Impostor Game

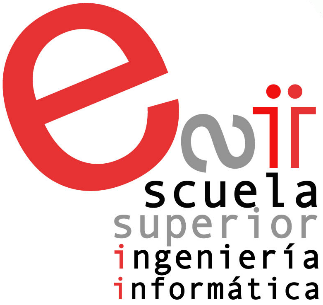
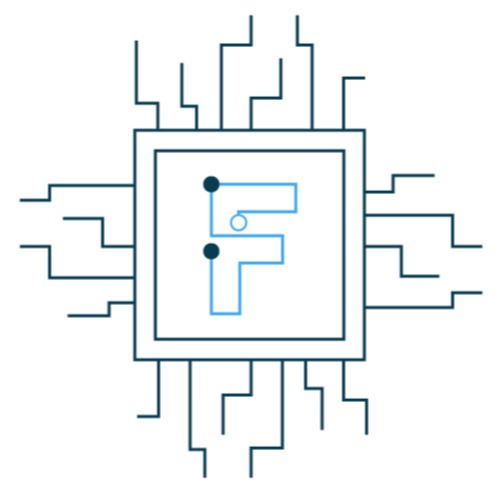
Sprint 1 Arquitectura lógica base y toma de contacto

Procesos de ingeniería software

2 de noviembre de 2020

Costa Tébar, Felipe

Felcot



**Abstract**: Incluye la representación del trabajo realizado durante el primer sprint. El objetivo por tanto del documento es esclarecer la arquitectura lógica desarrollada. Concluyendo, veremos cómo se ha adaptado el equipo de desarrollo en su primera toma de contacto con la herramienta kunagi.

**Keywords**: Sprint, Product backlog, historias de usuarios, Story Points.

Índice

[1. Introducción 1](#_Toc55490691)

[1.1 Reunión con los stakeholders 1](#_Toc55490692)

[1.2 Objetivos 2](#_Toc55490693)

[2. Investigación de aplicaciones parecidas 3](#_Toc55490694)

[2.1 Introducción 3](#_Toc55490695)

[2.2 Among Us 3](#_Toc55490696)

[2.3 Los hombres lobo de Castronegro 4](#_Toc55490697)

[2.4 Análisis de las aplicaciones de los competidores 4](#_Toc55490698)

[*3.* *Desarrollo del proyecto – Sprint 1 Arquitectura lógica base y toma de contacto* 6](#_Toc55490699)

[3.1 Product Owner e Historias de usuario 6](#_Toc55490700)

[3.1.1 Crear partida 6](#_Toc55490701)

[3.1.2 Unir a partida 6](#_Toc55490702)

[3.1.3 Iniciar partida 6](#_Toc55490703)

[3.1.4 Abandonar Partida 7](#_Toc55490704)

[3.1.5 Asignar tareas 7](#_Toc55490705)

[3.1.6 Asignar Impostor 7](#_Toc55490706)

[3.1.7 Evaluar Partida 7](#_Toc55490707)

[3.1.8 Atacar 7](#_Toc55490708)

[3.1.9 Votaciones 7](#_Toc55490709)

[3.2 Equipo y Tareas y horas estimadas 8](#_Toc55490710)

[3.2.1 Crear Partida 8](#_Toc55490711)

[3.2.2 Unir a Partida 8](#_Toc55490712)

[3.2.3 Iniciar Partida 9](#_Toc55490713)

[3.2.4 Abandonar Partida 9](#_Toc55490714)

[3.2.5 Asignar Tareas 9](#_Toc55490715)

[3.2.6 Aginar Impostor 9](#_Toc55490716)

[3.2.7 Evaluar Partida 10](#_Toc55490717)

[3.2.8 Atacar 10](#_Toc55490718)

[3.2.9 Votaciones 10](#_Toc55490719)

[3.3 Diseño de la base lógica 11](#_Toc55490720)

[3.3.1 Arquitectura cliente servidor 11](#_Toc55490721)

[3.3.2 Arquitectura base del juego 11](#_Toc55490722)

[3.3.3 Diagramas de secuencia y explicación del código 14](#_Toc55490723)

[3.3.3.1 Crear Partida 14](#_Toc55490724)

[3.3.3.2 Unir a Partida 16](#_Toc55490725)

[3.3.3.3 iniciarPartida 18](#_Toc55490726)

[3.4 Reunión al final del sprint 19](#_Toc55490727)

[3.4.1 Producto onwer 19](#_Toc55490728)

[3.4.2 Retrospectiva del equipo 19](#_Toc55490729)

[Bibliografía 21](#_Toc55490730)

Tabla de figuras

[Figura 1: Arquitectura Cliente-Servidor, primer Sprint. 11](#_Toc55490731)

[Figura 2: Arquitectura Cliente-Servidor, final del primer Sprint. 11](#_Toc55490732)

[Figura 3: Diagrama de clases, primera arquitectura base. 12](#_Toc55490733)

[Figura 4: Arquitectura final del sprint 1 13](#_Toc55490734)

[Figura 5: Tareas con TodoIst 20](#_Toc55490735)

Tabla de Diagramas

[Diagrama de secuencia 1: Crear Partida 14](#_Toc55490736)

[Diagrama de secuencia 2: Unir a Partida 16](#_Toc55490737)

[Diagrama de secuencia 3: iniciarPartida 18](#_Toc55490738)

[Diagrama 1: BurnDown 19](#_Toc55490753)

Tabla de Código

[Código 1: crearPartida usuario 14](#_Toc55490739)

[Código 2: crearPartida Juego 15](#_Toc55490740)

[Código 3: obtenerCodigo Juego 15](#_Toc55490741)

[Código 4: unirAPartida – Juego 17](#_Toc55490742)

[Código 5:agrearUsuario & puedeAgregarUsuario – Partida 17](#_Toc55490743)

[Código 6: agregarUsuario - fase inicial 17](#_Toc55490744)

[Código 7: agregarUsuario - fase completado 18](#_Toc55490745)

[Código 8: agregarUsuario - fase jugando 18](#_Toc55490746)

[Código 9: agregarUsuario - fase final 18](#_Toc55490747)

Tabla de Código Test

[Código test 1: Crear partida 16](#_Toc55490748)

Capítulo 1

# Introducción

Hemos recibido una petición de proyecto, los stakeholders quieren realizar un juego, según han mencionado el juego se llamara, Impostor Game, basado en el popular juego Among us¸ que a su vez está basado en el juego coloquialmente conocido como El pueblo duerme, en el apartado de la reunión esta todo más detallado.

Por tanto, este documento se encargará de recoger la descripción que delimitará el proyecto. Además, también recoge el proceso realizado en el primer sprint, se incluirá también la arquitectura que se va a emplear, así como las diferentes herramientas y su utilización.

Por otro lado, tras la reunión con los stakeholders se acordaron una serie de historias de usuario que han gestado el código fuente, que estará adjunto al documento, sin embargo, este documento contiene una descripción detallada de los bloques de código, además de sus diagramas de secuencia o actividad en su caso.

Para continuar, se detallará la finalidad con la que se han utilizado las pruebas en *Jasmine*, y porque se han utilizado estas. Y en su caso cuales deberán ser implementadas en un futuro cercano.

Para finalizar, se verá cual ha sido mi problema de trabajar en kunagi, y que por tanto, han generado el Anexo C:- Contrato con kunagi, para mejorar la gestión.

## Reunión con los stakeholders

Durante la reunión con los stakeholders nos comunicaron que estaban enganchados al juego *Among us*, y que querían realizar su propia versión del juego en una plataforma web, para ello creen que es preciso realizar una investigación previa que esta recogida en el *capítulo 2* de este mismo documento. Además, los stakeholders han establecido diferentes roles y como ingenieros que desarrollaran la petición de estos clientes, hemos definido las peticiones que desea cumplir el sistema. Más adelante se verá durante el *capítulo 3* las historias de usuario recogidas en esta primera reunión, pero por ahora, enumerar algunas de las peticiones, que nos han solicitado, que el juego disponga de la creación de partidas, de la operación de votar, matar, e iniciar partida. Así como, el resto de las funciones que veremos en el *capítulo 3.*

Por otro lado, se nos ha pedido que demos parte de nuestro progreso y es con esta finalidad, que este documento es creado, se ha determinado el siguiente objetivo para llegar a la meta del primer Sprint.

