UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA INFORMÁTICA

VIDEOJUEGO DE NAVEGADOR BASADO EN CANVAS 2D, PHONEGAP Y WEBSOCKETS EN NODE.JS

Realizado por
ANTONIO REY DÍAZ

Dirigido por
ANTONIO JESÚS NEBRO URBANEJA

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

MÁLAGA, Noviembre de 2017

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente/a Dº/Dª. _____

Secretario/a Dº/Dª. _____

Vocal Dº/Dª. _____

para juzgar el proyecto fin de carrera titulado:

Videojuego de navegador basado en Canvas 2D, Phonegap y Websockets en Node.JS

realizado por Dº **Antonio Rey Díaz**,
tutorizado por Dº **Antonio Jesús Nebro Urbaneja**,
ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga a ___ de _____ del 20__

El/La Presidente/a | El/La Secretario/a | El/La Vocal

Fdo: | Fdo: | Fdo:

AGRADECIMIENTOS

La creación de este proyecto ha sido posible, además de por mi esfuerzo personal, por la ayuda que me han brindado muchas personas.

En primer lugar, me gustaría recordar a mi abuela Encarna, que falleció hace justo un año, y que aún en su enfermedad me seguía recordando y siempre se portaba fenomenal conmigo y con todos sus nietos.

Tengo que valorar por supuesto la ayuda no solo económica sino personal de mis padres Antonio y Ana y de mi hermano Javier, que siempre han estado ahí para darme todo lo que necesitara. Mis padres siempre me enseñaron que hay que hacer, o por lo menos intentar hacer siempre las cosas lo mejor posible, y mantener una conducta ética y moral hacia todo el mundo. Todo lo que me han enseñado me ha valido no solo para este proyecto, sino para toda mi vida.

Por último, me gustaría dar las gracias a una persona que ha sido imprescindible para la realización de este proyecto, a la que considero una gran profesional y una gran persona y que se llama Lidia Colmenero. Siempre ha tratado de rescatar lo mejor de mí y alejar todo aquello que me pudiera perjudicar, y su constancia y su ayuda han sido verdaderamente increíbles. Si todo el mundo fuera como ella, este sería un mundo mejor. Gracias Lidia.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

Los videojuegos son juegos en los que se usa un ordenador y a través de vídeo y de una pantalla, además de audio, todos los jugadores pueden ver cómo evoluciona el juego, y además pueden interactuar con él usando múltiples dispositivos electrónicos.

Aparecieron a partir del surgimiento de los primeros ordenadores de sobremesa domésticos en la década de los 70 del siglo pasado, a la vez que surgieron las videoconsolas, que son ordenadores diseñados para el propósito específico de jugar a videojuegos. En un principio, los videojuegos eran de un solo jugador, o si eran de varios jugadores, éstos tenían que conectarse físicamente y directamente a la videoconsola o al PC a través de algún dispositivo como un teclado o un joystick. Es decir, si varias personas querían jugar a un videojuego, tenían que hacerlo en la misma habitación y solían ser familiares o amigos. Existían redes locales para poder jugar a videojuegos de varios jugadores (multijugador), como un sistema llamado PLATO que surgió en los años 70 del siglo pasado, pero cuando de verdad se popularizaron los videojuegos multijugador en red fue cuando el uso de Internet se trasladó también al ámbito doméstico a mediados de los años 90.

A través de Internet, personas ubicadas en distintas regiones del planeta podían jugar al mismo videojuego usando cada uno su propia videoconsola o su PC. La mayoría de estos videojuegos, exigían instalar un software del videojuego en el PC o videoconsola de cada jugador. Para navegar por Internet se usaba un software fundamental: el navegador. En un principio, el navegador solo servía para visualizar archivos localizados en Internet, dada la dirección o URL donde estaba localizado. Pero con el tiempo, y hasta nuestros días, el navegador ha ido evolucionando hasta convertirse en un software muy complejo con el que se pueden ejecutar programas completos y no solo visualizar archivos. Ahora mismo, con un navegador se puede ver todo tipo de contenido multimedia que cambia en tiempo real. Y a la vez, el usuario puede interactuar con el navegador e indicarle órdenes que se pueden transmitir a través de Internet. Así que se pensó que para jugar a videojuegos, era una buena forma hacerlo con un navegador. Y así es como surgieron los videojuegos de navegador multijugador.

1.1 Objetivo del proyecto

El presente proyecto tiene como objetivo el desarrollo de un videojuego de navegador multijugador que se puede jugar en red a través de Internet o de alguna red local que use sus mismos protocolos. Para su desarrollo, se ha querido utilizar tecnologías de reciente creación y que actualmente se están usando mucho.

Un videojuego de navegador es una aplicación web que sigue la arquitectura cliente/servidor. Es decir, el software a desarrollar está dividido en dos partes: el cliente y el servidor. El cliente es la parte que se ejecuta en el navegador que está instalado en el PC, o en el smartphone o tablet de cada uno de los jugadores. El software del servidor, se ejecuta, como su nombre indica, en el servidor, que es una máquina o máquinas a la que se conectan todos los jugadores desde sus navegadores, y es desde donde se gestiona la comunicación e interacción entre todos ellos y donde están guardados los datos del juego.

En este proyecto, para la parte cliente se han usado las siguientes tecnologías: HTML5, CSS, JavaScript , Canvas 2D y websockets. Además, para el caso de que el cliente se ejecute en un smartphone o tablet, se ha usado Phonegap, que permite ejecutar la parte cliente desarrollada en las tecnologías anteriores con muy pocos cambios salvo para acceder a las funciones especiales que tienen los dispositivos móviles, como por ejemplo el acelerómetro. Phonegap es propiedad de la compañía Adobe, pero ha permitido distribuirlo con licencia Apache bajo el nombre de Cordova. Aunque con distinto nombre, se trata en ambos casos del mismo software.

En la parte de servidor se han usado las siguientes tecnologías: Node JS, Express JS, websockets (Socket.io) y MongoDB. Todas estas tecnologías usadas se explicarán en los capítulos siguientes de la memoria.

