar, ya que simplemente había que ajustar parámetros de retardos al tratarse de un sistema distribuido en cuanto al paso de mensajes se refiere. De esta manera no se despliega toda la aplicación de nuevo sino que sólo una parte de ella.

Para el diseño de los diagramas de clases se ha utilizado la herramienta StarUML, la web draw.io y excel para los diagramas de las pruebas de caja blanca y las tablas para las pruebas de caja negra. Se ha utilizado un plugin Floobits de IntelliJ para la programación en parejas ya que nos permitía trabajar remotamente a dos o más miembros sobre el mismo fichero en tiempo real utilizando dos cursoros diferentes. Esta herramienta se ha utilizado para los métodos más difíciles como por ejemplo lanzarCarta de la clase EstadoPartida.

5.2.4. Control de versiones

La utilización de git ha sido de gran ayuda a la hora de integrar el código desarrollado por cada miembro del grupo. Si algún miembro de un equipo modifica algún fichero de otro equipo, el commit que se realice especifica de forma clara y directa qué es lo que ha modificado. Para cambios grandes se pone en contacto con el responsable del equipo para solucionar el problema. Se ha utilizado git desde el software de desarrollo IntelliJ IDEA, en vez de hacerlo directamente sobre la línea de comandos, para subir y actualizar el proyecto de manera más rápida y cómoda. Esto ha sumado en el tiempo de aprendizaje del uso de los IDE's. Para evitar problemas relacionados con que dos miembros del equipo hagan el mismo trabajo se realizan commits de manera frecuente y se utilizan las task lists de GitHub para llevar un control de lo que se ha realizado o de lo que falta por realizarse o probar. Para ficheros con información privada como un fichero *properties* con la información privada de acceso a la base de datos, no se utiliza un software de control de versiones ya que no debe de estar subido al repositorio público. Al no existir este tipo de ficheros en un repositorio público, se tiene que compartir de forma privada y cada miembro debe colocar el fichero teniendo cuidado de provocar errores al compilar dependiendo de la ruta del fichero.

En cuanto a la integración entre el acceso a datos y el backend, se detectó un claro problema de comunicación entre los grupos a mitad del desarrollo debido a que no existía una interfaz definida de forma precisa desde un primer momento. Únicamente se había comentado una interfaz ambigua que diferentes partes entendían de forma distinta, y que además, sin las definiciones concretas de las funciones, impedía llevar a cabo el desarrollo en paralelo. Cuando se detectó el problema, hubo una reunión entre ambos grupos para definir esta interfaz de acceso a los datos. Gracias a este incidente se ha aprendido la importancia del desarrollo de interfaces entre los distintos componentes de los sistemas. El despliegue del servidor de bases de datos se realizó en AWS sin mayores problemas. En cuanto al despliegue del servidor web hubo problemas debido a una configuración de red de la máquina virtual. Tras localizar el problema se pudo solventar y finalmente acceder al servidor mediante SSH sin problemas.

Para desplegar la aplicación web localmente, se ha utilizado el servidor Tomcat junto con la

herramienta IntelliJ. Se ha adaptado el código para que funcione en la última versión para evitar problemas con la versión del Tomcat de cada miembro. En cuanto a la parte jugable, se despliega directamente con la herramienta WebStorm y se ejecuta en el navegador Chrome.

5.2.5. Pruebas del software

Para asegurar el correcto funcionamiento del software, cada equipo ha realizado pruebas unitarias para las funciones más complicadas que puedan generar errores.

Para probar la interfaz de juego se ha utilizado la herramienta Jasmine¹, que permite hacer tests unitarios en JavaScript. Se ha creado un protipo de una partida de prueba y se comprueba que el software pasa todos los tests diseñados.

La base de datos implementada en MySQL fue probada mediante la inserción de datos falsos de ejemplo, y la realización de consultas sencillas sobre ella, que garantizan el correcto funcionamiento. La interfaz de acceso a datos ha sido probada mediante pruebas unitarias. Se desarrolló un programa de pruebas al final del desarrollo de cada clase DAO, probando función a función sobre los datos de ejemplo introducidos, y solucionando errores de implementación. Para comprobar el funcionamiento de las clases a una escala mayor se escribió un programa en Python que generaba el código de Java que probaba la base de datos insertando nuevos datos. De esta forma también se tiene la base de datos poblada con datos de apariencia real, lo que puede ser beneficioso para pruebas de otros equipos.

Todas las funciones de la lógica han sido sometido a algún tipo de prueba. El criterio que se ha seguido consiste en probar la funciones más triviales mediante pruebas de caja negra utilizando la técnica de clases de equivalencia y análisis de casos extremos para las funciones más triviales que operan en ciertos rango. Por ejemplo, para la función de getPuntuación perteneciente al módulo Carta se ha utilizado la técnica de análisis de casos extremos. De esta forma se prueban todas las cartas que tienen una puntuación mayor que 0 y una de las que tienen puntuación 0.

| Devuelve la puntuación de la carta según las normas del guíñote | |
|---|-----------|
| public int getPuntuación() | |
| Casos extremos (valor de la carta) | Resultado |
| 1 | 11 |
| 3 | 10 |
| 12 | 4 |
| 10 | 3 |
| 11 | 2 |
| 2 | 0 |

Cuadro 6: Pruebas de la función getPuntuación().

Para funciones con más complejidad como pueden ser los constructores y que además están formados por bucles se realizan pruebas de caja blanca utilizando la técnica de análisis de caminos. Este tipo de pruebas es especialmente utilizado en los constructores de las clases y algunos getters no triviales de dichas clases. A continuación se muestra el grafo de caminos y los valores de las pruebas para la función getCartasEnMano de la clase Jugador.

¹ <https://jasmine.github.io/>

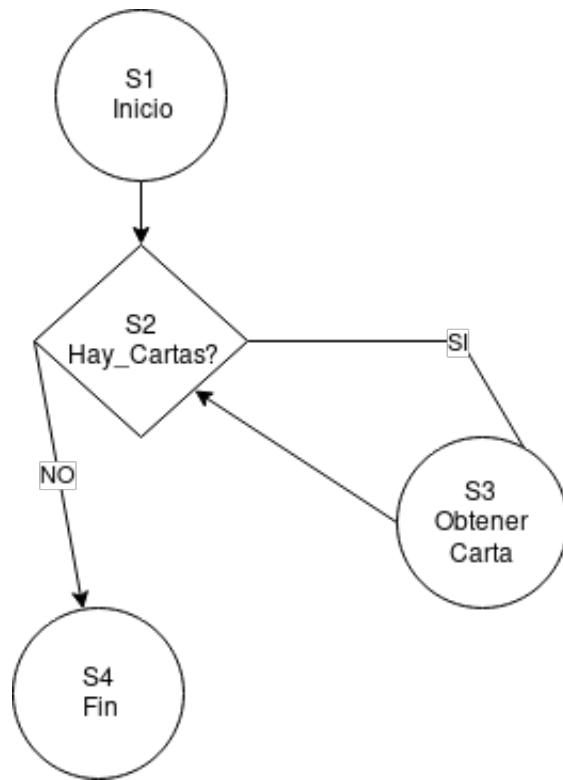


Figura 32: Grafo de caminos de la función.

| | | |
|---|----------------------------------|-----------|
| Devuelve una lista con las cartas en mano del jugador | | |
| public int getPuntuación() | | |
| Cartas del jugador | Camino de la ejecución (estados) | Resultado |
| C1 | S1, S2, S3, S4 | C1 |

Cuadro 7: Entorno de prueba de la función.

Basta con una ejecución de la función porque si el jugador tiene solo una carta en la mano se prueban todos los caminos posibles del grafo.

Para las más complejas de la lógica como la función lanzarCarta de EstadoPartida y la función siguienteRonda se han realizado pruebas de caja negra para comprobar el correcto funcionamiento ya que son las funciones más importantes de la lógica del guíñote.

