

RETROSPECTIVA DEL SPRINT 3

DEL 3 DE NOVIEMBRE AL 1 DE DICIEMBRE DE 2025

- Asignatura: **Diseño y Pruebas I (Grado en Ingeniería del Software, Universidad de Sevilla).**
 - Curso académico: **2025/2026.**
 - Grupo/Equipo: **L4-4.**
 - Nombre del proyecto: **Saboteur.**
 - Repositorio: <https://github.com/gii-is-DP1/dp1-2025-2026-l4-4-25>
 - **Integrantes (máx. 6):**
 - **Alejandro Caro Pérez (FQY7185 / alecarper@alum.us.es)**
 - **Lorenzo Valderrama Román (WRG8176 / lorvalrom@alum.us.es)**
 - **Diego Rey Carmona (RHQ7780 / diereycar@alum.us.es)**
 - **Marcos Ángel Ayala Blanco (GBK4935 / marayabla@alum.us.es)**
 - **Carlos Borrego Ortiz (HKP3295 / carborort@alum.us.es)**
 - **Luis Calderón Carmona (JGR9196/ luicalcar@alum.us.es)**
 - **Fecha y duración de la sesión:** La sesión tuvo lugar de forma online (mediante la plataforma *Discord*) el 3 de diciembre de 2025, durando aproximadamente dos horas.
 - **Metodología:** La metodología que usamos para redactar esta retrospectiva consistió en una reunión vía *Discord* con todos los participantes del grupo donde expusimos un resumen de todo lo implementado en el Sprint, los aspectos positivos, los aspectos a mejorar y las acciones que se tomarán para mejorarlos. Prácticamente, aplicamos una retrospectiva como tal, para después describirla formalmente.
-

1. Objetivos abordados en el Sprint 3

El objetivo principal de este Sprint 3 ha sido la implementación de al menos el 75% de las historias de usuario del documento de Análisis de Requisitos del sistema. También se ha trabajado en los siguientes documentos del proyecto: Documento de Diseño del Sistema (para añadir los nuevos patrones de diseño implementados), Documento del Plan de Pruebas (donde se detalla la obtención de un 80% de cobertura en las pruebas de *backend*) y el Documento de Uso de IA; todos ellos actualizados a fecha de finalización del Sprint 3.

Las tareas específicas realizadas se dividen en dos áreas principales:

- **Frontend:** Durante este Sprint, el desarrollo del *Frontend* se ha centrado en la materialización de la lógica (*core*) del juego y la integración completa del ecosistema social, logrando una experiencia de usuario fluida y sincronizada en tiempo real (gracias a la implementación del patrón *WebSocket*).
- **Tablero de Juego y Sincronización Real-Time (“Board.js”):** El avance más significativo ha sido la implementación completa de la clase “[Board.js](#)” (a falta de corregir bugs e implementar mejoras). Se ha desarrollado el sistema de reparto inicial de cartas (cálculo dinámico según el número de jugadores) y su renderizado en la mano del jugador. El tablero es ahora totalmente funcional, incorporando:
 - **Lógica de validación de movimientos:** Restricciones para la colocación de cartas de túnel (coherencia de caminos), mediante la implementación del algoritmo **BFS**, con modificaciones adaptadas a las reglas del juego, aunque necesita pulirse.
 - **Integración con WebSockets:** Se ha migrado y perfeccionado la comunicación implementada en el Sprint 2. Ahora, todas las acciones críticas (colocación de cartas, robo del mazo, cambios de turno y actualización de contadores de cartas) se sincronizan mediante el patrón Pub/Sub a través de canales públicos (o *topics*) de la partida, dichos canales se encuentran definidos de forma receptora en la pantalla “Board.js”. Esto garantiza que el estado del juego sea consistente para todos los jugadores conectados a una partida simultáneamente.