1.2 ¿En qué consiste el videojuego del proyecto?

El videojuego consiste en un tablero en dos dimensiones. En ese tablero existen unos círculos de colores. Algunos de esos círculos representan jugadores y se mueven por el tablero accionados por ellos. Otros de esos círculos representan las bolas, que no se mueven y pueden ser comidas por los jugadores, para así ganar puntos y aumentar de tamaño. Para un jugador, aumentar de tamaño es fundamental, porque ello le permite poder comerse a jugadores más pequeños, lo que a su vez hace que el jugador crezca aun más en el tamaño del jugador comido. Habrá también jugadores que se moverán solos sin que ningún ser humano mande sobre ellos: los bots. Para equilibrar la balanza en el juego, y que los jugadores más grandes no tengan el monopolio del juego, todos los jugadores podrán lanzar una bomba cada cierto tiempo. Si una bomba choca contra un jugador (o contra cualquier elemento del juego), ésta explotará, y su onda expansiva hará más pequeños o matará a los jugadores que estén en su radio de alcance. Aunque una bomba no choque contra nada, al cabo de un tiempo, explotará sola. El juego también permite que los jugadores puedan chatear por voz a través de notas de voz. Cada jugador podrá a escuchar a otros y ver el nick de quien le esté hablando en ese momento. Toda la funcionalidad del videojuego se detallará y explicará en el capítulo 2, dedicado al análisis del software.

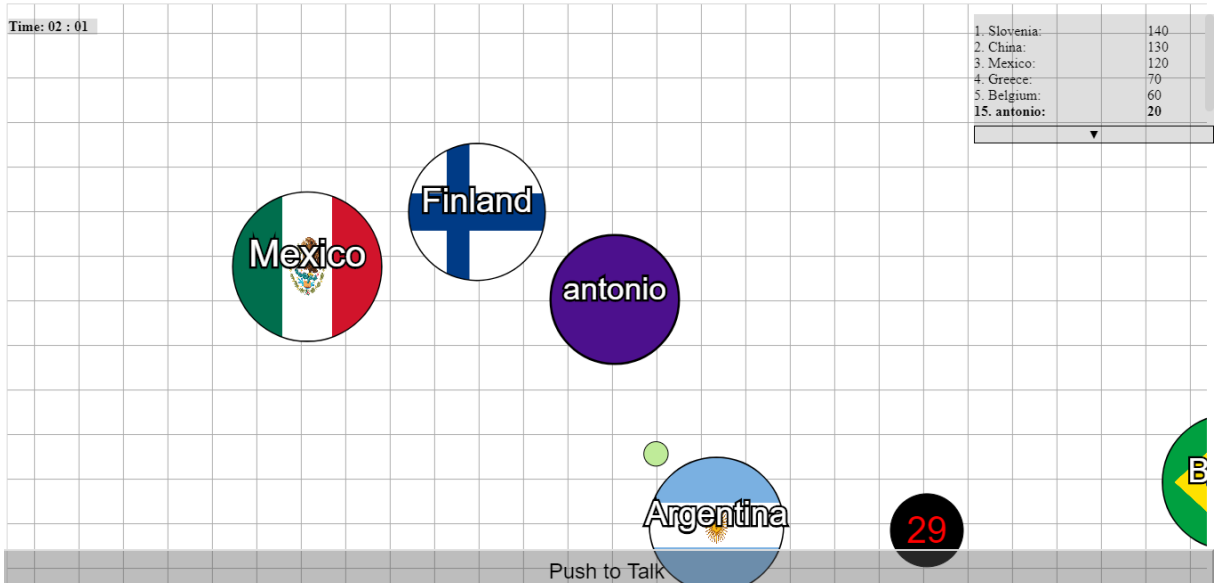


Figura 1.1 Captura de pantalla del videojuego

1.3 Aportación realizada en el proyecto

La principal aportación de este proyecto es el uso y análisis de tecnologías de reciente creación para el desarrollo del videojuego. A continuación se van a describir, tanto para la parte cliente como para la parte servidor del software, las tecnologías que se han usado tradicionalmente para el desarrollo de videojuegos de navegador, contrastándola con las nuevas tecnologías usadas en este proyecto. Se dividirá en dos apartados llamados “aportaciones para la parte cliente” y “aportaciones para la parte servidor”.

1.3.1. Aportaciones para la parte cliente

Las tecnologías que tradicionalmente se han usado para la parte cliente son HTML, CSS y JavaScript.

HTML (HyperText Markup Language) es un lenguaje basado en etiquetas que permite definir qué elementos existen en una página web. En el caso de un videojuego, permite definir qué elementos forman parte de la interfaz gráfica. Por ejemplo, con este lenguaje se pueden definir los formularios para introducir datos por parte del jugador a la hora de conectarse con su nick y contraseña. O se pueden definir botones y dónde están ubicados.

CSS (Cascading Style Sheets) es un lenguaje también basado en etiquetas que permite definir el diseño gráfico de los elementos definidos en HTML. Por ejemplo se pueden definir los colores de las interfaces gráficas, su tamaño, los bordes, etc.

JavaScript en cambio es un lenguaje de programación. Permite definir el comportamiento de la parte cliente. En un videojuego, JavaScript permite por ejemplo especificar que un texto vaya cambiando de forma dinámica, o cambiar su diseño gráfico, o enviar información al servidor.

Lo que durante mucho tiempo no ha podido definir estas tres tecnologías es algo fundamental para un videojuego: el vídeo, sobre todo el vídeo en el que el usuario puede interactuar con él. Ante este vacío tecnológico, para realizar videojuegos de navegador, tradicionalmente se han usado tecnologías ajenas al navegador que había que instalar aparte. Son los llamados plugins del

navegador. Uno de los más usados para este menester ha sido el Adobe Flash Player. Estos plugins dan y han dado muchos problemas a lo largo del tiempo. Usan lenguajes de programación propios, no estandarizados, lo cual ha generado muchos problemas relativos a la seguridad. Y hay más problemas como que por ejemplo los desarrolladores tienen que aprenderse lenguajes distintos, problemas de compatibilidad y que el hecho de tener que instalar software aparte del navegador va en contra de la filosofía de que el único software que se tiene que usar en el cliente sea el navegador.