Para la función lanzarCarta las pruebas se realizan para los casos donde ya no quedan cartas en el mazo (ronda de arrastre), debido a que antes del arrastre no existen restricciones en las cartas que se pueden tirar. Se separan las pruebas en dos: las pruebas para una partida de uno contra uno al segundo jugador tiene que lanzar una carta y las pruebas para el tercer jugador en lanzar cuando la partida es dos contra dos. No se han realizado pruebas para el primer jugador en lanzar ya que no tiene restricciones ni al cuarto ya que el código ejecutado es el mismo que para el tercer jugador.

| | |
|---|---|
| Lanza la carta c por el Jugador "jugador". Si no es su turno, no posee dicha carta o incumple alguna de las reglas del guñote, lanza una excepción. | |
| <pre>public EstadoPartida lanzarCarta(String jugador, Carta c)</pre> | |
| Casos durante la ronda de arrastre (2º jugador) | Resultado |
| Lanza una carta de un palo diferente al inicial teniendo del mismo palo | Lanza excepción CartaIncorrecta |
| Lanza una carta del mismo palo al inicial pero de valor inferior teniendo una carta de valor superior y mismo palo al inicial | Lanza excepción CartaIncorrecta |
| Sin tener cartas del palo de la carta inicial, lanza una carta de un palo cualquiera teniendo triunfo | Lanza excepción CartaIncorrecta |
| Lanza una carta superior del mismo palo al inicial | Se añade la carta a la lista de cartas en el tapete |
| Lanza una carta inferior y del mismo palo a la inicial sin tener en la mano una superior del mismo palo | Se añade la carta a la lista de cartas en el tapete |
| Sin tener cartas del palo inicial, lanza una carta del palo del triunfo | Se añade la carta a la lista de cartas en el tapete |
| Sin tener cartas del palo inicial o del palo del triunfo, lanza una carta cualquiera | Se añade la carta a la lista de cartas en el tapete |
| Casos durante la ronda de arrastre (3º jugador) | Resultado |
| Lanza una carta de un palo diferente al inicial teniendo del mismo palo | Lanza excepción CartaIncorrecta |
| Con obligación de matar(el compañero no tiene la carta con mayor valor), lanza una carta que no mate teniendo una carta de valor superior y mismo palo al inicial que mata al rival | Lanza excepción CartaIncorrecta |
| Con obligación de matar(el compañero no tiene la carta con mayor valor) y sin tener carta del palo inicial, lanza una carta que no mate teniendo triunfo | Lanza excepción CartaIncorrecta |
| Habiendo matado el rival sin triunfo, lanza una carta superior del mismo palo al inicial | Se añade la carta a la lista de cartas en el tapete |
| Habiendo matado el rival sin triunfo, lanza triunfo para matar | Se añade la carta a la lista de cartas en el tapete |
| Habiendo matado el compañero y sin tener del palo inicial, lanza una carta de un palo cualquiera | Se añade la carta a la lista de cartas en el tapete |

Para la función siguienteRonda se ha probado cada caso donde un jugador diferente ganaba y se ha comprobado que sumaba los puntos correspondientes al equipo que ganaba. Además se ha verificado que tras la última baza si se alcanzaban los 100 puntos por parte de un equipo se daba por finalizada la partida y en caso de no alcanzarse se daba comienzo a la partida de vueltas.

Al tratarse de un videojuego, es difícil abordar todos las posibles situaciones de la partida, por lo que para probarlo, se hace el trabajo similar al de un tester, donde juega partidas y fuerza situaciones que se salen de lo normal para comprobar el comportamiento de la partida. Cuando se detecta algún bug, se notifica a través de un issue en GitHub para que el equipo causante del mismo lo solucione.

Para las pruebas funcionales, una vez probados los componentes por separado y una vez integrados, todos los miembros del equipo han probado el sistema desplegado haciendo pruebas manuales, como

| Asigna el turno y puntuación de las cartas del tapete de cada ronda al ganador de la ronda public EstadoPartida siguienteRonda() | |
|---|---|
| Casos (4 jugadores) | Resultado |
| Se ejecuta la función habiendo lanzado carta los tres primeros jugadores | Lanza excepción RondaNoAcabada |
| Se ejecuta sin que ningún jugador lance triunfo ni supere al primer jugador | Turno==0 ganadorUltimaRonda==0 Puntuación j1 igual a la suma del valor de las cartas Puntuación j1 == Puntuación j3 |
| El jugador 2 mata con una carta superior | Turno==1 ganadorUltimaRonda==1 Puntuación j2 igual a la suma del valor de las cartas Puntuación j2 == Puntuación j4 |
| El jugador 3 mata con un triunfo después de que el jugador 2 haya matado | Turno==2 ganadorUltimaRonda==2 Puntuación j1 igual a la suma del valor de las cartas Puntuación j1 == Puntuación j3 |
| El jugador 4 mata con un triunfo superior después de que el jugador 3 haya matado | Turno==3 ganadorUltimaRonda==3 Puntuación j2 igual a la suma del valor de las cartas Puntuación j2 == Puntuación j4 |
| En la última baza el jugador 4 mata con un triunfo superior después de que el jugador 3 haya matado sin superar los 100 puntos | Turno==3 ganadorUltimaRonda==3 Puntuación j2 igual a la suma del valor de las cartas Puntuación j2 == Puntuación j4 de Vueltas==1 |
| En la última baza el jugador 4 mata con un triunfo superior después de que el jugador 3 haya matado superando así los 100 puntos | Lanza excepción PartidaFinalizada |

Cuadro 8: Pruebas de la función siguienteRonda().

si fuesen usuarios, con todas las funcionalidades detalladas en las especificaciones del sistema para determinar que se cumple los objetivos del sistema y se realizan correctamente.

Para realizar las pruebas de rendimiento del sistema, se han jugado varias partidas para comprobar que el tiempo de respuesta tanto de la IA como del sistema es razonable. También se ha probado el sistema en entornos con una conexión mala a internet. En dichas situaciones se percibe un retardo inicial considerable para cargar todas las imágenes de la interfaz, aunque una vez iniciado el juego, el funcionamiento es bastante razonable obteniendo un tiempo de respuesta inferior a los 2 segundos en la mayoría de los casos.

Dado que la máquina de Amazon no dispone de unas especificaciones técnicas muy buenas se ha decidido realizar las pruebas de sobrecarga en un ordenador personal. Se han jugado 10 partidas simultáneas estando el servidor desplegado en un Intel i7-7500 y el juego no sufre grandes retrados en las comunicaciones. Además, pueden estar navegando por la página web aproximadamente unos 15 usuarios sin percibir un efecto de relantización. El límite de conexiones a la base de datos es de 90, ya que lo establece Amazon AWS según el servicio contratado.

En las pruebas de volumen realizadas en la base de datos se han insertados unos 500 usuarios y todo funcionaba correctamente. Así que, por parte de la base de datos la única limitación que se tiene es el número de conexiones simultáneas, aunque estas podrían resolverse contratando una tarifa

mayor en Amazon AWS en un futuro.

Las pruebas de seguridad realizadas en la páginas web se basan verificar que la entrada de datos están protegidas, de forma que un usuario no pueda acceder a la cuenta de otro sin saberse su contraseña y usuario. Además, se ha probado a acceder directamente a diferentes vistas del administrador sin serlo, o a ciertas ventanas con contenido específico de cada usuario, como la tienda o perfil, y su acceso fue denegado.

Progreso y Divergencias frente a la planificación inicial

Para la primera iteración era necesario realizar una demostración al cliente de una versión funcional el juego del guiñote. Se ha realizado una versión básica funcional jugable a través de interfaz. Sin embargo, se ha dejado para la segunda iteración el requisito de cambio de dispositivo y de personalización del juego principalmente. Además, se garantizó que el modo espectador debía estar implementado. Sin embargo, no está desarrollado plenamente. La situación actual de la implementación de dicho requisito está casi terminada, a falta de la integración con la comunicación entre capas y las pruebas correspondientes.