## Objetivos

El **objetivo principal**, consiste en el desarrollo de un videojuego denominado *Impostor Game*, centrandonos en el primer sprint que abarca del miércoles 07 de octubre del año 2020 al jueves 22 del mismo mes y año, en el que se busca la implementación de la lógica básica del juego. Por tanto, para lograr llegar a la meta, se han detallado los siguientes **objetivos específicos**:

* Investigar sobre los competidores directos, *Among us o El pueblo duerme.*
* Investigar y aprender a utilizar las tecnologías que usaremos para la implementación.
* Diseñar la arquitectura del juego.
* Implementar la arquitectura, incluyendo los mecanismos para comprobar su correcto funcionamiento.
* Generar la documentación necesaria para vislumbrar el funcionamiento de la aplicación.
* Validar las pruebas generadas.

Con el fin de cumplir el objetivo, veremos una serie de calificaciones que nos permitirán comprender si se ha logrado superar, además se deberán cumplir las pruebas de *Jasmine.* A su vez, tendremos en el anexo A:- La documentación de Kunagi asociada al primer sprint.

Capítulo 2

# Investigación de aplicaciones parecidas

## Introducción

En la actualidad, existe un juego multiplataforma, denominado *Among us* y otro ajeno a la red denominado *El pueblo duerme,* o también conocido como *los hombres lobo de Castronegro*. Los cuales son los competidores directos de *Impostor game*. El beneficio que causa este capítulo en el proyecto puede ser determinante para el desarrollo de la aplicación pues permitirá centrar las ideas rigiendo las particularidades de cada juego, permitiendo crear en un futuro nuestra propia identidad.

Este juego ha sido adaptado en innumerables ocasiones desde el año 1986 que fue creada la primera versión conocida como *Mafia* por *Dimitry Davidoff* creada en el departamento de Psicología de la universidad Estatal de Moscú. Este juego consistía en dotar de información a los denominados “*mafiosos*”, y por otro lado, los ciudadano que solo conocen que existen los mafiosos pero no quienes son, teniendo que votar para eliminarlos ([1] Migdal, 2010) ([2] Wikipedia - Mafia Game, 2020).

## Among Us

Como mencionaba, este juego multiplataforma surgió en el año 2018, sin embargo, no fue hasta la fatídica cuarentena del año 2020 donde algunas personas con cierta influencia en la población más joven comenzaron a jugarlo.

Este funciona de tal manera que un conjunto de 4 a 10 jugadores entre los cuales puede existir de 1 a 3 jugares que ejercen el rol de impostor. Estos últimos se encargan de causar terror entre los tripulantes, pues el entorno del juego es una nave espacial. Este terror, lo causarán segando la vida del resto de tripulantes, además de sabotear las funciones vitales de la nave, como pueden ser el oxígeno, las comunicaciones o el mismísimo reactor que mueve la nave.

Por otro lado, los tripulantes deberán realizar una serie de tareas, que permiten avanzar en el proceso de la misión espacial, algunas son tan simples como tirar la basura, pero otra puede ser realizar un chequeo médico, o detectar viales en mal estado.

Por último, durante el juego, si se encuentra algún cuerpo o se sospecha que alguien no realiza sus tareas, siguiente el dogma espacial, los tripulantes tendrán la opción de reportar el caso. Acusando con o sin pruebas a otro tripulante. De tal manera, que por acuerdo el más votado será eyectado de la nave, sin embargo, en caso de sufrir algún empate en la votación esta terminará inconclusa.

Los tripulantes que han fenecido podrán pasar al otro lado, solo cuando sus tareas de ese día hayan sido completadas. Y en caso de que un impostor muera, este se verá evocado a sabotear la nave. De tal manera, que solo se alcanza el estado de victoria en dos ocasiones al igual que el estado de derrota. Es decir, solo podrán ganar los impostores si consiguen matar a toda la tripulación o bien ganará la tripulación si pillan a los impostores o terminan las tareas de ese día ([3] Lugris, 2020).

## Los hombres lobo de Castronegro

Fue creado por Dimitry Davidoff, Philippe des Palliéres y Hervé Marly. Estes diseñaron el juego de mesa con este mismo nombre inspirándose en el juego *Werewolf* de 1986, el cual fue simplemente el rediseño del juego *Mafia* ([4] Wikipedia - Pueblo duerme, 2020)*.*

En este juego se necesitan un mínimo de 5 jugadores, que toman diferentes roles, el más importante es el de Narrador, encargado de mediar en el debate entre los jugadores, luego tenemos el rol de hombre lobo se encarga de matar a todos los jugadores. Otro de los roles es el de bruja, encargada de la magia de la aldea es capaz de matar o revivir a un ciudadano del pueblo utilizando sus brebajes. Otro importante rol, aunque deberá alegar intuición pues está mal visto ser vidente en los tiempos en los que se desarrolla el juego, podrá conocer antes del amanecer el rol de cada jugador. Y, por último, tenemos aquel que sin forma de defenderse solo dispone de su intuición y su voto, para poder sentenciar a pena capital a un ciudadano, esperando que sea el hombre lobo.

Como se puede intuir del párrafo anterior, el juego se compone de dos fases, diurna y nocturna, en la primera los ciudadanos tendrán una asamblea diaria en la cual podrán realizar una votación, tras debatir entre ellos quien puede ser el hombre lobo. Sin embargo, en la fase nocturna, los hombres lobo actúan primero, y tendrán la posibilidad de matar a un ciudadano. Aunque el narrador antes de cambiar de fase tendrá que llamar a los jugares que tienen algún rol especial, es decir, a la bruja para indicarle si desea o no salvar al ciudadano que haya muerto, o si por el contrario quiere perpetuar el terror matando a alguien. Además, tenemos al vidente que será conocedor del rol de un jugador que este decida ([4] Wikipedia - Pueblo duerme, 2020).

## Análisis de las aplicaciones de los competidores

Simplemente con la descripción de los juegos vemos muchas similitudes, cierto es que *los hombres lobo de Castronegro* es el juego más fiel al original denominado *Mafia*, también es cierto que participo en la creación de este el creador de *Mafia*. Sin embargo, monetariamente hablando, el juego que más atrae beneficios es *Among Us* pues tiene un sistema de skins que es el motor financiero de la aplicación. Sin embargo, el juego del pueblo duerme, cuenta con un juego de mesa y los usuarios más nostálgicos prefieren de este. Pero como deseamos crear una aplicación basada en el juego de *Among us,* nos centraremos en este. ¿Como funciona?, ¿de qué se quejan los jugadores?, o ¿cómo hacer rentable un juego parecido a este? serán preguntas que tendremos que solucionar en un futuro cercano.

Por ahora nos centramos en la pregunta ¿Cómo funciona? Dispone de una arquitectura cliente servidor, donde los clientes interaccionan en un escenario que se lleva a cabo la partida. Comencemos por ahí entonces para el primer sprint. Como creamos una partida, como se unen los jugadores o como pueden matarse entre ellos dentro del juego, son las respuestas que resolveremos en el capítulo 3.

Capítulo 3

# *Desarrollo del proyecto – Sprint 1 Arquitectura lógica base y toma de contacto*

Tras la reunión con los stakeholders el equipo se dispuso a esclarecer como funcionaban las aplicaciones permitiendo así entrar en el domino de la aplicación que se deseaba desarrollar engendrando el *capítulo 2,* por otro lado, se dispuso a realizar la primera reunión ya con finalidad de realizar el primer sprint. Para ello el ***Product Owner*** detectó que el objetivo de este sprint seria implementar la lógica básica como se menciona en el *capítulo 1,* para ello se convocó a los stakeholders para desarrollar las siguientes historias de usuario.

## Product Owner e Historias de usuario

Durante la reunión con los stakeholders, el ***Product Owner*** comunico que en el primer sprint tendríamos que generar la lógica base del juego, para ello, solicito a los stakeholders que realizarán una descripción detallada de las funcionalidades deseadas, dando lugar a las historias de usuario.

La información de las historias de usuario que viene a continuación está recogida en el Anexo A : Documentación de Kunagi.

### Crear partida

En esta historia de usuario se pedía que, “*como usuario, quiero poder crear una partida de modo que el sistema le asigne un código*”. Es decir, permite la creación y por tanto identificación inequívoca de las diferentes partidas.

### Unir a partida

En la siguiente historia de usuario los stakeholders solicitaban que, *“Como usuario, quiero unirme a una partida según un código que me pasan.”* Por tanto, se detectó que debíamos implementar un diseño de la funcionalidad unir a partida además de las pruebas asociadas a este, además como veremos cuando rediseñamos la primera arquitectura que se pretendía desarrollar, nos dimos cuenta de que tendríamos que rediseñar esta funcionalidad que ya habíamos implementado, por lo que añadimos una tarea de revisión del diseño.