Por eso, en este proyecto, se va a utilizar una tecnología nueva en sustitución del uso de plugins: Canvas. En concreto, su versión para gráficos de dos dimensiones (Canvas 2D). Canvas es una nueva etiqueta de la nueva versión de HTML (HTML5) que permite dibujar gráficos a través del lenguaje JavaScript y de la API de Canvas. Con esta tecnología ya no se necesita aprender un lenguaje de programación nuevo, porque se usa JavaScript. Solo hay que aprenderse la API de Canvas, que permite dibujar líneas, círculos, dibujar imágenes, etc.

Otra tecnología que se va a usar en este proyecto es para la ejecución del videojuego en dispositivos móviles. Anteriormente para ejecutar videojuegos de navegador en dispositivos móviles, había que desarrollar una aplicación específica para el sistema operativo del dispositivo. Pero con Phonegap, ahora es posible reutilizar el cliente desarrollado en HTML, CSS y JavaScript, ya que lo que hace es generar automáticamente una aplicación en la plataforma especificada (Android, iOS, etc), y solo hay que cambiar lo respectivo al acceso al hardware del dispositivo móvil usando la API de Phonegap: el acelerómetro, los sensores, los botones que son diferentes a los del PC, etc.

1.3.2 Aportaciones para la parte servidor

Para la parte de servidor se suelen usar muchas tecnologías web diferentes. Normalmente suele haber un servidor web Apache, unido a un intérprete de un lenguaje de programación de servidor junto con un framework, que genera el contenido dinámico que se envía al cliente. Ejemplos de las tecnologías más usadas en el servidor son PHP, Java y Microsoft .Net.

Pero en este proyecto se va a usar una tecnología diferente, llamada Node JS. Node JS es un entorno de ejecución que ejecuta código escrito en el mismo lenguaje de programación de la parte cliente:

JavaScript. En Internet existen paquetes para Node JS que se pueden descargar e instalar, como por ejemplo uno llamado Express con el que se puede crear un servidor web y uno que permite convertir el servidor en un servidor de websockets (Socket.io). Los websockets permiten establecer una conexión permanente entre el cliente y el servidor, algo que HTTP no permite.

Para la parte servidor se suelen utilizar gestores de bases de datos basados en el lenguaje estandarizado SQL, como por ejemplo SQL Server, MySQL, etc. Sin embargo, en este proyecto se va a usar un gestor de base de datos no basado en el lenguaje SQL, es decir, es un gestor de bases de datos No-SQL. Su nombre es MongoDB, y para el caso de este proyecto puede ser útil.

1.3.3 Esquemas de aportaciones en el proyecto

En cada uno de los siguientes esquemas se puede ver en el lado izquierdo las tecnologías tradicionalmente usadas para juegos de navegador, y en el lado derecho las tecnologías que se van a usar en este proyecto.

En el primer esquema se ve el contraste entre las tecnologías usadas en el cliente PC (navegador web):

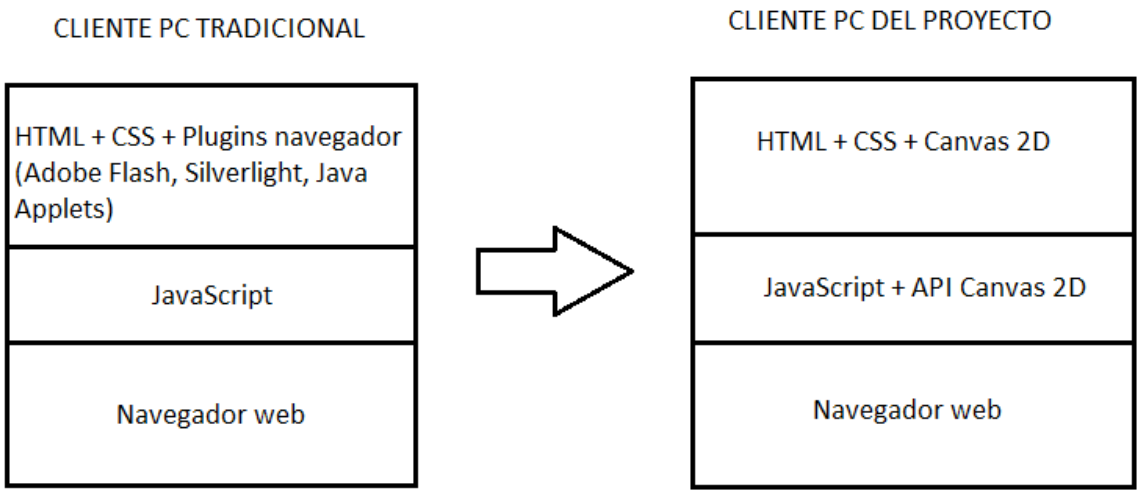


Figura 1.2 Aportaciones en el cliente PC

En este segundo esquema se puede ver el contraste entre las tecnologías usadas en el cliente móvil (smartphone, tablet):