En el primer plan de gestión se especificó que tanto el matchmaking y como la web dinámica deberían estar avanzados para la primera iteración. En cuanto a la web dinámica, faltan bastantes vistas y funcionalidades básicas debido a la sobre carga de trabajo que han sufrido los equipos de backend y frontend. Ambos equipos se centraron en obtener las funcionalidades básicas del juego dejando de lado el resto de aspectos. Si bien es cierto que ya se ha alcanzado un tercio del total, mientras que el objetivo en esta primera iteración era de la mitad.

En cuanto a la capa de persistencia y la capa de acceso de datos, el equipo de bases de datos ha cumplido el diagrama de Gantt establecido, exceptuando la implementación del acceso a datos relacionado con los torneos, que estaba planificado para la primera iteración y se debe alargar a la segunda, sin suponer ningún problema de dependencias con el resto de componentes del sistema y que no se estima muy costosa. También se puede comentar que, en lo que respecta al diseño de la base de datos, se requirió algo menos tiempo del estimado (se finalizó con una semana de antelación), mientras que el diseño del acceso a datos requirió más tiempo del estimado por lo que fueron compensados, cumpliendo bastante bien la planificación inicial.

En cuanto a la 2^a iteración, se ha conseguido implementar el resto de requisitos en su plenitud. De los cuales, los más importantes eran el matchmaking, torneos, espectadores, permitir el cambio de dispositivo y la incorporación de la inteligencia artificial. El más problemático ha sido implementar la posibilidad de los torneos, ya que el equipo se ha enfrentado a diferentes problemas de diseño relacionados con el abandono de jugadores. La integración de la inteligencia artificial también ha resultado ser problemática ya que respondía prácticamente de manera instantánea y el jugador no podía ver la carta que ésta había lanzado.

Debido a los problemas comentados anteriormente las horas de trabajo se incrementan considerablemente. Además, habría que sumarle los problemas relacionados al despliegue y la latencia de la red que aparece y como lidiar con este nuevo problema.

BackEnd

El equipo ha cumplido el diagrama de Gantt en el apartado de implementación de lógica del juego. Si bien es cierto que se ha logrado con un retraso de media semana. Puesto que el equipo se centró en la implementación de la lógica la parte de desarrollo de la web dinámica no ha cumplido el

calendario impuesto al comienzo del proyecto y arrastra un leve retraso que deberá ser compensado en la segunda iteración. El retraso se debe a la gran cantidad de trabajo asignado en la planificación inicial, muy superior a la capacidad del equipo.

Debido a las divergencias entre las horas de implementación reales y las estimaciones iniciales, el equipo ha decidido realizar un registro y posteriormente un estudio de las horas empleadas en el desarrollo del proyecto. El equipo ha realizado 349 horas de trabajo, de las cuales 106 se han invertido en tareas de gestión, una cifra muy superior a establecida en la estimación inicial. También cabe destacar las 33 horas de realización de pruebas y corrección de errores, que deberían asegurar la calidad del proyecto. En cuanto a las estimaciones sobre los diferentes componentes, la implementación del juego y su comunicación, junto con el desarrollo de la web dinámica están realizadas de forma precisa, completando las horas asignadas. Sin embargo, los componentes de la base de datos y las vistas de la web se han sobredimensionado ligeramente, por lo que el equipo ha tenido que realizar menos trabajo del especificado.

5.3. Cierre del proyecto

Los procesos de control no se han llevado a cabo tal y como estipulaba el primer plan de gestión, ya que el director del proyecto no se ha coordinado con los responsables de grupo. Sin embargo a nivel local, los responsables de grupo sí han dirigido y organizado sus respectivos grupos. Además no se ha producido ninguna situación que requiera de mediación ni necesidad de intervención por parte de personas ajenas al equipo.

5.3.1. Planificación

Con respecto a la planificación y los cambios que ha ido sufriendo esta a lo largo del proyecto, se muestran a continuación los diagramas de Gantt inicial y final.

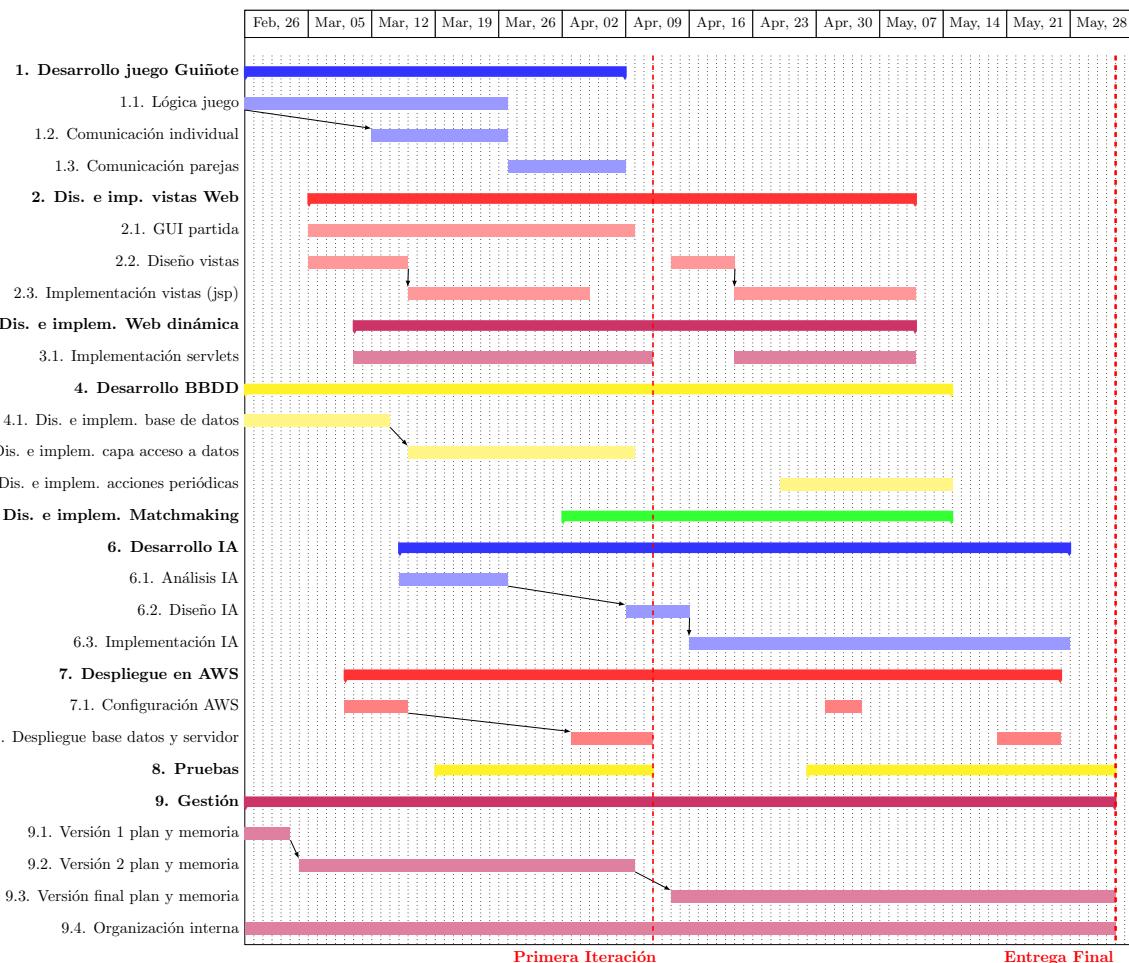


Figura 33: Diagrama de Gantt Inicial (02/03/2018)