### Iniciar partida

Los stakeholders nos comunicaron que las partidas tendrían un propietario, que seria el creador de esta, y que debía poder iniciar la partida cumpliendo que como mínimo existían 4 jugadores. Por ello se estableció la siguiente descripción “Como propietario, quiero poder iniciar la partida en cualquier momento. Si existen 4 usuarios registrados.” Para ello se tenia que revisar la arquitectura diseñada, y seguidamente implementar y establecer las pruebas que son necesarias.

Por otro lado, como desarrolladores debíamos tener en cuenta que las pruebas serian cumplidas comprobando que se había cambiado de fase correctamente y se habían establecido correctamente el rol de impostor y las tareas de los usuarios. Además, se debía de comprobar que en la fase inicial saltaba la excepción N410C, indicando que en la fase inicial no se puede iniciar una partida. Anexo B - Excepciones.

### Abandonar Partida

Se solicito que los usuarios debían poder salir de una partida en cualquier momento. Por lo tanto, se anotó “*como usuario, quiero poder abandonar la partida en cualquier momento*”. Detectando las tareas comunes a las anteriores historias diseño – refactor, implementación y pruebas.

### Asignar tareas

Se nos comunico que los jugadores debían tener una serie de tareas, por lo tanto, se decidió que “Como sistema, tengo que asignar un conjunto de tareas a un grupo de usuarios.”.

### Asignar Impostor

También se solicitaba que existiera un jugador que se denominaba impostor por ello, se definió que “El sistema, asigna el rol de impostor a un usuario. El usuario que matar al resto de usuarios.” De esta por tanto se deduce que tendríamos que desarrollar la historia de usuario Atacar, que veremos más adelante.

### Evaluar Partida

Además, los stakeholders nos pidieron que las partidas debían tener un final y por que conjunto de usuarios era alcanzada la victoria. Para ello “El sistema debe ser capaz de detectar si la partida ha terminado y que conjunto de usuarios es el ganador.”

### Atacar

Como se mencionaba en la historia de usuarios “*Asignar impostor*” existe un rol denominado impostor el cual debe ser capar de atacar a un usuario vivo y que su estado pase a muerto. Por ello se establece que “*Como impostor, quiero poder atacar al resto de usuarios.*”.

### Votaciones

Esta historia de usuario viene definida porque según los stakeholders el juego necesita que los usuarios puedan votarse entre ellos, con el fin de detectar quien es el impostor. Para activar la votación debía ser utilizado un botón de report o bien reportar un cuerpo. De tal manera que “*Como usuario, quiero poder activar el botón de emergencia o me encuentro un cadáver quiero poder reportar la información, durante la votación. Los usuarios pueden votar a un usuario sospechoso o pueden saltar la votación.*” Además, se solicitó también si era posible para este sprint incorporar un chat “Si os da la vida incorporar un chat para el debate.”.

## Equipo y Tareas y horas estimadas

### Crear Partida

Para la resolución de esta historia el equipo acordó las siguientes tareas.

* Elegir un diseño básico de la solución (el cual viene recogido en el apartido de diseño de la base lógica).
* Tutorial de JQuery
* Tutorial Node.js
* Tutorial de HTML y JavaScript
* Implementar crear partida
* Realizar pruebas

Como vemos al ser la primera historia de usuario debíamos entrar en el dominio de la tecnología que deberíamos utilizar. Como querían una plataforma web, los desarrolladores tenían que refrescar su conocimiento de HTML y se optó por JavaScript porque ya lo había utilizado el equipo con anterioridad por ello simplemente se utilizaron tutoriales y una serie de Quiz. Además, ya teníamos constancia que se utilizaría una arquitectura Cliente-Servidor por ello incluimos Node.js y JQuery con el fin de lograr que el equipo lo fuera aprendiendo poco a poco para migrar los avances al servidor.

Esta historia de usuario quedaría resulta cuando todos los quiz queden resueltos y además deberán resolverse las pruebas relacionadas con crear partida, con la tecnología J*asmine*. La historia de usuario se estima que tendrá 2 Story Point desde ahora SP, es decir, teniendo en cuenta que cada medio SP se entenderá estimada 1 hora para esta historia de usuario tendríamos que utilizar 4 horas. En kunagi no se pueden poner medias horas, por lo que se deja registrada 1 hora para los tutoriales de HTML y JavaScript, pero dejó constancia que solo se ha empleado una hora en total para ambos tutoriales, por conocimiento previo de las herramientas. En el Anexo A, tenemos recogida en la primera página esta historia de usuario*.*

Finalmente se emplearon para finalizar esta historia de usuario un total de 8 horas, prácticamente el doble de lo estimado. Ese tiempo extra se empleó en la adquisición del conocimiento de JQuery y Node.js.

### Unir a Partida

Las tareas asignadas a esta historia son:

* Implementar diseño, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar el diseño “*Comprobar que todo concuerda con el diagrama de secuencia*”.

Como se observa, para realizar esta historia el equipo ya con los conocimientos previos de HTML y JavaScript, podía empezar a trabajar mucho más rápido, sin embargo, se estimó que tendría una complejidad de 1 SP, equivaliendo a un total de 3 horas una por cada tarea.

Esta estimación si se ajusto a la realidad pues se tardo aproximadamente 2 horas y 45 minutos, para la realización, pues se empleo 1 hora y media en las pruebas de *Jasmine*, 45 minutos en la implementación, y finalmente 30 minutos en la revisión. Si kunagi hubiera permitido establecer intervalos de 30 minutos, posiblemente se ajustaría mucho mejor la estimación de cada tarea.

### Iniciar Partida

Las tareas asignadas a esta historia son:

* Implementa, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar“*Comprobar que todo concuerda con el diagrama de secuencia*”.

Para realizar esta historia el equipo estimó que tendría una complejidad de 2 SP. Esta estimación se ajusto bastante pues realmente finalizo con una complejidad de 3 SP, aunque no se reestimo y se mantuvieron los 2 SP. Esto se debe a que realmente no podríamos hacer las pruebas hasta no tener el diccionario de excepciones. Pero la historia estaba resulta en la estimación prevista a excepción del punto relacionado con el objeto Exception.

### Abandonar Partida

Las tareas asignadas a esta historia son:

* Implementar, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar“*Comprobar que todo concuerda con el diagrama de secuencia*”.

Para realizar esta historia el equipo estimó que tendría una complejidad de 0.5 SP, ya que simplemente debía ser eliminado el usuario del conjunto de usuarios que componen una partida.

### Asignar Tareas

Las tareas asignadas a esta historia son:

* Implementar, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar “*Comprobar que todo concuerda con el diagrama de secuencia*”.

Se estimo en un principio 2SP, sin embargo, el esfuerzo fue de tan solo 1 SP. Ya que fue bastante sencillo implementar una clase Encargo que recoge los encargos y se encargaba de suministrar uno aleatorio. Además, que la función alojada en partida solo tenia que llamar al objeto que se encargaba de esta tarea pasándole los usuarios.

### Aginar Impostor

Las tareas asignadas a esta historia son:

* Implementar, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar “*Comprobar que todo concuerda con el diagrama de secuencia*”.

Se estimo en un principio 1SP, al igual que Asignar Tareas se sobre estimo, pues se estableció que al final se establecería un impostor de forma aleatoria. Dejando registrado en el propio usuario una variable denominada *impostor* a true o false, para determinar si era o no un usuario.

### Evaluar Partida

Esta historia de usuario tiene asignadas las tareas:

* Implementar, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar “*Comprobar que todo concuerda con el diagrama de secuencia*”.

Se estimo en un principio 3SP, sin embargo el esfuerzo fue superior pues el equipo tuvo que investigar sobre el patrón *observer* y el patrón *mediator*. Pues se propuso implementar la clase *contenedor* que perseguía centralizar las actualizaciones de usuarios, con el fin de que se encargara de avisar a la partida que deba actualizarse.

### Atacar

Para la historia de usuario Atacar se estimaron 3SP, siendo las tareas por realizar:

* Implementar, “*A partir del diagrama de secuencia asociado a esta tarea implementar el código*”.
* Pruebas “*Pruebas asociadas en Jasmine*”.
* Revisar “*Comprobar que todo concuerda con el diagrama de secuencia*”.

Los SP estimados en esta historia fueron acordes a la situación pues las pruebas se llevaron la mayor parte del esfuerzo. Siendo la revisión y la implementación las partes mas sencillas, ya que el equipo estaba familiarizado y rápidamente tendrían el diagrama de secuencia de la funcionalidad matar.