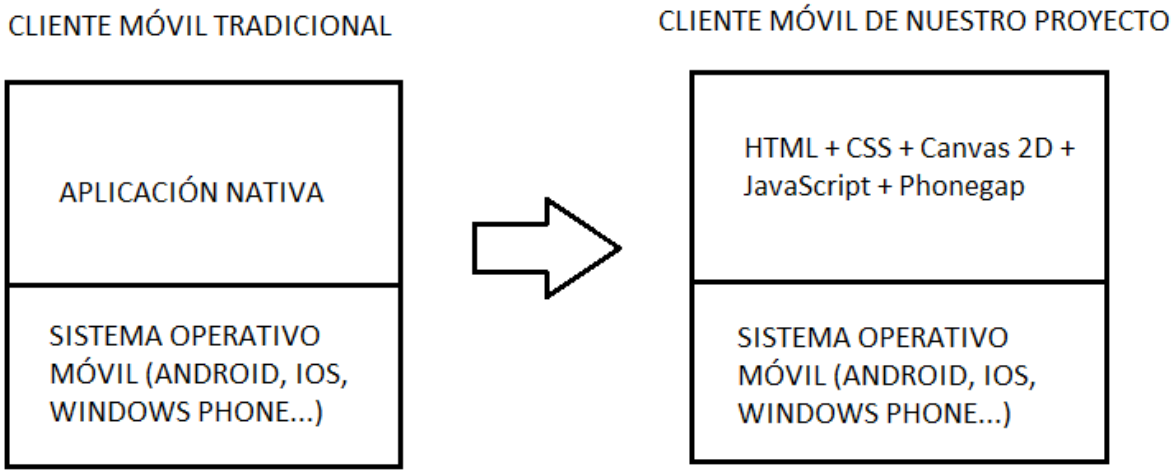


Figura 1.3 Aportaciones en el cliente móvil

Y en este tercer esquema se puede ver el contraste entre las tecnologías usadas en el servidor:

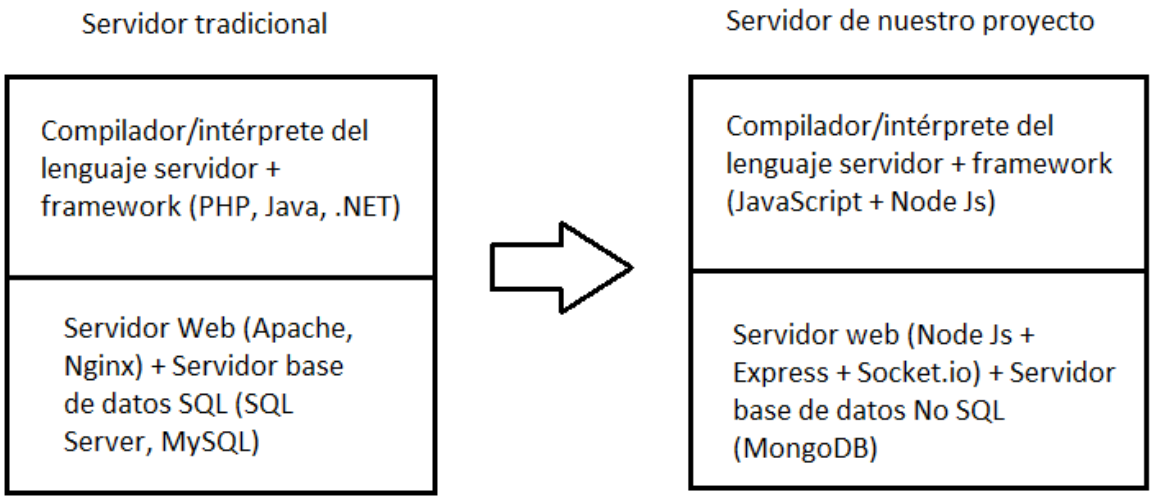


Figura 1.4 Aportaciones en el servidor

1.4 Planificación del proyecto

La planificación del proyecto consistió en un principio en la realización de un prototipo. Este prototipo sirvió para eliminar incertidumbre sobre qué tareas realizar y cuánto se iba a tardar en cada una de ellas. El prototipo consistió en realizar un esqueleto del software implementando algunas pequeñas funcionalidades, probando así cada una de las tecnologías para ver su dificultad, como por ejemplo la implementación de mostrar una bola del juego para probar la tecnología Canvas 2D, algunas pequeñas funcionalidades para aprender a comprender el modelo de ejecución asíncrono de JavaScript, y la implementación de una conexión por websockets.

A partir de este prototipo, se desarrolló el resto del software, haciendo un desarrollo basado en iteraciones y en casos de uso. En cada iteración se desarrolló un caso de uso, aunque durante el desarrollo hubo casos de uso que se dividieron en varios, se añadieron casos de uso que se vieron interesantes para el videojuego y hubo funcionalidades que al final no se implementaron porque la extensión del proyecto era ya larga.

A continuación se va a detallar por un lado la lista de recursos utilizados durante el proyecto, y luego la lista de tareas que se planificó en un principio y cómo éstas luego se modificaron.

1.4.1 Recursos utilizados

Recursos software:

- Entorno de desarrollo Netbeans.
- StarUML, una herramienta para hacer diagramas UML, que ha servido para realizar los diagramas UML para el análisis y diseño del software.
- Node Js, con el que viene el software NPM, que sirve para descargar paquetes de Node Js.
- Los paquetes de software descargados con NPM llamados Express y Socket.io (Express para crear un servidor web y Socket.io para crear un servidor web con websockets)
- Gestor de base de datos MongoDB.
- Driver para poder conectarse a MongoDB con Node Js (descargado con NPM).

- Mocha, un software para poder hacer tests en JavaScript tanto en Node Js como en los navegadores web.
- Navegadores web Google Chrome y Mozilla Firefox.
- Phonegap, que sirve para empaquetar una aplicación web en el cliente basada en HTML, CSS y JavaScript para su ejecución en un sistema operativo de dispositivos móviles como Android.
- Windows 10 y Android.

Recursos hardware:

- PC portátil Intel Core i3
- Smartphone Huawei

1.4.2 Tareas planificadas

Las tareas planificadas en un principio en el proyecto fueron las siguientes:

- Aprendizaje y repaso de los lenguajes HTML y CSS, y todo lo referente al diseño web.
- Aprendizaje del lenguaje de programación JavaScript, que es el que más se va a usar en el proyecto.
- Aprendizaje de la API de Canvas 2D y su uso con JavaScript.
- Aprendizaje de Phonegap.
- Aprendizaje de Node Js, Express y del manejo de websockets.
- Aprendizaje de MongoDB: su modelo de datos y cómo se hacen consultas en este gestor.
- De forma paralela a las tareas anteriores, realizar el prototipo del software, explicado anteriormente.
- Una vez terminado de realizar el prototipo y aprendido un poco todas las tecnologías a usar y su dificultad, hacer un listado de los posibles requisitos que va a tener el sistema.
- Establecer una tarea distinta para cada uno de los desarrollos iterativos de los requisitos listados anteriormente. Cada una de esas tareas consistirá en el análisis, diseño, implementación y pruebas de cada uno de los siguientes requisitos:

- Registro de un usuario en el juego.
- Logueo de un usuario con su contraseña.
- Muestra del jugador propio con todos sus atributos.
- Muestra de los demás jugadores con todos sus atributos.
- Movimiento de un jugador y su transmisión al resto de jugadores.
- Un jugador puede comer bolas.
- Proceso de que un jugador mate a otro y la comunicación del suceso al resto de jugadores.
- Proceso de que algún elemento del juego mate o dañe a algún jugador y la comunicación a todos los jugadores.
- Gestión de las estadísticas del juego.
- Un jugador puede hablar por micrófono con otros jugadores.
- Un jugador puede chatear por escrito con otros jugadores.
- Panel de administración donde el administrador pueda añadir jugadores, eliminarlos, cambiar sus atributos, etc.

La mayor parte de estos requisitos iniciales son los mismos que los requisitos finales que se detallan en el capítulo 2 dedicado al análisis del software. Otros se han dividido en varios requisitos, se han añadido algunos y se han eliminado otros. Esto ha provocado que se añadan, modifiquen o eliminen algunas tareas de la planificación durante el desarrollo, ya que cada tarea consistía en el análisis, diseño, implementación y pruebas de un requisito. En concreto, se han añadido como tareas el desarrollo de los siguientes requisitos:

- Lista de jugadores ordenada de mayor a menor por puntos durante la partida.
- Los casos de uso referentes a las partidas. Durante el desarrollo del proyecto se vio interesante que hubiera partidas con tiempo de finalización determinado.

Se han eliminado las tareas de desarrollar los siguientes requisitos por hacer muy extenso el proyecto (no estaban incluidos en el anteproyecto, pero en un principio se pensó en desarrollarlos):

- Un jugador puede chatear por escrito con otros jugadores.
- Panel de administración donde el administrador pueda añadir jugadores, eliminarlos,

cambiar sus atributos, etc.

Y se han modificado algunas de las tareas planificadas al principio porque los requisitos que se desarrollaban en ellas se han dividido en más requisitos, los cuales se detallan en el capítulo de análisis de la presente memoria. Además de modificar la tarea que consistía en el desarrollo del software que permitía que unos elementos de la partida pudieran dañar o matar a los jugadores, para hacer que estos elementos sean las bombas con el comportamiento especificado en los siguientes capítulos de la memoria.

1.5 Estructura de la memoria

La memoria se divide en cinco capítulos y dos apéndices. A continuación se resume el contenido de cada capítulo o apéndice:

Capítulo 1. Introducción
En este capítulo se cuenta el origen de los videojuegos de navegador multijugador, los objetivos del proyecto, una descripción resumida del videojuego que se ha desarrollado, la explicación de las aportaciones tecnológicas que se han realizado en el proyecto diferenciándolas de las tecnologías de desarrollo habituales en la actualidad y la planificación del proyecto.

Capítulo 2. Análisis
En este capítulo se hace el análisis de toda la funcionalidad del software, describiendo los requisitos funcionales y no funcionales, y describiendo los casos de uso mediante diagramas UML y descripciones textuales.

Capítulo 3. Diseño e implementación
En este capítulo se describe el diseño e implementación del software. Tiene dos apartados que cada uno explica el diseño e implementación de la parte servidor y el de la parte cliente respectivamente.
En cada uno de esos apartados se explica el diseño explicando primero su estructura mediante diagramas de clases en UML y luego explicando su dinámica o comportamiento mediante diagramas de actividad UML. Y después se describen las tecnologías utilizadas para pasar este diseño a la implementación.

Capítulo 4. Pruebas

En este capítulo se explica cómo se han realizado las pruebas del software. Por un lado se explica cómo se han realizado las pruebas de la parte servidor y por otro lado las de la parte cliente.

Capítulo 5. Conclusiones y trabajo futuro

En este capítulo se hace una especie de resumen de toda la memoria en el que se explica a qué conclusiones se ha llegado sobre todo en la utilización de las nuevas tecnologías comparándolas con las que se suelen utilizar habitualmente. Y también se explica qué posibles ampliaciones se podrían hacer si el desarrollo del software continuara en un futuro.

Apéndice A. Contenido del CD y Manual de instalación

Este apéndice describe el contenido del CD que acompaña a esta memoria, además del manual de instalación del software tanto para instalar el software en el servidor y en los dispositivos móviles.

Apéndice B. Manual de usuario

Contiene el manual de usuario, para que los jugadores sepan cómo jugar al juego tanto desde un PC como desde un dispositivo móvil.

CAPÍTULO 2: ANÁLISIS

En este capítulo se va a realizar el análisis del software. Realizar el análisis es importante, pues permite ver de forma esquematizada y detallada qué software hay que desarrollar. En primer lugar, se va a realizar el análisis de requisitos. Los requisitos se dividen en requisitos funcionales y no funcionales. Los requisitos funcionales se refieren a la descripción en detalle de cada uno de los requisitos o funcionalidades que debe tener el software, y los no funcionales se refieren a requisitos que debe tener nuestro sistema pero que no describen una funcionalidad. Después del análisis de requisitos se va a realizar el análisis de los casos de uso, exponiendo su descripción textual detallada.