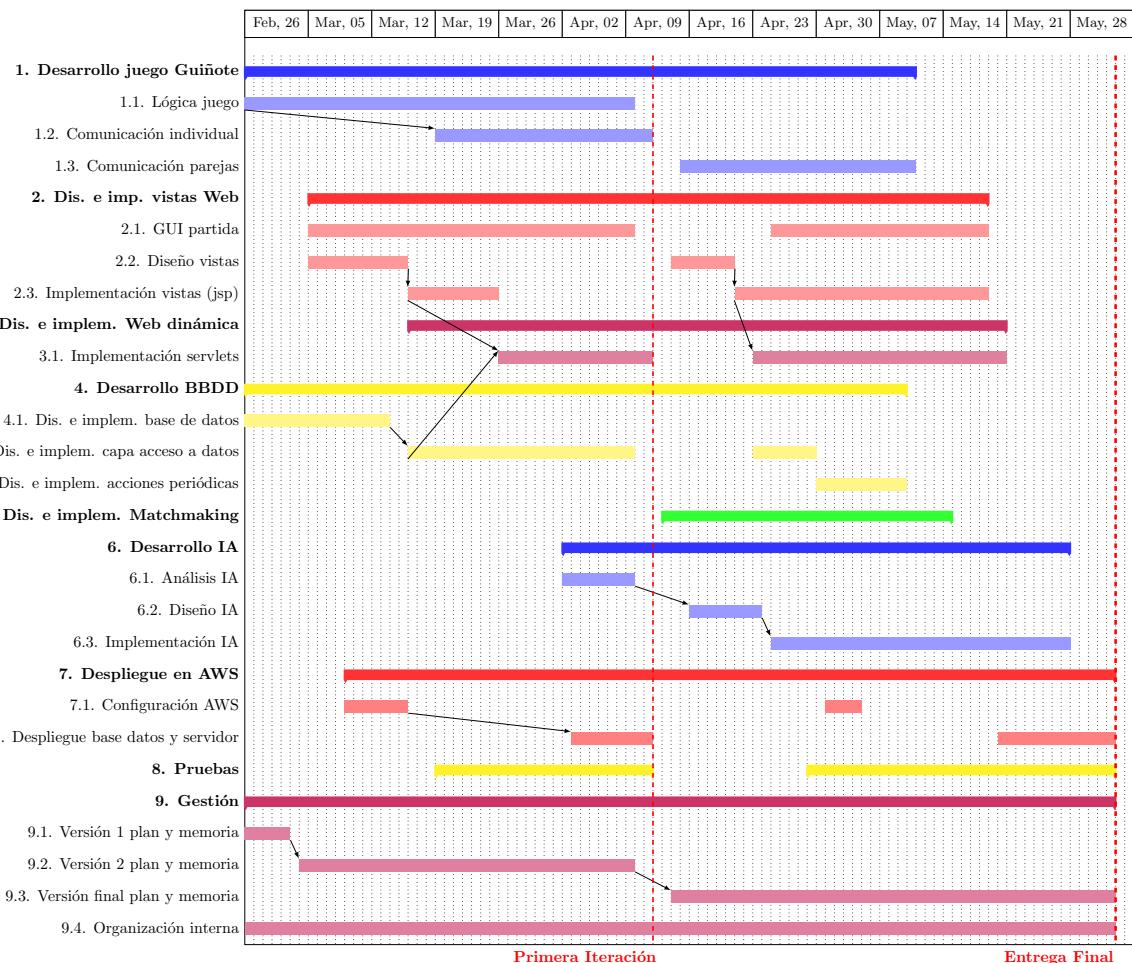


Figura 34: Diagrama de Gantt Final (01/06/2018)

Como se puede observar, ha habido algunos cambios con respecto a los plazos planificados y los plazos que realmente ocurrieron.

Destaca claramente en primer lugar la larga duración del desarrollo del juego en relación con lo planificado, ya que esperaba ser terminado en la primera iteración y finalmente se alargó hasta pasada la mitad de la segunda. Esto se debe a que, como se reflejará posteriormente en la distribución de horas, no se estimó correctamente la dificultad que requería un juego multijugador como el guiñote, especialmente en las partes de lógica y la depuración de esta, y la gestión de las comunicaciones (principalmente estas últimas en el entorno de despliegue).

En cuanto al desarrollo de las vistas web y la web dinámica, la planificación se ha cumplido bastante, alargándose algo menos de una semana al final, en especial porque no se detectaron correctamente las dependencias en el diagrama de Gantt inicialmente planteado, y porque se alargó la parte de desarrollo de la GUI del juego, al añadir todas las personalizaciones de artículos, posibilidad de espectadores y efectos de sonido.

El desarrollo de la base de datos también cumplió con los plazos establecidos. Como diferencias, destacar que el desarrollo de la interfaz de datos se vio alargado por la dificultad extra que entrañaban los torneos y por la reestructuración de equipos que centró a estos en el análisis e implementación de la IA durante las primeras semanas de la segunda iteración. Por el contrario, la implementación de acciones periódicas supuso menos tiempo del planificado con lo que se terminó esta parte adelantándose a los plazos establecidos.

La implementación del matchmaking sufrió un ligero retraso en su instante de comienzo, pero no se retrasó mucho de lo estimado en un principio, como se refleja en el número de horas dedicadas a este. De forma similar, el desarrollo de la inteligencia artificial también se retrasó en su inicio pero finalmente cumplió con todos los plazos estipulados.

Finalmente, en cuanto al despliegue, aunque se cumplieron los plazos, se detectó que no se había planificado correctamente con la antelación suficiente para poder solucionar todos los errores (principalmente de latencias) que suponía garantizar el funcionamiento de la aplicación de forma robusta, lo que implicó un trabajo extra de horas en las últimas semanas del proyecto, que no se esperaba.

Como decisión importante de planificación, destaca que, al final de la primera iteración, tras un análisis del progreso del proyecto y como respuesta a los problemas de divergencias de horas frente a la planificación, se realizó una reestructuración de los equipos con respecto al de Inteligencia Artificial. Los miembros del equipo de Lógica y BackEnd se mantuvieron en este únicamente durante la segunda iteración y fueron los miembros del equipo de Base de Datos los que comenzaron a trabajar en la IA, dejando la organización final como la que se muestra en el organigrama de la sección 2. Se decidió incorporar a los nuevos miembros en este equipo y no a otras tareas porque era el único que todavía no había comenzado a trabajar y, por tanto, la reestructuración no afecta al rendimiento del equipo de trabajo como ocurriría si fuesen añadidos a mitad (tal y como se explica en el libro *Mythical Man-month* cuyo tema principal es la idea de '*adding manpower to a late software project makes it later*' [4].)

Como conclusión, destacar que la planificación ha sido un elemento esencial de la gestión de este proyecto, y que, a pesar de que pueda haber bastantes divergencias, solamente gracias a ser conscientes de ello se ha aprendido mucho para futuros proyectos.

5.3.2. Comparación estimaciones iniciales

En la fase inicial del proyecto se realizó una estimación inicial del coste de 630 horas de desarrollo de software y 170 horas en gestión del proyecto, de la calidad del software y de las configuraciones. Un total de 800 horas, de las cuales se han llevado a cabo 778 horas, un 97%. A pesar de que la estimación global se haya cumplido, una análisis más fino de la distribución de las horas revela unas conclusiones diferentes.

Como se puede apreciar en la tabla, la distribución de las horas estimadas y la de las invertidas son muy diferentes. Estas divergencias entre la estimación y los resultados finales se deben a la falta de experiencia en la realización de proyectos de esta magnitud, el desconocimiento de la dificultad del aseguramiento de la calidad en juegos online y algunos errores en el diseño del proyecto. Como por ejemplo la subestimación de los componentes de comunicación y sincronización.