- **Cartas Especiales:** Implementación efectiva de las cartas de Acción (romper y reparar herramientas con actualización visual inmediata de los iconos de estado en tiempo real en todos los navegadores), cartas de Mapa (revelación temporal de objetivos para el usuario activo) y cartas de Derrumbe (destrucción de caminos existentes).
- **Experiencia de Usuario (UX) y Feedback Conscientes** de la latencia provocada por la carga de recursos de la partida y conexiones *WebSocket*, se ha implementado una Pantalla de Carga (“LoadingScreen.js”) al inicio de la partida. Esto asegura que todos los activos y estados (jugadores, chat, mazo, casillas...) estén listos antes de renderizar el tablero, evitando "parpadeos" o estados inconsistentes. Queda pendiente ajustar esta pantalla para que el tiempo de carga sea consistente para todos los jugadores y que permita mostrar el modal del rol que le ha tocado a cada uno, que actualmente queda solapado con la pantalla de carga. Adicionalmente, se ha integrado la lógica de fin de ronda en “GameEndLogic.js”, gestionando correctamente la finalización de rondas, la distribución de pepitas de oro y las transiciones entre partidas.
- **Comunicación y Trazabilidad Para mejorar la interacción dentro de la partida**, se han añadido dos componentes clave actualizados mediante *polling*:
 - **Log Global de Acciones:** Un registro visual accesible para todos los integrantes que narra cronológicamente los movimientos (quién ha jugado, qué carta, cambios de turno, etc.). También se actualiza en tiempo real para todos los jugadores.
 - **Chat:** Un sistema de mensajería en tiempo real que permite la comunicación estratégica entre los jugadores durante la sesión.
- **Módulo Social y Estadísticas** fuera del tablero, se ha enriquecido la aplicación con interfaces robustas para el ecosistema del jugador:
 - **Módulo Social Completo:** Desarrollo de funcionalidades para buscar usuarios, enviar solicitudes de amistad, listar las mismas y aceptarlas/rechazarlas. Además, se ha implementado la capacidad crítica de invitar a amigos directamente a una partida en espera de inicio (solo puede invitar el creador de dicha partida).
 - **Ranking y Estadísticas:** Implementación de las interfaces de *Ranking* global y visualización de *Stats* (estadísticas globales y personales)

dentro del perfil de usuario, permitiendo el seguimiento del rendimiento.

- **Gestión de Logros:** Interfaz CRUD completa para los administradores para gestionar *Achievements*.
 - **Navegación:** Incorporación de la *WelcomeScreen* como punto de entrada amigable a la plataforma.
-
- **Backend:** El desempeño de esta parte es fundamental, pues es la encargada de respaldar a la parte de *frontend* con los errores que puedan ir surgiendo debido a la integración *backend-frontend* y de responder a dudas generales sobre las partes que se hayan ido desarrollando (*Services, Controllers, Repositories...*).

Cabe destacar que el *backend* ha tomado un papel fundamental respecto a la implementación de la comunicación entre jugadores (para llevar a cabo la actualización simultánea en tiempo real) debido a que se han implementado los archivos necesarios en esta parte para poder gestionar correctamente el funcionamiento del *WebSocket*. Para empezar, en la carpeta “configuration” se ha añadido el archivo “WebSocketConfig.java” crucial para inicializar toda la lógica del *WebSocket*. Además, ha sido necesario implementar en la mayoría de controladores de las entidades relacionadas con la jugabilidad (*Game, Board, ActivePlayer, Deck, Round...*) el establecimiento del canal y del envío del mensaje por *WebSocket* y su respectivo *payload* (contenido del mensaje) que el *frontend* se encargará de gestionar. Esta acción se ha hecho siempre en los *PATCHs* de dichos controladores de las entidades, debido a que el *frontend* realizaba *PATCHs* al *backend* cada vez que se realizaba cualquier acción jugable dentro de “*board.js*”.

Además, se ha remodelado la clase “*Achievement.java*”, añadiendo tipo de logro y umbral para obtenerlo además de toda la lógica detrás de esta clase. También se han añadido nuevas entidades necesarias para conseguir que la partida se actualice correctamente, dichas entidades serían *Log*, entidad que guarda las acciones realizadas durante una partida y la entidad *Request*, que es necesaria para el módulo social, ya que indica una solicitud de amistad entre dos usuarios.