2.1 Análisis de requisitos

2.1.1 Requisitos funcionales

1. **Registro de jugadores:**
   El software permitirá a un jugador registrarse en el videojuego con su nick, contraseña, y un avatar opcional. Si el nick del jugador no estaba registrado, permitirá el registro y le mandará un mensaje al jugador de que se ha registrado correctamente. En cambio, si el jugador ya estaba registrado y la contraseña no coincide con la de ese jugador, no permitirá que se registre y mandará un mensaje indicándole que el registro no se ha realizado. Si el jugador ya estaba registrado y la contraseña proporcionada es la de ese jugador, modificará el avatar por el proporcionado en este nuevo registro e indicará que el registro se ha realizado correctamente.
2. **Logueo de jugadores:**
   El software permitirá a un jugador registrado entrar en el juego mediante su nick y contraseña. Si el jugador con ese nick no está registrado, o la contraseña no es la correcta, le indicará al jugador que el login es incorrecto y que lo intente de nuevo. En cambio, si los datos introducidos son correctos, entrará en la partida que se esté jugando en ese momento, o entrará en la siguiente si aún no ha empezado ninguna. Si aún no ha empezado ninguna partida, se le mostrará un mensaje de que espere un poco a que empiece la partida. Habrá un máximo de jugadores en la partida, y si un jugador intenta loguearse y el número de jugadores en la partida sobrepasa este máximo, no le permitirá loguearse y mandará un

mensaje de que la sala está llena.

3. **Mostrar la partida actual.**

3.1. Mostrar las bolas.

Cada jugador podrá ver las bolas que hay en el área del tablero de juego que puede ver. Las bolas sirven para que los jugadores las coman y así aumenten su tamaño. Cada bola viene representada por un círculo de un color aleatorio.

3.2. Mostrar al jugador.

Cada jugador podrá ver la posición de su jugador en el tablero, a la vez que podrá ver su representación gráfica: un círculo con su nick escrito en el interior, y de fondo en el círculo un color aleatorio si no tiene avatar, o su avatar si es que lo tiene. Si aún no se ha logueado, mostrará una posición aleatoria en el tablero.

3.3. Mostrar al resto de jugadores.

El videojuego mostrará a cada jugador la posición del resto de jugadores que están en la partida, restringido a un área del tablero alrededor de la posición de cada jugador. Cada jugador se mostrará gráficamente mediante un círculo con su nick escrito en su interior y de fondo un color aleatorio si no tiene avatar, o su avatar si es que lo tiene.

3.4. Mostrar las bombas.

Las bombas se mostrarán como un círculo de color negro en cuyo interior se verá un cronómetro de los segundos que quedan para que explote por sí sola. Cuando una bomba explota, se mostrará la onda expansiva de la explosión, como una circunferencia que representa su radio de acción.

3.5. Mostrar el cronómetro de la partida

En cada momento de la partida se podrá ver un cronómetro que indica el tiempo que queda para que termine la partida.

3.6. Mostrar la lista de jugadores ordenada por puntos.

En todo momento en la partida se podrá ver una lista ordenada por puntos con los nicks de todos los jugadores y sus respectivos puntos. En la lista que ve cada jugador, quedará destacada en negrita su posición. Si la lista de jugadores es demasiado grande, solo se mostrarán los 5 primeros jugadores, y el resto se podrá ver si se hace click sobre una lista desplegable.

3.7. Mostrar los resultados de la partida

Al finalizar una partida, se mostrarán los resultados de ésta. Los tipos de resultados son

tres:

3.7.1 Mostrar los resultados del jugador.

Para cada jugador, se mostrarán los resultados que ha tenido en la partida: su nº de puntos, el nº de bolas que se ha comido, el nº de jugadores que se ha comido, el nº de jugadores que ha matado con bombas, y el nº de veces que ha muerto.

3.7.2 Mostrar los resultados de la partida.

Se mostrarán los resultados de la partida, es decir, quién ha obtenido la máxima puntuación en la partida en nº de puntos, de bolas comidas, de jugadores comidos, de jugadores matados con bombas y de veces que ha muerto. Para cada resultado, se mostrará el nick del jugador y el valor obtenido.

3.7.3 Mostrar los resultados globales

Se mostrarán los resultados globales. Es decir, se mostrará quién ha obtenido la máxima puntuación, sumando las puntuaciones de todas las partidas ya jugadas, en los apartados: nº de puntos, nº de bolas comidas, nº de jugadores comidos, nº de jugadores matados por bombas y nº de veces que ha muerto. Para cada resultado se mostrará el nick del jugador y el valor del resultado.

3.8. Mostrar la partida redimensionada.

El jugador podrá redimensionar la ventana de juego, y así podrá también redimensionar correctamente la parte del tablero que el jugador puede ver, ajustándose a la proporción de ancho y alto de la ventana y de tamaño.

4. Acciones de los jugadores.

Cada jugador durante la partida podrá realizar una serie de acciones que son las siguientes:

4.1. Mover la posición de su jugador.

Cada jugador podrá mover la posición de su jugador a lo largo del tablero, sin salirse de sus límites. Para mover el jugador, si juega desde un PC, usará para ello el ratón. El jugador se moverá acercándose a la posición del cursor del ratón. Si el cursor está sobre el jugador, no se moverá, y se moverá más rápido conforme el cursor esté más lejos. Si juega desde el smartphone, usará para ello el acelerómetro, es decir, la bola se moverá siguiendo la fuerza de la gravedad.

4.2. Comerse una bola.

Un jugador podrá comerse una bola, incrementando así su área en el área de la bola, aumentando sus puntos en los puntos establecidos por comerse una bola y aumentando

en uno su puntuación en nº de bolas comidas en la partida.

4.3. Comerse a otro jugador.

Un jugador podrá comerse a otro si es más grande que el otro. Al comérselo, crecerá en el área del jugador que se ha comido, sumará a sus puntos los puntos del jugador comido y aumentará en uno las estadísticas de las bolas comidas por el jugador en la partida. El jugador comido morirá, su número de puntos en la partida se pondrá a cero y su número de muertes aumentará en uno.

4.4. Lanzar una bomba.

Cada jugador podrá lanzar una bomba cada cierto tiempo. Para lanzar una bomba, en el PC se pulsa la barra espaciadora, y en el smartphone el botón de menú. Si la bomba choca contra otro jugador o contra otro elemento del tablero, explotará, y se creará una onda expansiva que afectará a los jugadores que estén en su área. Los jugadores que estén en el área de la explosión, disminuirán sus puntos y su tamaño progresivamente mientras estén en el área de explosión, y si disminuye del mínimo de área, morirá, y el jugador que ha lanzado la bomba incrementará su puntuación en uno en cuanto a jugadores que ha matado con el uso de bombas.