A nivel individual el equipo ha trabajado de forma uniforme y repartiendo la carga de trabajo. La división de los integrantes del equipo en grupos de trabajo se ha llevado a cabo de forma efectiva

| Distribución de las horas del proyecto en diferentes agrupaciones | | | |
|---|------------|------------------|---------------------|
| Agrupacion | Estimacion | Horas invertidas | Porcentaje cumplido |
| Desarrollo del juego | 128 h | 160 h | 125 % |
| Pagina web | 140 h | 82 h | 58 % |
| Base de datos | 122 | 88 h | 72 % |
| Despliegue | 10 h | 20 h | 200 % |
| Implementacion IA | 85 h | 64 h | 75 % |
| MatchMaking | 35 h | 26 h | 74 % |
| Gestión | 95 h | 141 h | 148 % |
| Gestión de configuraciones | 31 h | 20 h | 64 % |
| Aseguramiento de la calidad | 44 h | 136 h | 300 % |
| Integración | 0 h | 22 h | - % |

Cuadro 9: Distribucion de las horas por tipo de actividad

y muestra de ello es la siguiente tabla que recoge un resumen del trabajo de cada miembro.

| Distribución de las horas del proyecto por persona | | |
|--|------------------|---|
| Nombre | Horas trabajadas | Actividad principal |
| Javier | 106 h | Gestor de mensajes y MatchMaking |
| Marius | 116 h | Implementación de la lógica del juego y servlets |
| Sergio | 128 h | Diseño de la base de datos y representación del problema de la IA |
| Carlos | 108 h | Interfaz y vistas web |
| Víctor | 117 h | Web dinámica y representación de la lógica del juego |
| Julia | 123 h | Diseño de la base de datos y análisis de la IA |
| Ignacio | 79 h | Pruebas y lógica del juego |

Cuadro 10: Distribucion de las horas por persona

5.3.3. Lecciones aprendidas sobre gestión

Algunas de las tareas de gestión realizadas a lo largo de la ejecución del proyecto han ayudado a su correcta realización. Por ejemplo, gracias a la contabilización de horas se ha podido comparar con exactitud la estimación con el tiempo real lo que permite que la próxima vez se realicen estimaciones más precisas. Además dicha contabilización permite repartir las tareas entre los miembros de los equipos de forma más justa.

La creación de paquetes de trabajo ha permitido que el reparto de tareas sea claro y transparente lo que evita confusiones entre los miembros de los grupos y realización de tareas repetidas o por el contrario la falta de su realización.

Gracias a la existencia de un documento central con la gestión se podían consultar algunos apartados en caso de duda. Por ejemplo cuando se dudaba sobre el comportamiento de la aplicación se tenía recogido el reglamento del Guiñote así como los requisitos, donde se detallaba que

características debía de poseer el sistema sin ambigüedades.

EL diagrama de Gantt permitía ver el progreso según lo planificado. A lo largo del proyecto se ha aprendido a aprender a valorar las diferencias entre lo estimado y lo real, pues en un primer momento no parecían graves las discordancias pero solían acabar en retrasos en la finalización de las tareas. Es imposible conseguir que la estimación sea igual a la realidad pero se ha aprendido a tomar medidas para corregir y detectar estas diferencias.

5.3.4. Lecciones aprendidas sobre herramientas y tecnologías

La experiencia con IntelliJ, el entorno con el que se ha desarrollado el sistema, ha sido positiva en su totalidad. Gracias a su plena integración con Git, no hace falta realizar ninguna acción externa a la aplicación. Sin embargo, conforme el proyecto se iba desarrollando, los problemas de configuración del mismo aumentaban debido a las dependencias y la organización interna del mismo. Esto no habría pasado si se hubiera configurado con Maven o Graddle en vez de hacerlo manualmente.

En cuanto a la tecnología Phaser, la experiencia ha resultado ser positiva. Se trata de un software fácil de usar por lo que su aprendizaje no ha sido del todo costoso. Además, gracias a la comunidad disponible se han podido resolver las dudas sin problema alguno. La utilización de WebSockets para el paso de mensajes ha resultado más costosa ya que se deben tratar las desconexiones de los jugadores de manera individual. Habría sido más efectivo tratar de utilizar una tecnología basada en eventos, por ejemplo.

El software utilizado para los diagramas ha sido StarUML. Éste ha ofrecido una experiencia neutra. Cumple en todos sus aspectos pero el tener la versión gratuita hace que cada vez que sea ejecutado aparezca el mensaje de compra.

Todo el proyecto salvo la interfaz ha sido implementada en Java ya que era una tecnología con la que ya se ha trabajado pero en retrospectiva se considera que se podría haber ahorrado tiempo utilizando un lenguaje de programación menos verboso como Kotlin (por compatibilidad con la JVM), Go, Python, Javascript, etc, que además ofrecen facilidades para la gestión de dependencias, CI/CD y escalabilidad horizontal en Kubernetes por ejemplo, lo que sería especialmente útil para gestionar grandes volúmenes de partidas a la vez.

6. Conclusiones

Desde el inicio hasta el final del proyecto, el equipo Margaret Hamilton hemos adquirido mucha experiencia en el proceso de gestión y control de un proyecto de tamaño medio. Dónde la división del trabajo, la asignación de recursos y tareas, la calidad del software y la comunicación son esenciales. Dada la complejidad del proyecto escogido, el desarrollo del producto ha sido una tarea muy ardua. Esto se debe a la existencia de una gran cantidad de componentes diferentes y las dependencias entre estos. Además, la integración entre estos componentes ha consumido gran cantidad de los recursos del equipo. Sin embargo, esta característica del proyecto ha servido para que el equipo adquiera experiencia en el desarrollo de aplicaciones donde la sincronización y comunicación entre los sistemas es crítica. También se ha adquirido experiencia en el desarrollo de sistemas de inteligencia artificial donde la información del estado es incompleta y la evolución de los estados no es determinista. Dichos sistemas están en auge y tienen futuro prometedor.

Para la realización de futuros proyectos, gran parte de la experiencia adquirida será de gran utilidad. En el apartado de estimación de costes y planificación antes de iniciar el proyecto se ha adquirido una buena referencia que junto con otros proyectos sienten la base de unas estimaciones correctas. En el futuro, a la hora de escoger las tecnologías con las que desarrollar los proyectos se evitará utilizar java debido a su bajo rendimiento y falta de seguridad. En cuanto a los procesos técnicos se ha aprendido mucho en cuanto a la definición de interfaces y la documentación. En el futuro la definición de interfaces ocupará un lugar principal en el desarrollo de paquetes. Por otro lado, el apartado de pruebas deberá tener en cuenta la integración entre los diferentes componentes y los posibles efectos colaterales en la comunicación entre estos.

Gracias a las nuevas habilidades adquiridas a lo largo de este proyecto los intergrantes han logrado aumentar sus capacidades de trabajo colaborativo para mejorar el nivel de la calidad del software ofrecido.

Anexo I. Glosario

Amazon AWS: Amazon Web Services, plataforma cloud ofrecida por Amazon. [2]

Amazon RDS Aurora: Servicio de creación y mantenimiento de bases de datos relacionales. [3]

Amazon EC2: Servicio de servidores privados virtuales de AWS que permite lanzar máquinas virtuales con los sistemas operativos y configuraciones deseadas. Además, permite contratar más o menos recursos bajo demanda, adaptándose así a los picos de carga. [1]

PostgreSQL: es un sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia PostgreSQL.

MySQL: es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation.