### Votaciones

La historia de usuario, votaciones se estimo en 13SP pues aparentaba dificultad, sin embargo, como ya había sido implementada una forma de evaluar la partida, pudieron ser reutilizados bloques de código. Permitiendo que pasara a una estimación de 8SP quizás menos, esto no queda reflejado en kunagi pues no se reestimo la historia de usuario.

Las tareas que se asociaron fueron:

* Implementar, “A partir del diagrama de secuencia asociado a esta tarea implementar el código”.
* Pruebas “Pruebas asociadas en Jasmine”.
* Diseño “Generar un diagrama que permita conocer”.

## Diseño de la base lógica

### Arquitectura cliente servidor

Como se menciona al final del capítulo 2, la arquitectura utilizada por *Amoung us* es la arquitectura cliente servidor, esta se basa en la realización de diversas peticiones desde un cliente a un servidor, sin embargo, también se intuye que el servidor va a realizar peticiones Callback de tal manera que tendremos que resolver peticiones en ambos sentidos. Como vemos en la *figura 1* para el primer sprint desarrollamos la aplicación en el cliente, sin embargo, en la *figura 2* vemos que el desarrollo queda migrado al servidor, cuando hemos terminado el *sprint* como preparación para el siguiente *sprint*.

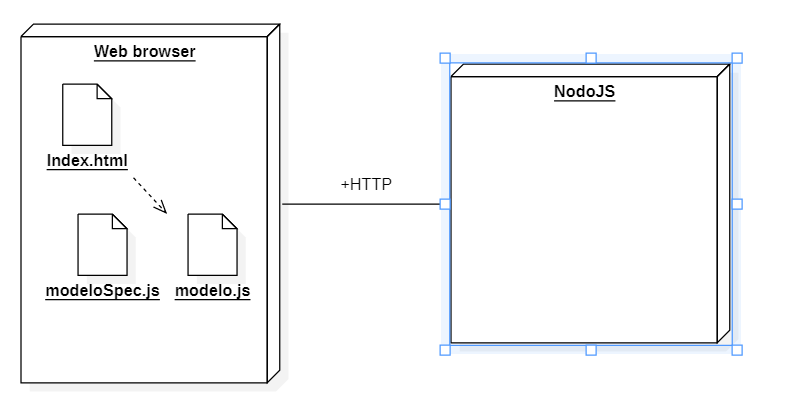


Figura : Arquitectura Cliente-Servidor, primer Sprint.

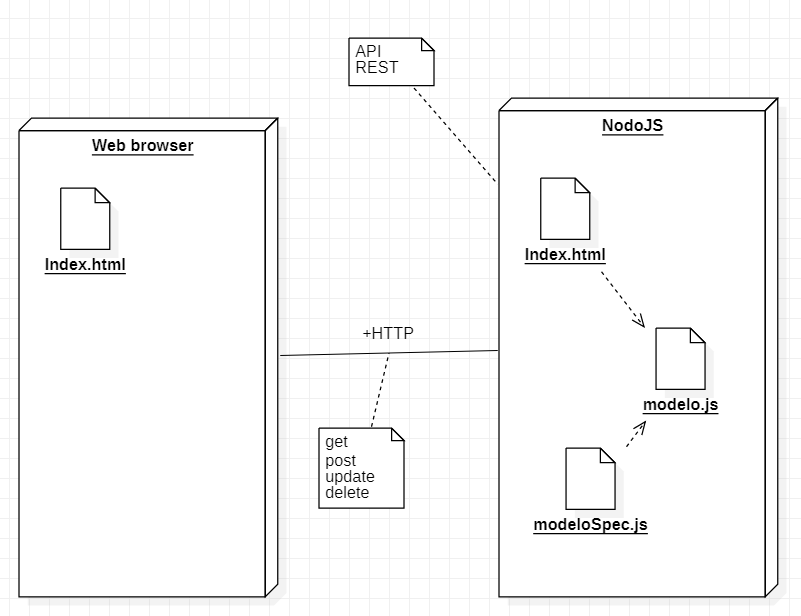


Figura : Arquitectura Cliente-Servidor, final del primer Sprint.

### Arquitectura base del juego

Al principio determinamos una arquitectura previa recogida en la *figura 3,* como se observa en la figura mencionada, tenemos una serie de funciones que a priori se han detectado y que serán necesarias de implementar. Algunas de estas son la creación de partidas o por ejemplo utilizando el patrón *State* las fases del juego. Que se decidió utilizar para delegar en estas algunas funciones que solo pueden ser realizar dependiendo de la fase del juego. Este patrón según la banda de los cuatro:” *Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Pareciendo así que cambie la clase del objeto*” ([5] Gamma, Helm, Johnson, & Vissides, 1994) por lo tanto es el patrón que se ajusta como un guante a la necesidad del juego.

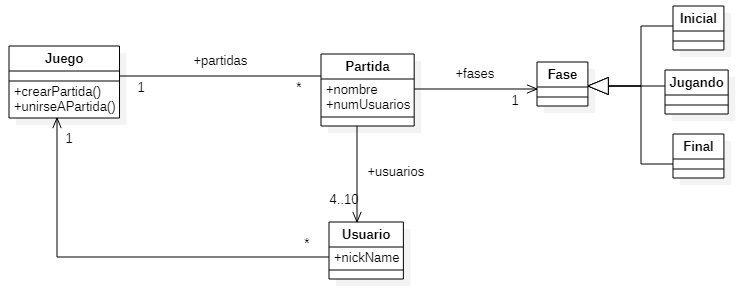


Figura : Diagrama de clases, primera arquitectura base.

Sin embargo, a pesar de tener una buena arquitectura, nos vimos evocador a realizar un rediseño, para ello podemos fijarnos en la *figura 4,* donde vemos se ha añadido utilizando de nuevo el patrón *State* cuya super clase es *Estado.* Por otro lado, se definieron todas las funcionalidades que abarcaría cada objeto. Y se contempló añadir el denominado *Contenedor* algo parecido a un *observer* que se encarga de avisarle al objeto *Partida* el cual actúa como *Mediator*, de que algún usuario del conjunto de jugadores a sufrido una modificación y la partida debe ser reevaluada, por sí esta está debe cambiar a la fase final, donde se indica que conjunto de jugadores es el ganador.

Además, con el fin de tener un espacio dedicado en exclusiva a los mensajes que emite la aplicación por consola, se generó la clase *Exception*, esta recoge un diccionario de errores, a los que se les ha asociado un nombre, formado de tal manera que la primera letra corresponde al nombre de la clase donde se genera el error, y la o las siguientes corresponden al comportamiento que genera el error, si bien, pueden ocurrir repeticiones que se solventa añadiendo dos números.

En la arquitectura, no se recoge la función de acceso general randomInt, esta función permite generar números enteros aleatorios comprendidos entre un mínimo y un máximo *ver Anexo B – Excepciones y otras funciones*.