4.5. Morir.

Un jugador muere cuando otro jugador se lo come, o cuando su tamaño disminuye del mínimo por la explosión de una bomba. Cuando esto sucede, su puntuación en el número de muertes que ha tenido durante la partida, aumenta en uno. Se le mostrará un mensaje en una ventana indicándole al jugador que ha muerto y un botón para resucitar y continuar en la partida.

4.6. Resucitar.

Un jugador que haya muerto, podrá volver a entrar a la partida pulsando en el botón de regresar a la partida. Si la partida ya finalizó, esperará a entrar en la siguiente partida.

4.7. Chat de voz.

El juego dispone de un chat basado en notas de voz. Cada jugador dispone de un botón que al pulsarlo graba su voz durante un tiempo, y finalizado éste, envía la nota de voz a todos los jugadores conectados. Los jugadores conectados podrán escuchar esta nota de voz a la vez que podrán ver el nick del jugador que les está hablando.

4.8. Aumentar el área del tablero mostrada para el jugador.

Cuando un jugador se hace muy grande, es necesario que el área mostrada del tablero

para el jugador aumente, para que así el jugador pueda seguir jugando y viendo a los demás jugadores. Cuando se vuelve a hacer más pequeño, vuelve a ver el rango de área de antes.

4.9. Salir de una partida.

Un jugador podrá salir de una partida en cualquier momento, pero constará como que no ha participado en la partida a la hora de que la partida finaliza y se guarden los datos sobre la partida de los jugadores que han participado en ella.

5. **Acciones de los bots.**

Los bots son jugadores que se mueven de forma automática, sin la intervención de un ser humano. Los bots podrán realizar las mismas acciones que los jugadores controlados por un humano, pero su comportamiento seguirá un algoritmo. El algoritmo consiste en que si tienen más cerca a una bola que a un jugador, se dirigen a comerse la bola. En cambio, si lo que tienen más cerca es un jugador, si pueden lanzar una bomba, le lanzan una bomba, y si no, intenta comérselo si es de menor radio que él. No obstante, si el jugador que tiene más cerca tiene mayor radio o es una bomba, huyen de él. Los bots muertos, resucitan automáticamente.

6. **Creación, inicio y finalización de partidas.**

El juego está organizado por partidas que tienen una duración determinada. Desde que el juego se inicia, se iniciará una partida, y al finalizar ésta, se guardarán las estadísticas de la partida para cada jugador: nº de puntos, nº de bolas comidas, nº de jugadores comidos, nº de jugadores matados con bombas, y nº de veces que ha muerto. Desde la finalización de una partida hasta el inicio de otra transcurrirá un tiempo determinado. Cuando una partida finaliza, se muestra al usuario los resultados de la partida. Durante la partida, el juego de forma automática irá añadiendo bolas en posiciones aleatorias hasta un máximo, y añadirá un nº de bots determinado en posiciones aleatorias.

2.1.2 Requisitos no funcionales

1. **Ejecución en tiempo real.**

Las acciones del jugador tendrán respuesta en tiempo real, es decir, en el acto. El tiempo que transcurre entre que por ejemplo el jugador manda la orden de mover el jugador y que se visualice el movimiento debe ser casi instantáneo.

2. Apariencia de la interfaz gráfica

2.1. Apariencia de las bolas

Las bolas se mostrarán como un círculo con un color aleatorio.

2.2. Apariencia de los jugadores

Si no tienen avatar, los jugadores se mostrarán como un círculo de un color aleatorio con el nick en su interior. Y si tienen avatar, se mostrará su avatar en lugar del color aleatorio.

2.3. Apariencia de las bombas

Las bombas se mostrarán como un círculo negro con un número que representa los segundos que queda para que explote, y cuando explotan, se mostrarán como una circunferencia con el radio de explosión.

2.4. Lista de jugadores.

La lista de jugadores se situará en la parte superior derecha de la pantalla, y el nick del jugador se mostrará en negrita.

2.5. Cronómetro de la partida

El cronómetro de la partida se situará en la parte superior izquierda de la pantalla y consistirá en dos dígitos representando los minutos que quedan de la partida, seguido de dos puntos (:) y los segundos.

2.6. Botón del chat de voz

Se situará en la parte inferior de la pantalla y mostrará un mensaje indicando que se pulse para comenzar a hablar.

2.7. Grid

El tablero de juego dispondrá de un grid o cuadrícula que permitirá ver mejor el movimiento de los elementos por el tablero.

2.2 Casos de uso

Los casos de uso del software se obtienen a partir de los requisitos obtenidos en el apartado anterior. Uno o más casos de uso se pueden corresponder con un requisito.

A continuación se van a describir los casos de uso del software. Para ello, primero se van a describir los actores que intervienen en los casos de uso, luego se va a exponer un diagrama de casos de uso para cada parte de la funcionalidad del sistema, y cada uno se va a describir textualmente, describiendo la secuencia de interacciones entre los actores en cada caso de uso.

2.2.1 Actores

Los actores que intervienen en el sistema son los siguientes:

- El jugador:
  El jugador representa a cada uno de los usuarios del juego que juegan con él.
- El juego:
  El juego representa al sistema que recibe las órdenes de los jugadores y les responde con lo que piden.
- El tiempo:
  El tiempo interviene en muchos casos de uso porque son activados cuando transcurre cierta cantidad de tiempo.