También señalar una implementación muy importante que se ha realizado durante el Sprint 3: La implementación del patrón *Builder* aprendido en clase para inicializar las rondas de las partidas. Durante este *sprint* determinamos que la lógica de implementación de partida se llevaría a cabo gracias a la entidad *Round*. Debido a la conexión de esta entidad con gran parte de entidades relacionadas con la partida, mediante las rondas podremos inicializar muchos componentes de la partida: el tablero (“*Board.java*”) con sus respectivas casillas (“*Square.java*”), el chat de juego (“*Chat.java*”) y el *log* (“*Log.java*”)... Además, debido a la relación de las rondas con

la entidad *Game*, conseguimos reunir todos estos componentes bajo un mismo “gameId”, esto nos será útil para el establecimiento de *topics* o canales del *WebSocket* y para meter a todos los jugadores en partida al mismo tiempo y en *real time* al pulsar el botón “START” de la pantalla *CreateGame* y realizando un *PATCH* a la entidad *Game* para que el “gameStatus” del juego esté en “ONGOING” y solo con esta condición se mande el mensaje por *WebSocket*. Para la implementación de este patrón que nos inicializa lo anteriormente comentado, se ha creado una carpeta llamada “builder” dentro de la carpeta “round”. En “builder” se definen las tres clases adaptadas a lo aprendido en clase: “AbstractRoundBuilder.java” (Clase abstracta con atributos generales), “RoundBuilder.java” (Interfaz), “StandardRoundBuilder.java” (clase que hereda de la clase abstracta). La lógica donde participan estas clases se lleva a cabo en la función “initializeRound” en el “RoundService.java” y dicha función del servicio es llamada en el *POST* del “RoundRestController.java” (función *create*) junto con un envío de mensaje del *WebSocket* para inicializar a todos los jugadores en la ronda.

Además, para verificar el correcto funcionamiento del Backend se han implementado clases de tests para todas las entidades del Backend. Habiendo realizado clases de tests para todas las clases de servicio y controlador, además de clases de repositorio para algunas entidades y clases de tests para verificar aspectos relevantes como *WebSocket* (relacionado con la entidad *Game*) y *JwtUtilsTests* (relacionado con la entidad *Configuration*).