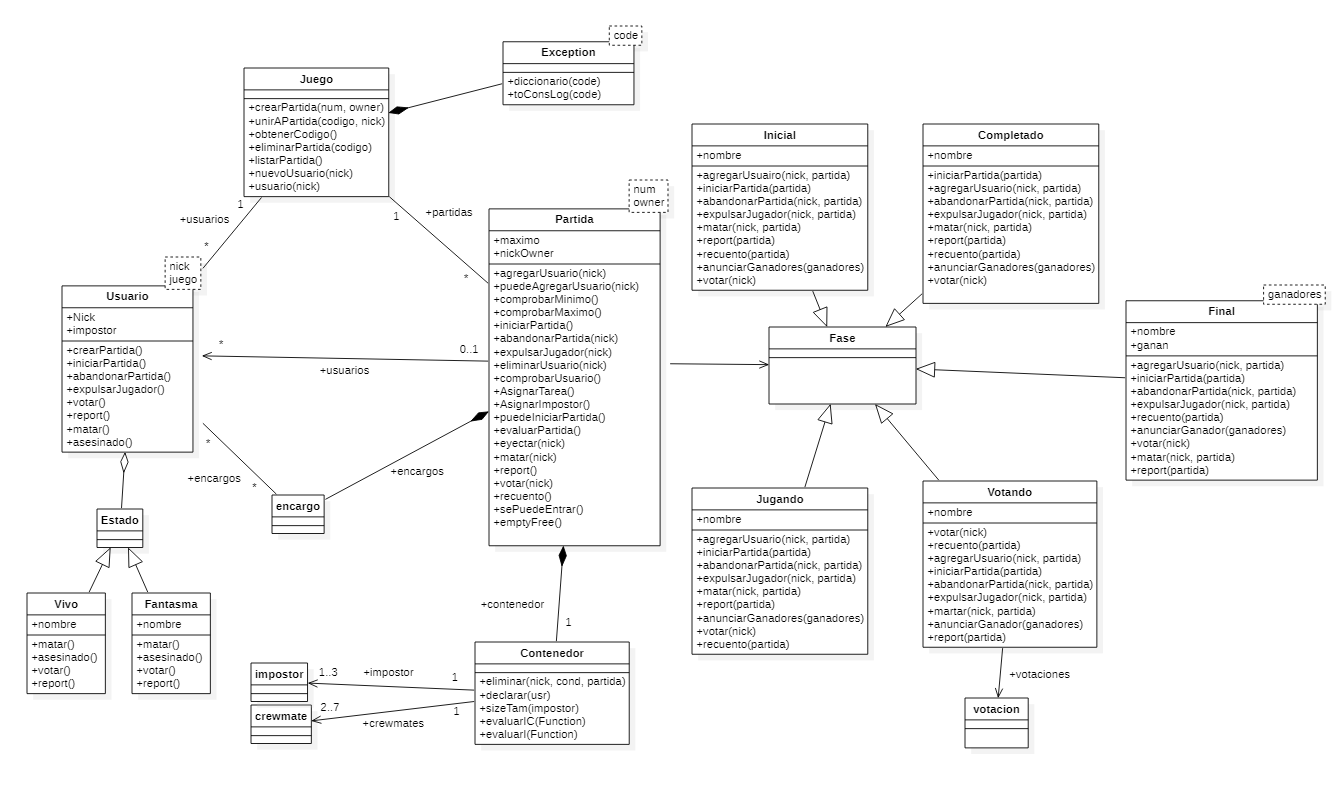


Figura : Arquitectura final del sprint 1

### Diagramas de secuencia y explicación del código

### Crear Partida

Como podemos observar en le *diagrama de secuencia 1* la funcionalidad crear partida, es solicitada desde un usuario al Objeto Juego, de tal manera que en este se evaluará si la partida puede ser creada, en caso de no ser posible, se lanzará una Exception emitiendo el mensaje N410. Pero en caso de poder crearse la partida, el juego solicitará obtener un código valido, e instanciará un objeto de la clase Partida utilizando ese código, de tal manera que al usuario se le tendrá que indicar que partida es la suya, y además se retornara el código de la partida creada.

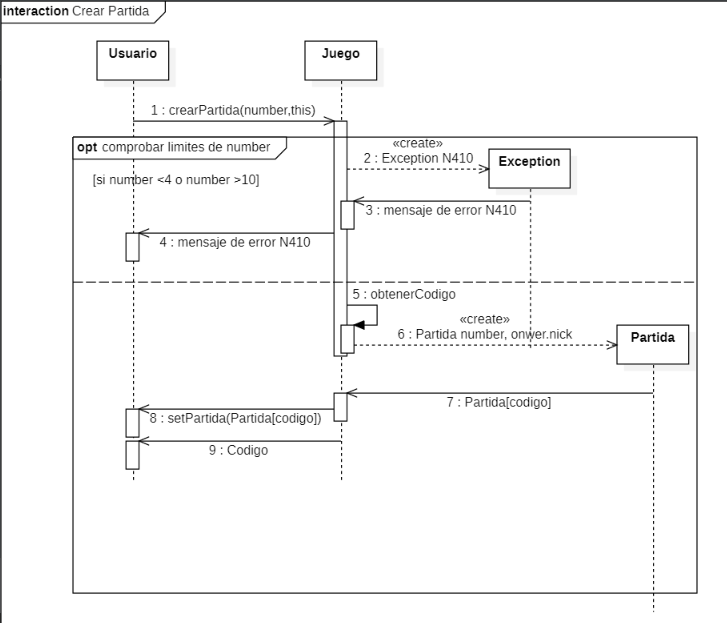
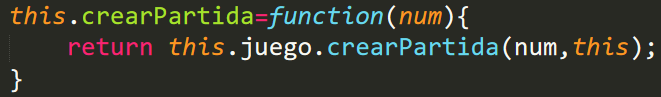
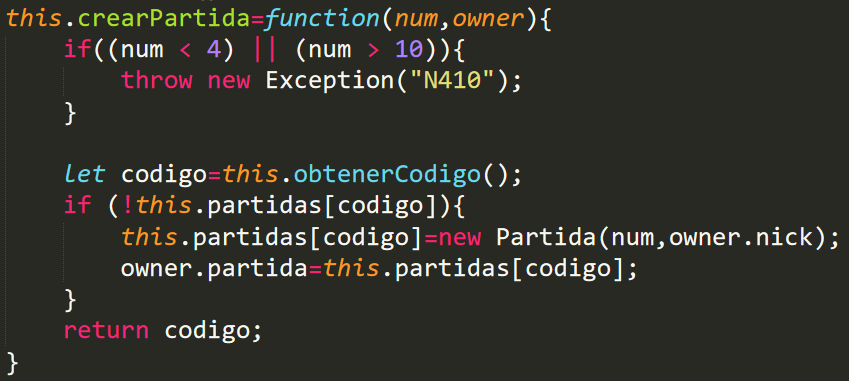


Diagrama de secuencia : Crear Partida

Además, el código asociado se puede ser en *código 1 y 2,* además de encontrar la información relacionada a la excepción N410 en el anexo B Excepciones.



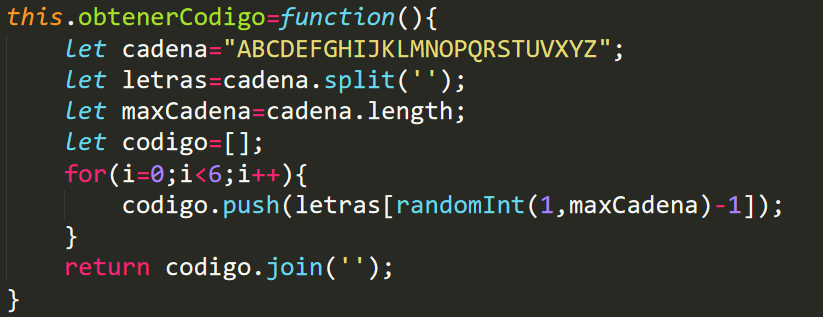
Código : crearPartida usuario



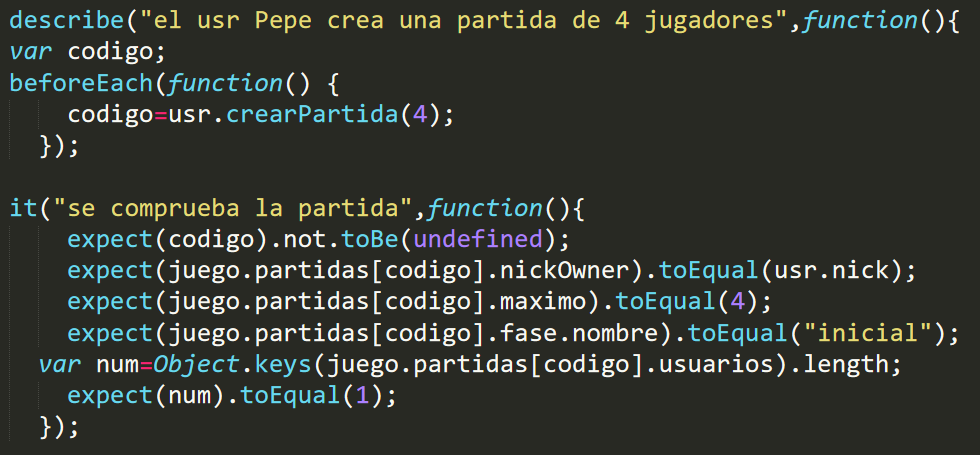
Código : crearPartida Juego

Como podemos observar en *código 1,* vemos que el jugador solicita crear una partida nueva al objeto *Juego.* Y este se encarga de la gestión de la creación de la partida, de tal manera que, primero hace una comprobación del parámetro *num,* que obligatoriamente tiene que estar comprendido en el rango [4,10], en caso de estar fuera del rango se ejecuta la directiva de excepción N410.