Guíñote: Juego de cartas español en el que pueden participar dos jugadores o dos parejas de jugadores. Se utiliza la baraja española de 40 cartas con cuatro palos. A continuación se explican detalladamente las reglas del juego:

- El juego comienza repartiendo 6 cartas a cada jugador y colocando una carta en medio del tapete que determina el palo que es "triunfo". La manera de jugar es por rondas denominadas bazas, en las que cada jugador tira una carta. La baza se gana si se tira el triunfo más alto, o si no hay triunfo, si se tira la carta con mayor valor del palo de la carta de salida. Al ganar la baza, se suma a la puntuación de la pareja o el jugador el valor de las cartas. Además si se gana la baza se puede intercambiar el siete del palo de triunfo por la carta que se encuentra en medio del tapa si su valor es superior a la del siete, o también se puede sumar puntuación si el jugador tiene un "cante" que significa que el jugador posee la sota y el rey de un mismo palo, lo cual da 20 puntos si el palo no es triunfo y 40 puntos si el palo es triunfo.
- El valor de las cartas de forma decreciente es el siguiente: As (11 pts), tres (10 pts), rey (4 pts), sota (3 pts), caballo (2 pts), siete (0 pts), seis (0 pts), cinco (0 pts), cuatro (0 pts) y dos (0 pts).
- Después de cada baza, y mientras queden cartas en el mazo, cada jugador roba una carta empezando por el jugador que se haya llevado la última baza. Tras esto, será el primer jugador en jugar la primera carta de la siguiente baza.
- Una vez que se acaban las cartas para robar se introducen restricciones a la hora de tirar las cartas. El primer jugador en tirar no tiene restricciones pero el resto de jugadores deberá tirar una carta del mismo palo o si no tiene, triunfo, además de que está obligado a "matar", es decir, tirar una carta para intentar ganar la baza. En caso de que el jugador no pueda cumplir ninguna de estas restricciones con las cartas que posee puede tirar la carta que desee. Finalmente, el equipo que gane la última baza ganará 10 puntos extra.
- Para determinar quién ha ganado, se suma la puntuación y gana el jugador o la pareja que supere los 100 puntos (divididos habitualmente en 50 "malas", los primeros 50 puntos, y 50 "buenas", los restantes). En caso de que nadie supere la anterior cifra se vuelve a repartir y a cada baza que se gana se incrementa la puntuación hasta que se llega a 100, momento en el que se gana. En el caso de que ambos equipos superasen los 100 puntos en la misma jugada, ganaría aquel que hubiese ganado la última baza.

Anexo II. Actas

Acta de reunión 1 Grupo 4

Ubicación: Sala 2.11, edificio Ada Byron, EINA (Universidad Zaragoza)

Fecha: 14 / Febrero / 2018

Hora: 12:00h

Asistentes: Profesor: Javier Lacasta. Equipo: Marius Crisán, Víctor Soria, Javier Corbalán, Carlos Marañés, Sergio Izquierdo, Ignacio Bitrián, Julia Guerrero.

Desarrollo de la sesión

Se presenta al profesor la idea de proyecto: Una aplicación web para jugar al guiñote. Se discuten, ideas que en un principio eran muy preliminares, las siguientes funcionalidades a incluir en el sistema:

1. Partidas síncronas con posibilidad de pausa del juego.
2. Posibilidad de jugar uno contra uno o dos contra dos.
3. Aplicación web para Google Chrome, responsive para poder acceder desde distintos dispositivos.
4. Autenticación vía correo electrónico y Facebook.
5. Torneos (partidas eliminatorias) y ligas (basadas en sistemas de puntos y que agrupen a usuarios por niveles). Matching con jugadores de un mismo nivel.
6. Administración (como una opción de la propia aplicación para usuarios con permisos) que permita gestionar ligas y torneos.
7. Posibilidad de personalización a través de una tienda con tres barajas y tres tapetes diferentes. Se decide añadir avatares (a decidir el número) de identificación de cada jugador a la tienda.
8. Divisa para comprar artículos de la tienda basada en recompensas por partidas ganadas.
9. Finalmente, posibilidad de añadir una IA que juegue en las partidas uno contra uno, añadiendo complejidad y carga de trabajo (alrededor de una persona), a implementar con árboles expectimax.
10. Otro aspecto a considerar añadir sería un chat, teniendo en cuenta que añade bastante complejidad (a decidir si comunicación vía el servidor o cliente-cliente).

Se esboza la arquitectura del sistema, distinguiendo entre la base de datos, el servidor de tomcat y la aplicación, realizando también un boceto muy preliminar de la GUI principal de la partida de guiñote.

Sobre tecnologías a utilizar se habla de JSP, y Jquery, HTML5 para la aplicación web (aunque todavía queda pendiente tomar la decisión de qué se utilizará) y Tomcat para el servidor. Utilización de Amazon para alojar tanto el almacenamiento como el servicio (aunque el servicio para estudiantes sea gratuito, pendiente consultar precios para añadir a la propuesta económica).

En referencia a la carga de trabajo, el profesor hace una estimación general, sin entrar en detalle de cada una de las partes del sistema, estimando que el trabajo comentado (sin chat) corresponde al número de horas a dedicar por unos cuatro alumnos y medio.

Se realiza una foto de grupo con los siete integrantes del equipo.

Finalmente, se comentan brevemente temas de gestión del proyecto, especialmente el coste de las reuniones grupales (una hora por cada persona lleva a invertir siete horas de trabajo en ellas) y, por tanto, la necesidad de minimizarlas y de una buena organización y documentación. Es necesario incluir el tiempo de gestión en el presupuesto.

Asuntos pendientes

Terminar de tomar decisiones y detallar mucho más cada uno de los requisitos, las tecnologías a utilizar y el trabajo requerido para implementar cada parte, pudiendo así realizar una mejor estimación de la carga de trabajo del proyecto. Queda abierta la posibilidad de añadir más funcionalidades si se necesitasen para cubrir la carga de trabajo correspondiente a siete alumnos.

Redactar completo el documento de propuesta técnica y económica, a entregar el lunes día 19 Febrero.

Establecer en la próxima reunión, en base a la propuesta realizada y a las apreciaciones convenientes del profesor, la propuesta técnica y económica final, para poder empezar a desarrollar el sistema.

Se cierra la sesión a las 13:25 horas.

Acta de reunión 2 Grupo 4

Ubicación: Sala 2.11, edificio Ada Byron, EINA (Universidad Zaragoza)

Fecha: 26 / Febrero / 2018

Hora: 11:00h

Asistentes: Profesor: Javier Lacasta. Equipo: Marius Crisán, Javier Corbalán, Carlos Marañés, Sergio Izquierdo, Julia Guerrero.

Desarrollo de la sesión

Se corrige la propuesta técnica y económica entregada previamente. A continuación, se explica la memoria de gestión del proyecto para la siguiente entrega y se discuten los aspectos más relevantes.

En cuanto a la corrección se discuten los requisitos. Uno de ellos es la compra de artículos en la tienda que no está bien especificado ya que se nombran únicamente los artículos pero no la tienda, por lo que hay que modificarlo.

Se propone añadir el requisito funcional de cambiar de dispositivo durante el transcurso de una partida, ya que no se indicaba en ninguna la parte.

Se indica que las cifras que se presentan en el documento han de estar redondeadas, a la alza o a la baja para mayor claridad ante el cliente.

En el requisito funcional número uno, se debería suprimir la palabra “por parejas”, ya que es redundante porque se comenta que las partidas serán 2 contra 2.

Para mayor claridad de los requisitos, han de organizarse por bloques.

El requisito funcional que hace referencia a los torneos debe explicar cómo serán los mismos para evitar ambigüedades.

Debido a que faltan horas para completar el mínimo de las mismas, se añade el requisito funcional de permitir espectadores en las partidas y que éstos puedan animar de alguna manera a los jugadores y también que los torneos se generen de forma automática y que no sólo los cree el administrador.

Los requisitos no funcionales 2, 3, 6 y 9 deben pasar a ser funcionales.

El requisito no funcional 1 debe explicarse mejor.