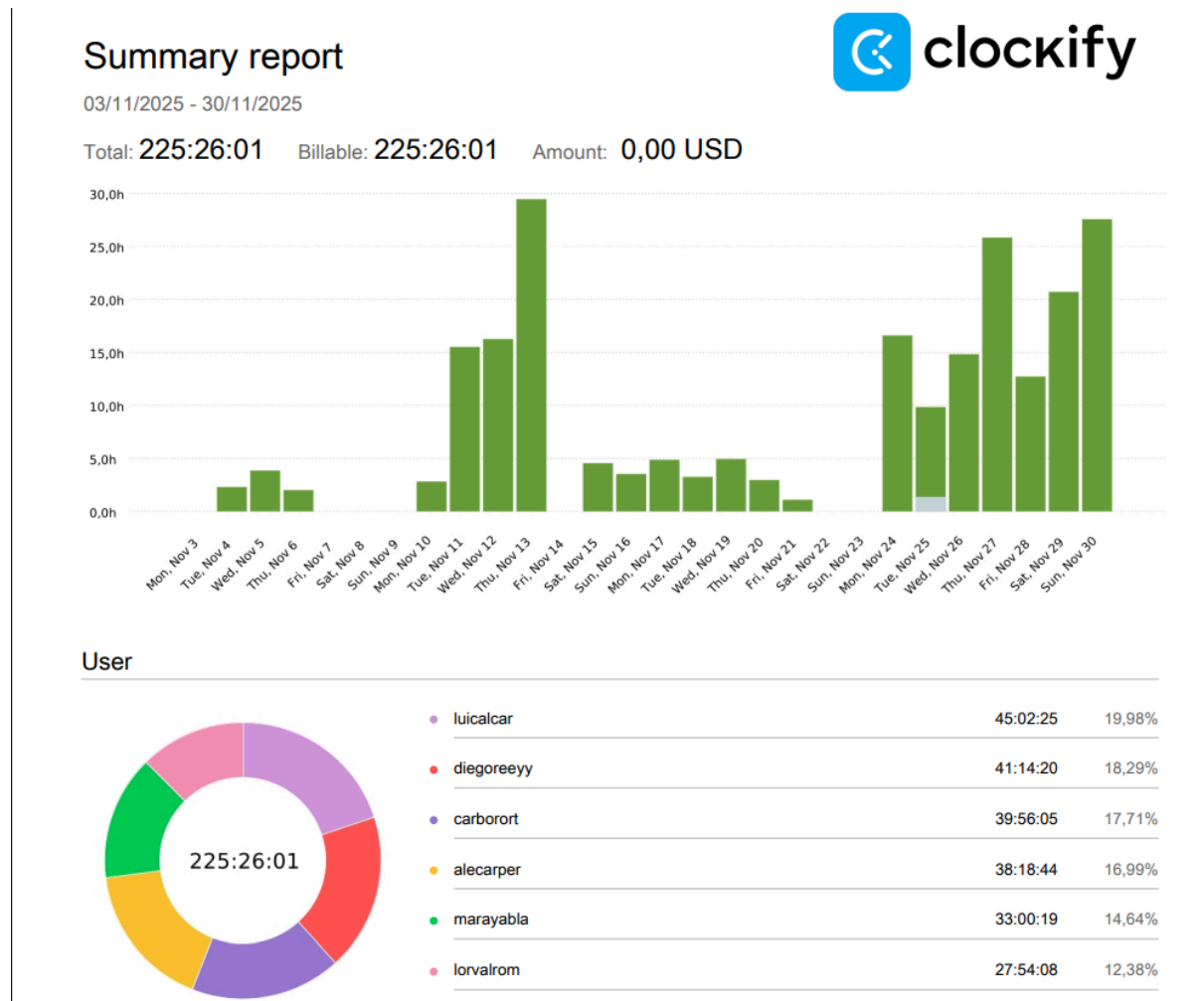
Se han implementado principalmente dos patrones de diseño para las clases de tests. Para las clases de tests de las clases de controlador de las entidades, se ha aplicado el patrón de diseño *Mock Object Pattern*, pues se pretendían clases de controlador de tipo solitarias, aislándose de la lógica de negocio y por tanto “mockeando” las clases servicio. En cuanto a las clases de servicio y de repositorio, se ha aplicado el patrón *Arrange-Act-Assert*, pues se plantearon tests sociables.

2. Aspectos positivos

- Se completó el modo espectador, se habilitó el invitar amigos a una partida y filtrar las partidas por amigos.
- Implementación dentro de “board.js” de: Lógica de finalización de ronda, pantalla de carga al tablero, pantalla de resultados de la partida, colocación de cartas y lógica de la misma, restricciones sobre las cartas en el tablero, algoritmo BFS, lógica sobre las cartas de acción y su uso en las herramientas, entre otras.
- Implementación del CRUD de Achievement.
- Implementación de las Estadísticas de un Usuario y globales.
- Implementación del Ranking de los jugadores.
- Implementación de pantalla de Bienvenida más amena y animada.
- Implementada la lógica sobre las rondas, creándose una después de la finalización de la otra, con el reparto de las pepitas y dando el rol ganador dentro de la misma.
- Implementación de la lógica sobre amistades de un usuario.
- Hubo una buena comunicación entre *Backend* y *Frontend*, lo que ha facilitado la resolución de errores que se daban en la aplicación y que se debían a la integración con el *backend* (además de la respuesta rápida a dichos errores y a otras dudas sobre la estructura de clases y herencias de los modelos)
- Cumplimos exitosamente todos los objetivos que fueron propuestos para el Sprint 3, incluso superando algunos que no estaban previstos para este (como se ha detallado antes, estos consistían en la implementación de al menos el 75% de historias de usuario y la elaboración de documentación y tests unitarios), llegando al 90% Historias de Usuario completadas.
- Hubo un gran trabajo entre todos los miembros del equipo. Trabajamos de una manera equitativa como se puede apreciar en la herramienta *Clockify*. Además, siempre fue imprescindible la organización y el compromiso, lo que consiguió que todos los participantes pudieran asistir a todas las reuniones agendadas.