Por otro lado, en caso de estar dentro del rango permitido el juego solicitará la generación de un código para la partida, es decir, se llamara a la función *obtenerCodigo* recogida en *código 3*, una vez se ha generado el código, se instancia un objeto partida que es almacenado en el array asociativo *partidas* alojado en el objeto Juego, además, de dar a conocer la partida a su propietario que es el usuario que solicito su creación. Esta función devolverá el código de la partida para poder tener un control en las pruebas de *Jasmine* que como veremos estas pruebas están recogidas en *código test 1.*



Código : obtenerCodigo Juego



Código test : Crear partida

### Unir a Partida

Se observa en el *diagrama de secuencia 2,* que un usuario solicita al juego unirse a una partida con un determinado código que hace referencia a la partida a la que se quiere unir el usuario. Siendo el juego quien realiza la petición final a la partida para que agregue si es posible un nuevo usuario. En caso de no poder agregar un usuario por algún motivo se lanzará la excepción correspondiente.

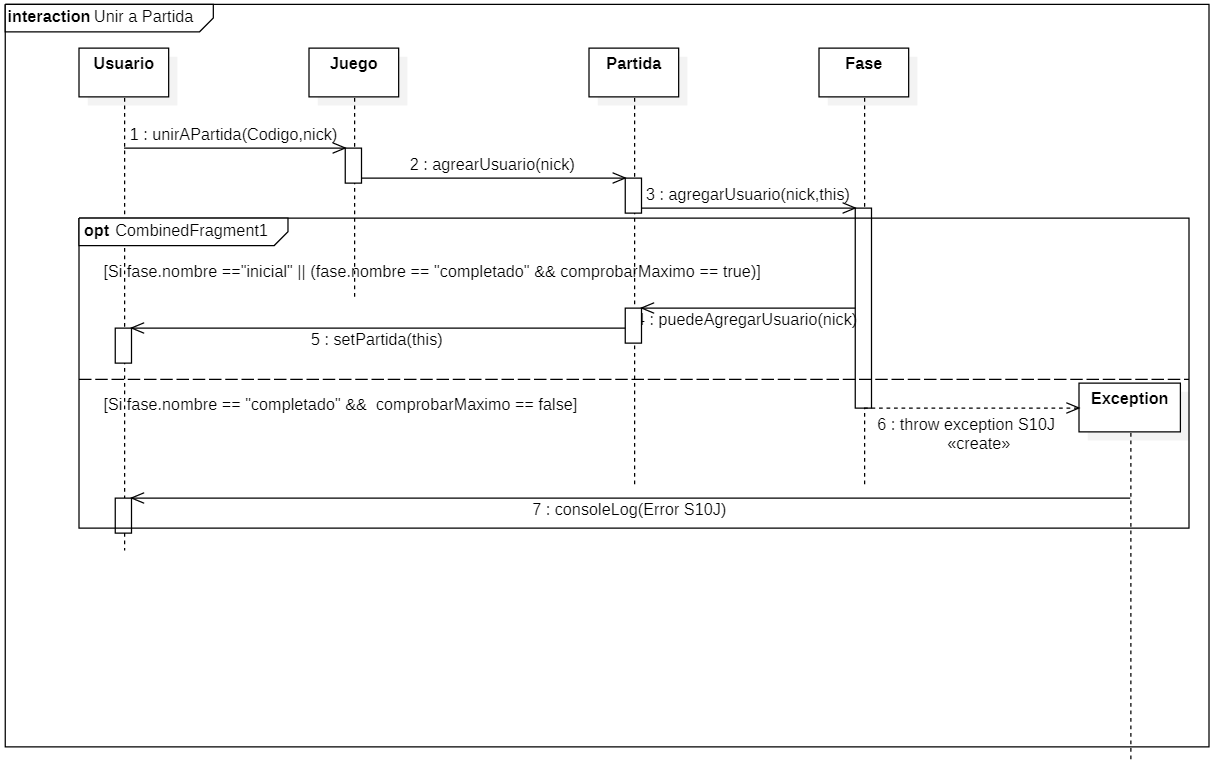
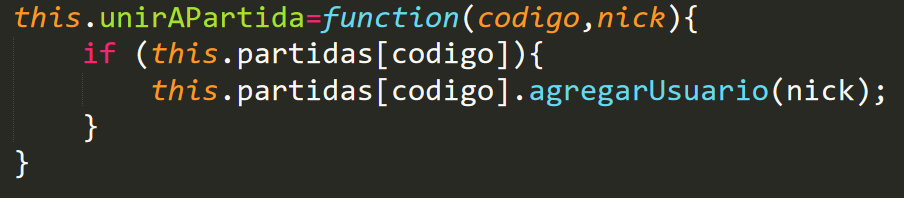


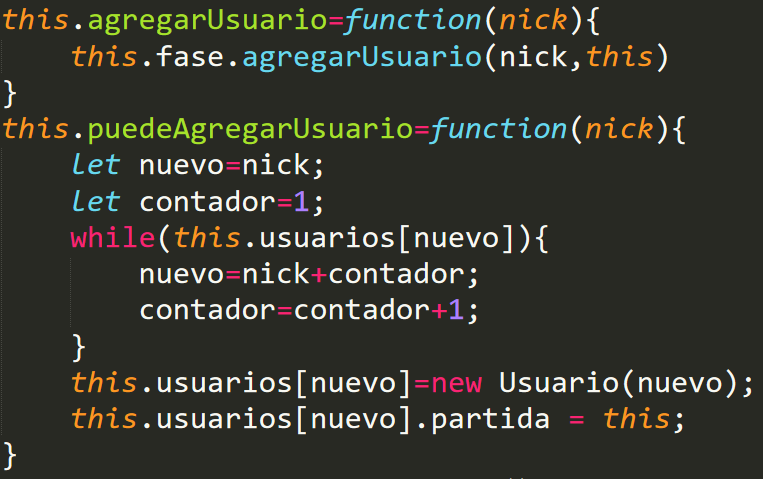
Diagrama de secuencia : Unir a Partida

El código resultante del diseño de este diagrama de secuencia viene recogido en el rango de códigos 4 *a 8*.

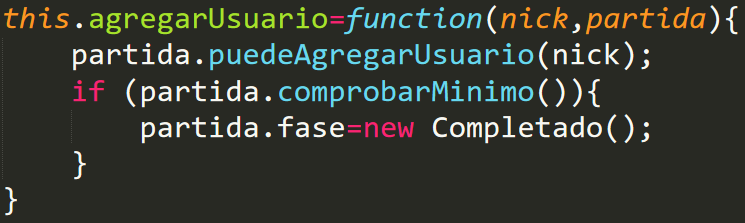


Código : unirAPartida – Juego

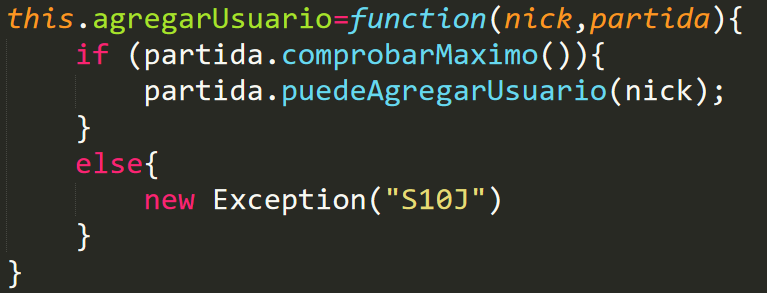
Como vemos en código 4, el Juego llama a la función agregar usuario de una partida de la cual disponemos del código, con el fin de ingresar un nuevo jugador a la partida a partir del Nick de este, a pesar de esta llamada, partida delega en fase la operación de agregar partida, para poder comprobar si esta en una fase en la que se pueda agregar un usuario a la partida elegida. Una vez se resuelve la fase ver *código 6 y 7* esta llama a la función alojada en partida *puedeAgregarUsuario* la cual se encarga de realizar la operación de agregar al usuario a la partida.



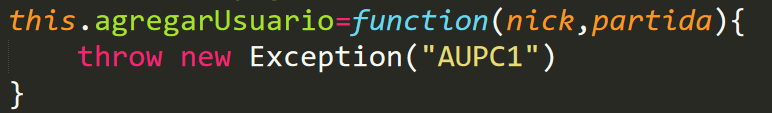
Código :agrearUsuario & puedeAgregarUsuario – Partida



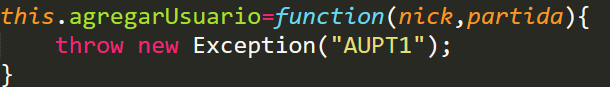
Código : agregarUsuario - fase inicial



Código : agregarUsuario - fase completado



Código : agregarUsuario - fase jugando



Código : agregarUsuario - fase final

Como partida delega en fase, tenemos que definir como actuara la llamada agregarPartida en cada fase, siendo en las fases *jugando*, *votancion* y *final*, cuando no puede realizar la operación lanzando una excepción en cada una de las fases.