El requisito que explica el sistema de la tienda debe indicar si para comprar algo en ella se requerirá un esfuerzo por parte del jugador o las monedas para gastar se conseguirán con cierta facilidad.

Se comenta que faltan interfaces como puede ser la de login o la de configuración de perfil. En la interfaz de la partida faltan las cartas restantes y el triunfo.

En el diagrama de despliegue, interesan que se incluyan las tecnologías con las que se va a desarrollar cada parte del sistema.

Interesa la existencia de un diagrama de componentes para complementar al de despliegue, porque se incluyen dos servicios diferentes, el de la partida y el de la aplicación.

En la parte de la propuesta donde se informa acerca del personal no es necesario incluirlo. No interesa mencionar al equipo en concreto.

Se felicita por los logros como organización.

El coste total se debería redondear.

Una vez discutidos todos los aspectos de la propuesta técnica se para a comentar los aspectos de la siguiente entrega.

Se indica que, semanalmente, en Moodle se han de poner las horas que ha dedicado cada miembro del equipo a cada tarea.

Se comenta la técnica de “pair programming” como un aspecto a la hora de desarrollar el sistema. Éste hace referencia a la parte de control de proyecto y control de calidad.

Añadir un diagrama de Gantt, ya que es de utilidad saber cómo de evolucionada va cada tarea de la que se forma el proyecto. Debe existir una comparativa entre nivel de arquitectura y de clases. Al cliente le interesa más el de arquitectura, porque es más general debido a su abstracción. Además, se han de añadir los bloques del presupuesto.

Se indica que el control de la documentación es de gran importancia.

Los ficheros que tratan sobre documentación y que no son código, han de tener un espacio de nombres con significado y agruparlos convenientemente.

Se cierra la sesión a las 12:10 horas.

Acta de reunión 3 Grupo 4

Ubicación: Sala 2.11, edificio Ada Byron, EINA (Universidad Zaragoza)

Fecha: 20 / Marzo / 2018

Hora: 10:00h

Asistentes: Profesor: Javier Lacasta. Equipo: Marius Crisán, Víctor Soria, Javier Corbalán, Carlos Marañés, Sergio Izquierdo, Ignacio Bitrián, Julia Guerrero.

Desarrollo de la sesión

Se corrige el plan de gestión, análisis, diseño y memoria del proyecto previamente entregado. Además, se presentan a grandes rasgos los avances del proyecto hasta el momento para ver si cumplen con los plazos propuestos inicialmente y se analizan las principales dificultades encontradas.

Si un grupo sufre un retraso con una de las tareas asignada, nunca se debe añadir nuevo integrante al equipo ya que al nuevo integrante le costará mucho adaptarse al equipo y además puede entorpecer a los demás compañeros. La acción adecuada es reasignar tareas futuras de ese equipo a otro equipo para que puedan dedicar más tiempo a la tarea que se ha retrasado. (Libro de referencia: *Mythical mad month*).

Para la parte de comunicación entre jugadores es mejor utilizar comunicación asíncrona utilizando alguna técnica de desarrollo web como AJAX.

Todos los diagramas que se crearán a lo largo del proyecto se deben añadir en la parte de diseño.

La parte de la lógica del juego, partidas en curso, usuarios conectados es mejor no meterlo en la base de datos ya que se pueden gestionar desde el broker u otros componentes.

La base de datos se debe explicar en detalle en la entrega final del plan de gestión.

Asuntos pendientes

Rediseñar el diagrama de Gantt para ajustarlo al nuevo plan según los avances actuales y dividir los elementos que están en la primera y segunda iteración en dos para identificar mejor qué partes del proyecto serán entregadas en la primera iteración y cuales en la segunda.

Definir detalladamente todas las interfaces de los módulos de cada grupo que interaccionan con los de otros grupos para que cada grupo pueda trabajar de forma independiente.

Añadir las horas reales dedicadas a cada tarea del proyecto a la tabla de la oferta económica para poder comparar la estimación y los resultados al final el proyecto y obtener las conclusiones correspondientes.

Eliminar los integrantes de cada grupo que están en la parte de organización del proyecto, ya que están en el diagrama.

Reestructurar la parte 3.2.1 y 3.2.2 porque hay información repetida.

Especificar detalladamente cuando se utilizan pruebas de caja negra y cuando de caja blanca.

Hay que justificar porque se ha optado por una aplicación de tres capas y no hay que nombrar otras propuestas, como por ejemplo una aplicación móvil. Es decir, hay que valorar alternativas de diseño e implementación no nuevas propuestas. Por ejemplo, la alternativa de diseño de una base de datos con PostgreSql está bien pero hay que especificar en detalle cuáles han sido las por las que no se ha elegido. Otra alternativa es Oracle.

En el despliegue hay que cambiar bottom-up por top-down.

Eliminar del diagrama de componentes el Phaser porque es una librería y no se incluyen las librerías externas en este diagrama.

Pensar para la parte de middleware que versión es mejor: hacer una instancia o varias.

Buscar y utilizar un gestor de tareas para mejorar la compenetración y productividad de los equipos. Por ejemplo se puede utilizar el de Github o Trelog.

Corregir errores semánticos del plan de gestión. Hay que utilizar el presente en lugar del futuro (estará-> está, se decidirá -> se decide...). No hay que decir que se va a utilizar una aplicación con 3 capas por popularidad sino porque ofrece mucho soporte, se ajusta mejor a las necesidades...

En la parte de 3.1.1 especificar mejor cuales son los recursos necesarios para el desarrollo del objetivo de cada participante.

Añadir diagramas de la interfaz como por ejemplo un mapa de navegación.

Se cierra la sesión a las 10:55 horas.

Acta de reunión 4 Grupo 4

Ubicación: Sala 2.11, edificio Ada Byron, EINA (Universidad Zaragoza)

Fecha: 13 / Abril / 2018

Hora: 16:00h

Asistentes: Profesor: Javier Lacasta. Equipo: Marius Crisán, Víctor Soria, Javier Corbalán, Carlos Marañés, Sergio Izquierdo, Ignacio Bitrián, Julia Guerrero.

Desarrollo de la sesión

En primer lugar, se comenta cómo va el avance del proyecto en relación a las horas estimadas inicialmente. Se muestra al profesor el documento Excel en el que se lleva toda la contabilización de horas y este comenta que también existen otras herramientas específicas para ello.

Se explica que en general se había estimado relativamente bien, pero se había claramente subestimado tanto horas de gestión como horas de pruebas. Al respecto, el profesor comenta que quizás las horas de gestión están algo sobre-calculadas al estar incluyendo en ellas las horas de reunión con los profesores, que en un contexto real no se darían y que las pruebas unitarias deberían contar como horas de desarrollo y no de pruebas (pruebas solamente incluyen pruebas de integración y del sistema final). Las horas desarrolladas hasta ahora, con el final de la primera iteración suman alrededor de 350, es decir, la mitad de las horas de dedicación del proyecto.

En cuanto al diagrama de Gantt, es necesario alargar algo la tarea de la GUI de la partida, ya que quedan detalles por perfilar todavía, algo que sí estaba contemplado en los paquetes de trabajo.

A continuación, se hace una demostración al profesor de cómo funciona la partida uno contra uno, jugando entre dos ordenadores distintos, comprobando que todo funciona correctamente. También se muestran las vistas de login y perfil de usuario, así como la de la tienda, que está todavía en proceso de desarrollo.

Relativo a la GUI de la partida se comenta que todavía queda algo de trabajo de hacerla más “bonita” y añadir algunas animaciones. Además, es necesario establecer por escrito, en las reglas del Guiñote, que no es posible mirar las cartas de las bazas ya ganadas por un jugador, y decidir si estará o no visible en todo momento las puntuaciones de ambos jugadores, o solamente cuando se va de vueltas.