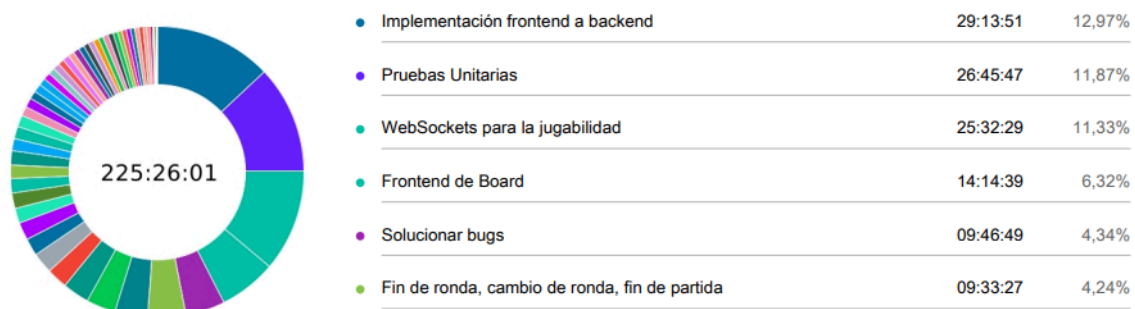
3. Tiempo invertido

- Estas son las horas dedicadas al proyecto en este Sprint:

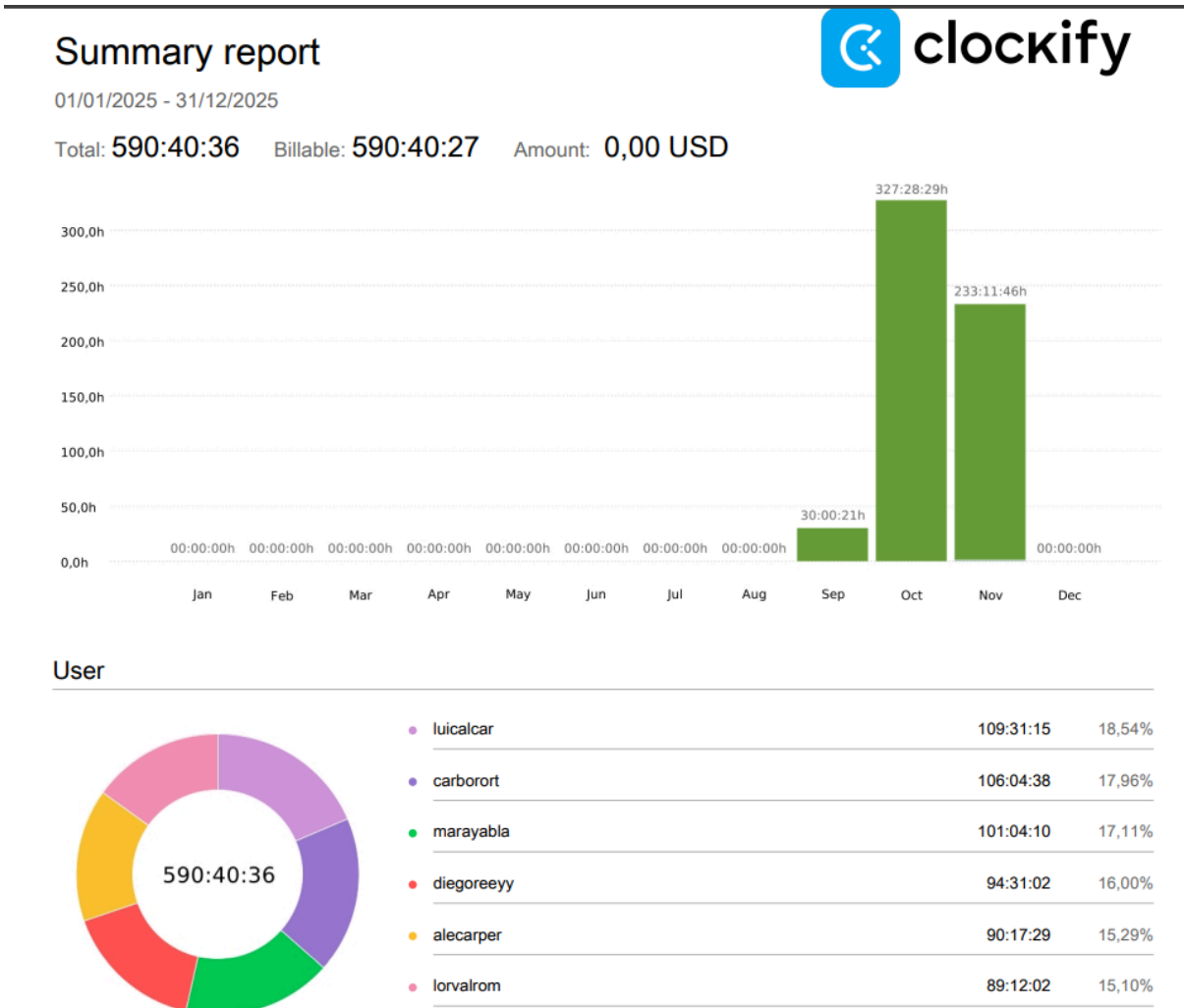


Tareas principales realizadas en este Sprint:

Description

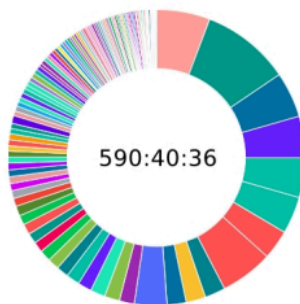


Reporte detallado de este Sprint [aquí](#). Este es el resumen del tiempo total dedicado al proyecto:



Actividades principales realizadas en el proyecto:

Description



● D2-Backend	34:42:53	5,88%
● D2 - BACKEND	57:06:54	9,67%
● Implementación frontend a backend	29:13:51	4,95%
● Pruebas Unitarias	26:45:47	4,53%
● WebSockets para la jugabilidad	25:32:29	4,32%
● Frontend de Board	24:08:34	4,09%

Universidad de Sevilla - DP1 Created with Clockify 1

● Frontend de HU-2 CREACIÓN DE PARTIDA	18:49:20	3,19%
● Documento de análisis de requisitos - Historia de usuario y Reglas de Negocio	34:04:32	5,77%
● Documento de análisis de requisitos - Historias de usuario y reglas de negocio	13:00:23	2,20%
● Frontend de HU-27 EDICIÓN DEL PERFIL PERSONAL	12:45:17	2,16%
● MD of Diseño y uso de IA	12:26:02	2,11%

Reporte detallado total [aquí](#).

4. Aspectos a mejorar

- **Caché de los navegadores:** Debemos buscar la manera en la que la caché de los navegadores no suponga un problema para el registro de los jugadores.
- **Velocidad de carga del tablero:** Debemos subsanar este problema ya que la velocidad de carga de los elementos del tablero, como son las cartas del mazo de cada jugador, es bastante elevada, se ha añadido una pantalla de carga para reducir este tiempo y que el jugador esté en una pantalla de carga, sin embargo, es algo a tener en cuenta. Se ha observado que en los navegadores como FireFox este error no ocurre.