### iniciarPartida

Como se observa en el *diagrama de secuencia 3* solo se podrá comenzar partida en la fase de completado, por lo que para el resto de las fases se lanzará una exception.

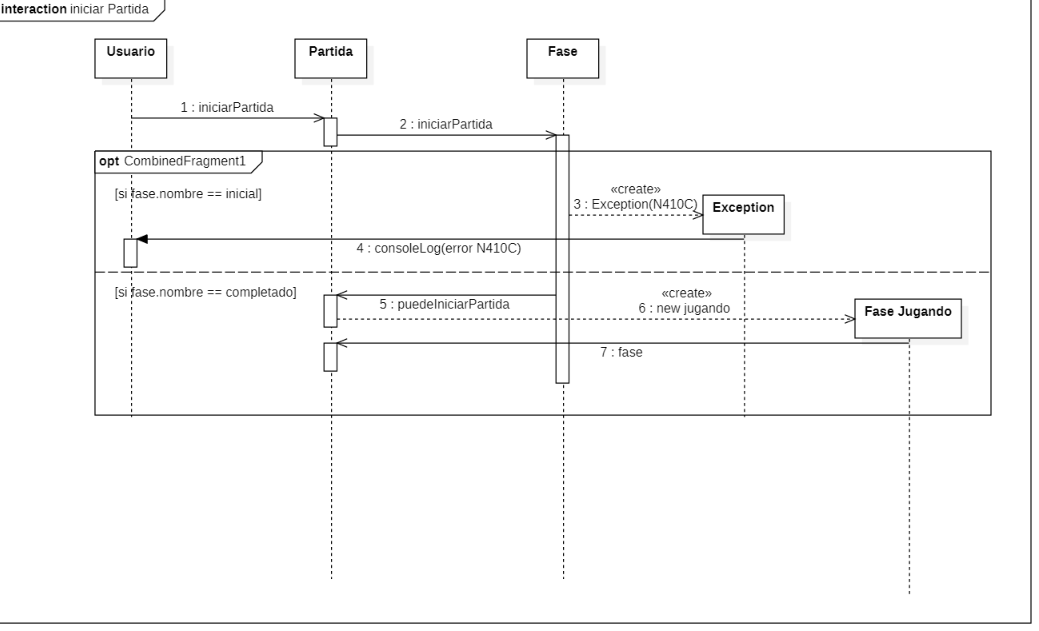


Diagrama de secuencia : iniciarPartida

## Reunión al final del sprint

### Producto onwer

Como vemos en le diagrama 1 el equipo, aparentaba trabajar tan solo los miércoles y jueves, después de consultarle al equipo que estaba ocurriendo, comentaron que no podían completar tareas hasta finalizar con los tutoriales mencionados de la historia de usuario crearPartida. Además, se avanzaba con pequeñas piezas de código, que dependían las unas de otras, por tanto, se cumplió con los plazos de estimación. Pero para el siguiente sprint el equipo ya esta avisado que debe fichar en kunagi, cada tarea finalizada. A pesar de las posibles dependencias. Dejando tan solo, la finalización de la historia de usuario sin cerrar para ser verificada.

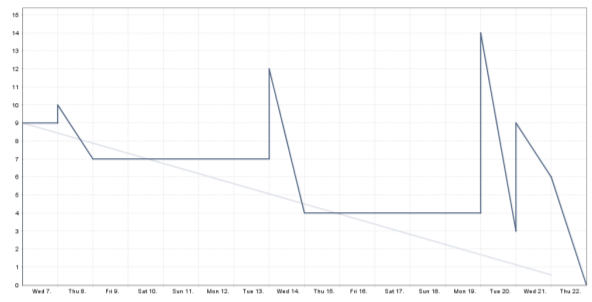


Diagrama : BurnDown

### Retrospectiva del equipo

El sprint se ha resulto con satisfacción, sin embargo, el equipo ha echado en falta la opción de crear tareas que permitan llevar a cabo tareas más complejas. Por ejemplo, para las tareas de diseño quizás se quería poder establecer un rediseño del diagrama de la arquitectura base, y además definir un diagrama de secuencia que permita esclarecer los movimientos en el flujo de las peticiones.

Esta carencia algunos miembros del equipo, han utilizado todoist para solventarla en la *figura* 5 se muestra un ejemplo de las tareas definidas en la aplicación. De tal forma que hasta que no estén completadas las subtareas de las tareas generales registradas en kunagi. Permitiendo así, acceder a un mayor detalle o mejor dicho definir con mayor detalle las tareas definidas en kunagi.

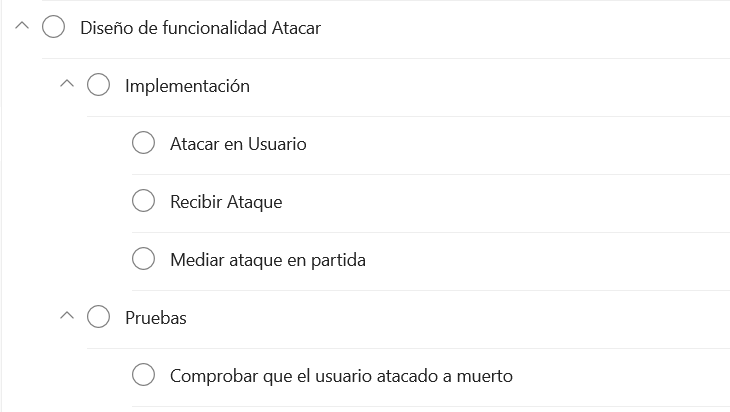


Figura : Tareas con TodoIst

Además, el equipo entiende que debe establecer un protocolo de fichaje en kunagi, con tal de que no ocurra lo mismo en el siguiente sprint de tener desinformado al product owner, y parecer que no se ha realizado ningún avance en el proyecto. Esto lo ha solicitado el product Owner para evitar por tanto que ocurra lo mismo que en el *diagrama 1 burndown*

# Bibliografía

[1] Migdal, P. (6 de septiembre de 2010). *Cornell University*. Recuperado el 2 de noviembre de 2020, de A mathematical model of the Mafia game: https://arxiv.org/abs/1009.1031

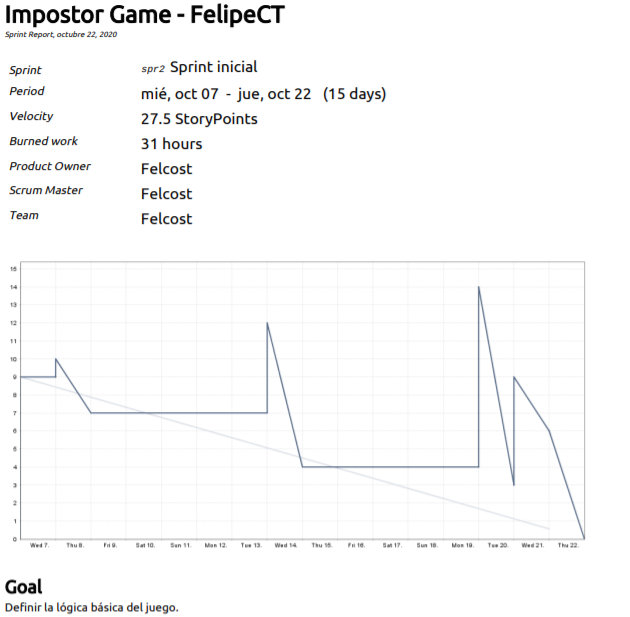
*[2] Wikipedia - Mafia Game*. (21 de septiembre de 2020). Obtenido de Mafia (juego): https://es.wikipedia.org/wiki/Mafia\_(juego)

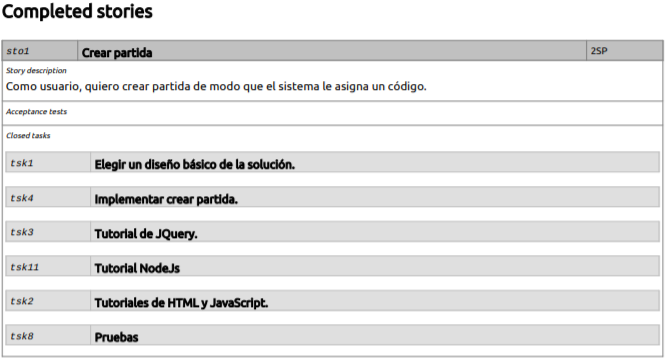
[3] Lugris, M. (31 de octubre de 2020). *TheGamer*. Obtenido de InnerSloth's Party Game Among Us Reaches 1.5 Million Simultaneous Players: https://www.thegamer.com/among-us-1-5-million-players-simultaneous-innersloth-party-game/