Se comentan al profesor los problemas encontrados con cómo probar correctamente la lógica del juego, ya que es muy difícil simular situaciones concretas y hay muchas horas de depuración. Se enseñan los diagramas de diseño, se destaca la necesidad de incluir un diagrama de secuencia (ya existe, pero no estaba incluido en el plan de gestión) y de algún tipo de diagrama para intentar controlar todas las casuísticas posibles en el desarrollo de las pruebas.

Finalmente, se habla del análisis de la Inteligencia Artificial, con la propuesta de utilizar un árbol expectimax y reglas para establecer las heurísticas. El profesor comenta que quizás no es necesario un expectimax siendo que todas las cartas poseen la misma probabilidad, pero se rebate que no todas las manos (formadas por 6 cartas, considerando dos cartas iguales si van a dar la misma

'puntuación' en la heurística) ya no tienen la misma probabilidad. También se comenta la posibilidad de utilizar redes neuronales, desechada por la falta de datos para entrenamiento.

Se concluye que en general el avance del proyecto es bueno.

Asuntos pendientes

Alargar la tarea de GUI de partida en el diagrama de Gantt y mantenerlo siempre actualizado.

Cambiar las horas de pruebas unitarias en la contabilización de horas por desarrollo. (las horas de las reuniones como gestión se pueden dejar pero siendo conscientes de que son contabilizadas)

Mejorar la depuración de la lógica (quizás con un diagrama de secuencia de lo que puede ocurrir en una partida).

Continuar con el proyecto como estaba establecido, incluido el diseño e implementación de la IA.

Se cierra la sesión a las 16:55 horas.

Acta de reunión 6 Grupo 4

Ubicación: Sala 2.11, edificio Ada Byron, EINA (Universidad Zaragoza)

Fecha: 29 / Mayo / 2018

Hora: 12:00h

Asistentes: Profesor: Javier Lacasta. Equipo: Marius Crisán, Víctor Soria, Javier Corbalán, Carlos Marañés, Sergio Izquierdo, Ignacio Bitrián, Julia Guerrero.

Desarrollo de la sesión

Se procede a realizar una demostración de las funcionalidades del software desarrollado. Se crea una cuenta nueva al profesor Javier Lacasta y se muestra a través de ella el funcionamiento de las partidas de dos jugadores, la expectación de partidas, partidas contra la IA, la tienda, el perfil de un jugador con el registro de partidas jugadas y la programación y funcionamiento de los torneos.

El profesor Javier Lacasta valora como exitosa la demostración presentada.

Se cierra la sesión a las 13:00 horas.

Anexo III. Presupuesto

| ESTIMACIÓN DE COSTES | | | | | Coste/hora uni | |
|--|---|----------|------------|------------------|----------------|--------------------|
| ESFUERZOS | | | | | COSTES | |
| Tarea/componente | Requisitos relacionados | Cantidad | Horas/item | Estimación final | Coste/hora | Coste (€) |
| Desarrollar el juego del guñote | Rf-01 | | | | | |
| Análisis juego | | 1 | 20 | 20 | 18,50 € | 370,00 € |
| Implementación java | | 1 | 40 | 40 | 18,50 € | 740,00 € |
| Implementación vista web | | 1 | 20 | 20 | 18,50 € | 370,00 € |
| Implementación interfaz | | 1 | 30 | 30 | 18,50 € | 555,00 € |
| Pruebas unitarias | | 1 | 18 | 18 | 18,50 € | 333,00 € |
| Diseño e Implementación vistas web | Rf-01-02-03-05-06-08-09-12-14-15-18-19-20 | | | | | |
| Diseño Pantallas | | 10 | 5 | 50 | 18,50 € | 925,00 € |
| Implementación Vista Web | | 10 | 10 | 100 | 18,50 € | 1.850,00 € |
| Desarrollar Base de Datos | Rf-02-03-04-05-07-08-09-10-11-12-13 -14-15-16-17-18-19-20 | | | | | |
| Diseño Base de Datos Relacional | | 1 | 15 | 15 | 18,50 € | 277,50 € |
| Implementación del modelo y carga de datos | | 1 | 20 | 20 | 18,50 € | 370,00 € |
| Implementación Capa Acceso Datos | | 9 | 4 | 36 | 18,50 € | 666,00 € |
| Diseño e implementación acciones periódicas | | 1 | 15 | 15 | 18,50 € | 277,50 € |
| Implementación Facade Base Datos | | 9 | 4 | 36 | 18,50 € | 666,00 € |
| Despliegue Base de Datos y Servidor Web | | | | | | |
| Despliegue AWS | | 1 | 10 | 10 | 18,50 € | 185,00 € |
| Implementación Web Dinámica | Rf-01-02-03-05-06-08-09-12-14-15-18-19-20 | | | | | |
| Implementación Servlets | | 9 | 5 | 45 | 18,50 € | 832,50 € |
| Implementación JSP | | 9 | 5 | 45 | 18,50 € | 832,50 € |
| Implementación MatchMaking | Rf-06-12 | | | | | |
| Algoritmo cálculo | | 1 | 20 | 20 | 18,50 € | 370,00 € |
| Algoritmo de emparejamiento | | 1 | 15 | 15 | 18,50 € | 277,50 € |
| Implementación IA | Rf-24 | | | | | |
| Analís del problema | | 1 | 15 | 15 | 18,50 € | 277,50 € |
| Representación del problema | | 1 | 35 | 35 | 18,50 € | 647,50 € |
| Implementación de clases | | 1 | 15 | 15 | 18,50 € | 277,50 € |
| Desarrollo de interfaz IA con el sistema | | 1 | 20 | 20 | 18,50 € | 370,00 € |
| Diseño de Artículos Personalizados | Rf-12-20-21-22-23 | | | | | |
| Añadir artículos a la tienda | | 1 | 3 | 3 | 18,50 € | 55,50 € |
| Implementación de la asignación de artículos | | 1 | 7 | 7 | 18,50 € | 129,50 € |
| TOTAL TAREAS/COMPONENTES | | | | 630 | 18,50 € | 11.655,00 € |
| Gestión | | 15% | | 94,50 | 18,50 € | 1.748,25 € |
| Gestión de configuraciones | | 5% | | 31,50 | 18,50 € | 582,75 € |
| Aseguramiento de la calidad | | 7% | | 44,10 | 18,50 € | 815,85 € |
| TOTAL MACROS | | | | 170,10 | 18,50 € | 3.146,85 € |
| Otros costes | | | | | | |
| Viajes | | 33 | 10,00 € | 330,00 € | | 330,00 € |
| Amortización equipos desarrollo | | 800,10 | 0,97 € | 777,88 € | | 777,88 € |
| TOTAL OTROS COSTES | | | | 1.107,88 € | | 1.107,88 € |
| TOTAL | | | | | | 15.909,73 € |

Anexo IV. Diagramas diseño

Referencias

- [1] Amazon. Amazon ec2. <https://aws.amazon.com/es/ec2/>. [Accessed 2018-02-22].
- [2] Amazon. Amazon web services. <https://aws.amazon.com/es/>. [Accessed 2018-02-22].
- [3] Amazon. Rds aurora. <https://aws.amazon.com/rds/aurora/>. [Accessed 2018-02-23].
- [4] F. P. Brooks, Jr. *The Mythical Man-month*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [5] G. Chaslot, M. Winands, H. Herik, J. Uiterwijk, and B. Bouzy. Progressive strategies for monte-carlo tree search. 04:343–357, 11 2008.
- [6] P. Cowling, E. Powley, and D. Whitehouse. Information set monte carlo tree search. 4:120–143, 06 2012.
- [7] D. W. Peter Cowling, Edward Powley. Ismcts.py. <http://www.aifactory.co.uk/newsletter/ISMCTS.txt>. [Accessed 2018-05-31].
- [8] R. C. Tausworthe. The work breakdown structure in software project management. *Journal of Systems and Software*, 1:181 – 186, 1979.