- **Ordenar el [board.js](#):** Debemos ordenar módulos tales como el [board.js](#) para una correcta legibilidad ordenándose por constantes, funciones, etc.
- **Problema con los usuarios en data SQL:** Debemos solucionar el problema de los dos jugadores que se encuentran implementados al inicializar el *Backend*, ya que por sus múltiples relaciones dan algunos problemas a la hora de jugar una partida.
- **Eliminación de comentarios absurdos:** Actualmente, existen muchos comentarios innecesarios y de funciones comentadas que tienen que ser eliminadas, y algunas simplificadas según las normas y criterios de evaluación.
- **Revisión de código inútil:** Debemos comprobar en todos los módulos que el código que se ha implementado sea el óptimo y más optimizado, evitando código sucio y excesivo.
- **Tests de frontend:** Debemos realizar los Tests de *Frontend* que son obligatorios para obtener la máxima puntuación en el apartado de las pruebas unitarias del sistema.
- **Control de ramas de GitHub:** Debemos organizarnos mejor a la hora de saber cuál es la rama que tiene los últimos cambios y tener cuidado de actualizar ramas con errores en ellas.
- **Aumento de la cobertura del Sistema:** Aunque ya hay un alto nivel de cobertura de las pruebas unitarias, debemos mantenerlo y aumentar la cobertura de la misma obteniendo el máximo porcentaje total posible.
- **Visualización del tablero en distintos navegadores:** Debemos mejorar la visualización del tablero en los distintos navegadores, de forma que se vea de manera parecida en todos los navegadores.
- **Mejorar el BFS:** Mejorar las restricciones y la funcionalidad del BFS ya que actualmente en ciertas ocasiones no permite la colocación de una carta en una posición correcta del tablero y viceversa.

5. Acciones de mejora

Acción	Descripción	Responsable(s)	Plazo estimado
Corrección de bugs/errores	Debemos de dar solución a todos los errores/ bugs del juego.	Todo el equipo	Durante todo el Sprint 4
Ampliar cobertura de pruebas frontend	Crear pruebas frontend para poder aumentar la cobertura aún más	Diego, Luis	Mitad del Sprint 4
Poner datos reales y calcular las métricas	Crear y calcular estadísticas reales y relacionarlas con los jugadores de la partida	Lorenzo, Carlos, Diego	Inicios del Sprint 4
Hacer reuniones periódicas	Hacer reuniones periódicas con el fin	Todo el equipo	Durante todo el Sprint 4

	de mejorar la comunicación y organización		
Sincronización Front-Back	Establecer hitos conjuntos para evitar bloqueos entre ambos módulos.	Marcos, Luis, Alejandro	Durante todo el Sprint 4
Refactorización de código	Quitar comentarios, modularizar, reordenar por funcionalidad, etc	Marcos, Carlos	Mitad a final del Sprint 4

6. Conclusiones generales

Para concluir, el Sprint 3 ha supuesto un gran avance en el desarrollo de nuestro juego Saboteur. Motivado por un clima productivo de trabajo se ha conseguido superar la implementación del 90% de las *Historias de Usuario* planteadas en el primer Sprint, además de finalizar el desarrollo del *Backend* y obtener un 80% de su cobertura a partir de las clases de tests.

En cuanto a *Frontend*, los principales hitos han sido la implementación del tablero interactivo de la partida, consiguiendo que se actualice en tiempo real a todos los jugadores de la partida sin fallos gracias al *WebSocket*, así como la colocación de las cartas en una casilla con sus restricciones correspondientes, además del correcto funcionamiento del *Chat* y del *Log*.

Además, otros hitos importantes han sido la implementación del módulo social, la implementación de *Achievement* y *Ranking*, junto a *WebSocket* para la simultaneidad general.

Igualmente, se han detectado ciertos aspectos a mejorar relacionados principalmente con la acción de rotación de algunas cartas, el problema de la caché que no permite acceder a la interfaz de usuario de la pantalla “Register.js”, además de otros bugs previstos y futuros.

Destacamos la constante y cercana comunicación entre los miembros del equipo como principal baza para los avances realizados en nuestro Sprint 3. Se ha fomentado la productividad de los desarrolladores gracias al conveniente clima de trabajo y al constante feedback grupal.

Concluimos que el Sprint 3 ha significado un frenético y significativo avance, casi culminativo, en el desarrollo del juego de mesa Saboteur, destacando el trabajo y colaboración colectiva de los miembros del grupo.