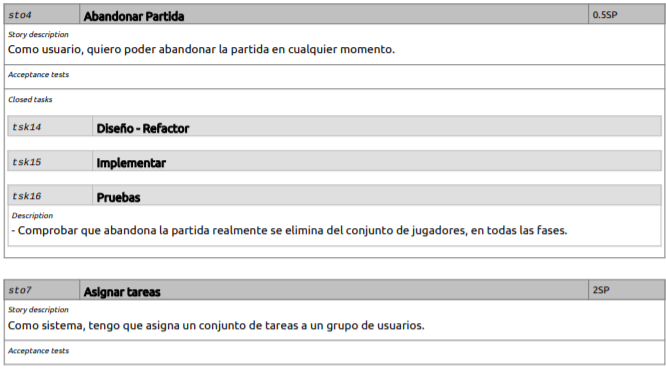
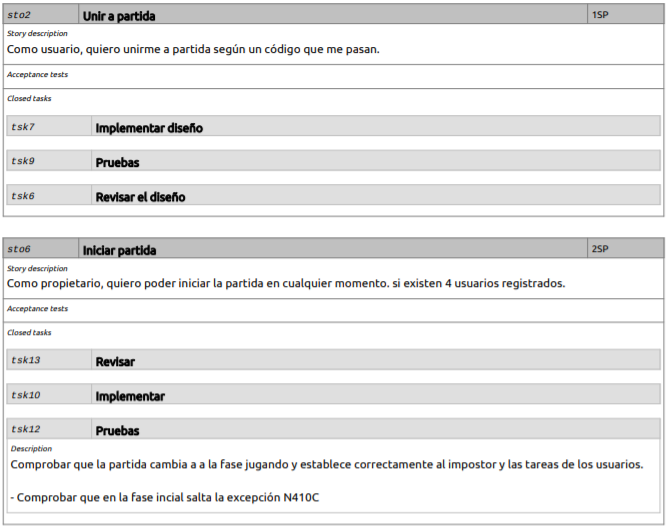
[4] Wikipedia - Pueblo duerme. (28 de agosto de 2020). *Wikipedia*. Obtenido de Los hombres lobo de Castronegro: https://es.wikipedia.org/wiki/Los\_hombres\_lobo\_de\_Castronegro

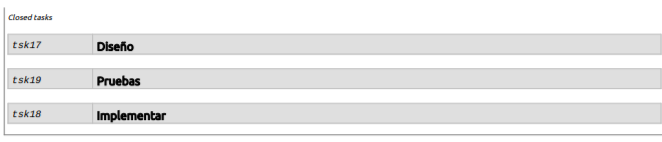
[5] Gamma, E., Helm, R., Johnson, R., & Vissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.*

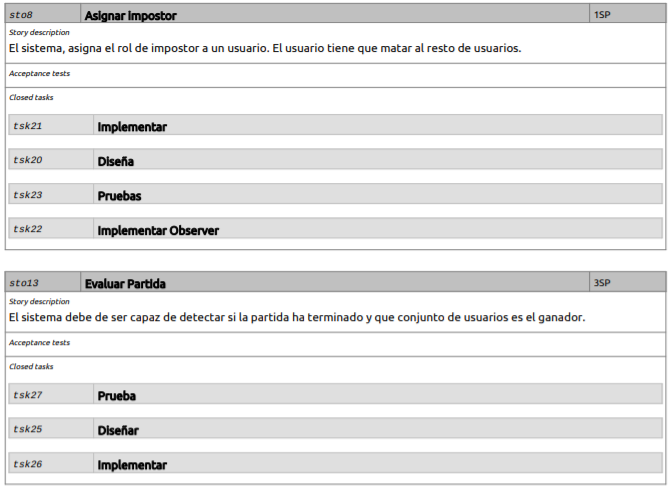
Anexo A - Documentación de Kunagi

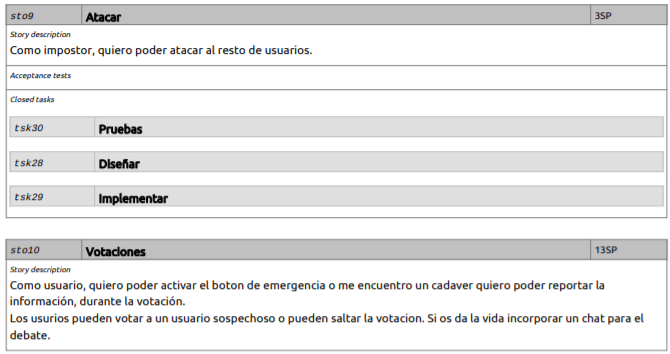


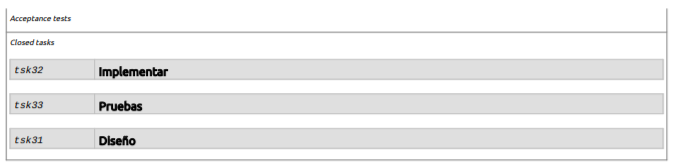






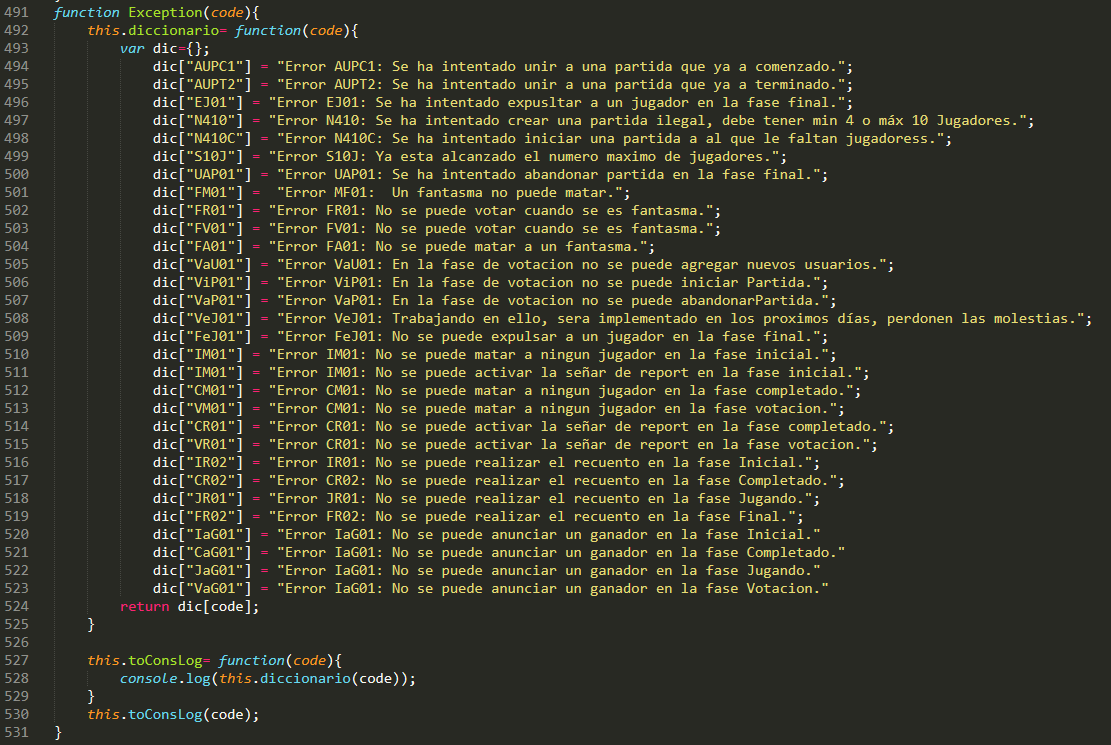






Anexo B – Exception y otras funciones

El objeto *Exception* es un diccionario que recoge todos los mensajes de excepción que soporta la capa de negocio. Además, se encarga de la propia gestión de mostrar por pantalla el mensaje de error correspondiente. Se optó por un diccionario o array asociativo, donde asignamos una clave inequívoca a cada mensaje de error.



Sin embargo, muy probablemente este diccionario termine siendo eliminado, y creando un artefacto Exception donde se recogen las excepciones, este artefacto utilizara tecnología JSON, sin embargo, el equipo esta tratando de gestionar como podría realizarse esta nueva versión.

Por otro lado, tenemos la función randomInt, esta función devuelve un numero entero entre dos valores, incluido estos.



Por último, tenemos la función *sizeDictionary* que como su nombre indica, devuelve el tamaño de un diccionario pasado como parámetro.