2.2.2 Diagramas de casos de uso y descripciones textuales

2.2.2.1 Diagrama de casos de uso de las acciones del jugador

El diagrama expuesto a continuación representa el diagrama de casos de uso de las acciones que puede realizar el jugador:

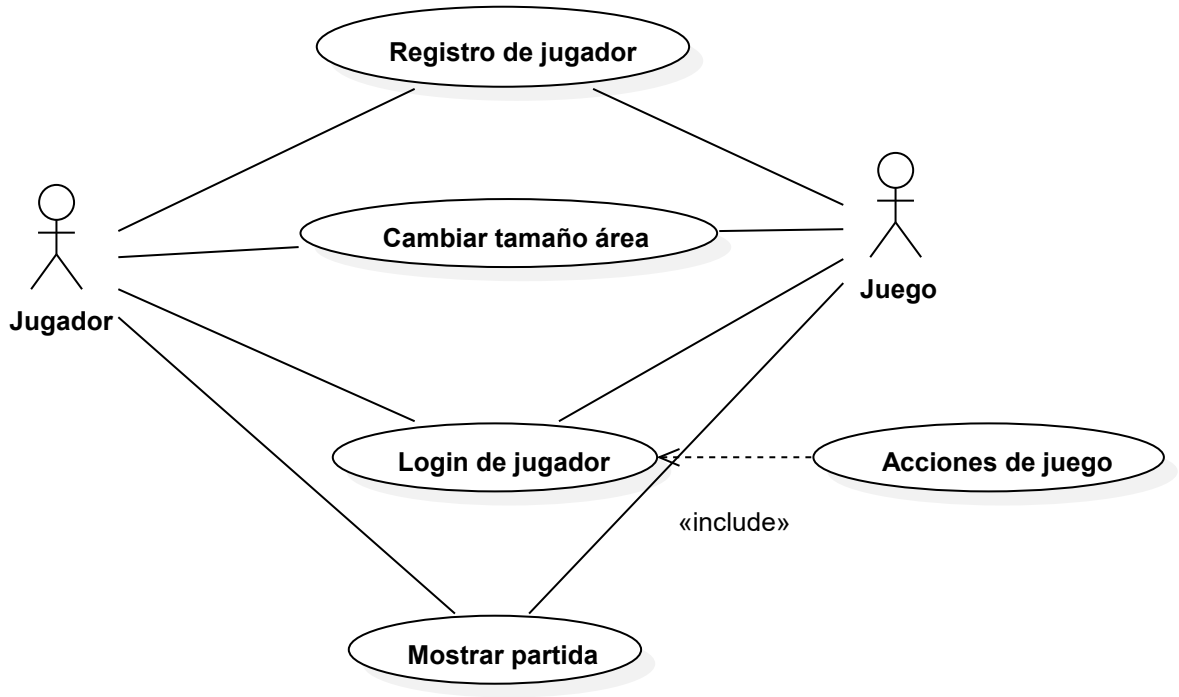


Figura 2.1. Diagrama de casos de uso de las acciones del jugador

Un jugador antes de loguearse en el juego puede registrarse (caso de uso "Registro de jugador") o loguearse (caso de uso "Login de jugador"), o cambiar el tamaño del área visible del tablero que visualiza (caso de uso "Cambiar tamaño área") o mostrar todos los elementos de la partida (caso de uso "Mostrar partida"). Y una vez logueado, puede hacer otras diversas acciones de juego (caso de uso "Acciones de juego"). En todos los casos de uso interviene también el juego. A continuación se exponen la descripción textual de cada uno de los casos de uso anteriores:

Etiqueta: CU-1

Nombre: Registro de jugador

Requisito con el que se corresponde: 1

Descripción:

El jugador se registra en el juego, dado su nick, su contraseña y la imagen de su avatar opcional. O bien modifica su avatar si ya se registró y coinciden el nick y la contraseña.

Actores:

El jugador y el juego.

Precondiciones:

El juego se ha cargado.

Secuencia normal:

1. El jugador introduce en un formulario el nick a registrar, la contraseña y el avatar, y pulsa el botón de enviar estos datos.
2. El juego envía un mensaje al jugador indicándole que espere a que se realice el registro.
3. El juego recibe los datos anteriores y los guarda.
4. El juego envía al jugador un mensaje de que se ha registrado correctamente, cerrando el mensaje de espera al registro.

Secuencia alternativa 1:

1. Pasos 1 y 2 de la secuencia normal.
2. El juego recibe los datos pero el nick ya está registrado y la contraseña no es la misma.
3. El juego envía el mensaje al jugador de que no se puede realizar el registro, cerrando el mensaje de espera al registro.

Secuencia alternativa 2:

1. Pasos 1 y 2 de la secuencia normal.
2. El jugador recibe los datos y el nick ya está registrado y la contraseña es la misma.
3. El juego modifica el avatar registrado por el nuevo introducido.
4. El juego envía al jugador el mensaje de que el registro se ha efectuado correctamente, cerrando el mensaje de espera al registro.

Postcondiciones:

El jugador se ha registrado, o ha modificado su avatar, o no ha hecho nada y ha enviado un mensaje de error al jugador.

Etiqueta: CU-2

Nombre: Login de jugador

Requisito con el que se corresponde: 2

Descripción:

Un jugador se loguea en el juego con su nick y contraseña.

Actores:

El jugador y el juego.

Precondiciones:

El juego se ha cargado.

Secuencia normal:

1. El jugador introduce en un formulario el nick y la contraseña y pulsa el botón de enviar.
2. El juego envía un mensaje al jugador indicándole que espere a que se realice el logueo de su jugador.
3. El juego recibe los datos anteriores y verifica que el jugador con ese nick y esa contraseña existe.
4. El juego cierra el mensaje de espera e introduce al jugador en la partida actual o muestra un mensaje al jugador de que espere a que comience una partida si aún no ha comenzado ninguna.

Secuencia alternativa 1:

1. Pasos 1 y 2 de la secuencia normal.
2. El juego verifica que o el jugador con ese nick no está registrado, o está registrado y no coincide la contraseña enviada, o la partida está en el máximo permitido de jugadores.
3. El juego cierra el mensaje de espera y envía un mensaje al jugador de que no se ha logueado.

Postcondiciones:

El jugador se loguea y entra en la partida actual o espera a la siguiente partida. O bien no se loguea y recibe un mensaje de error.

Etiqueta: CU-3

Nombre: Cambiar tamaño área.

Requisito con el que se corresponde: 3.8

Descripción:

El jugador cambia el tamaño y las dimensiones del área del tablero que puede visualizar.

Actores:

El jugador y el juego.

Precondiciones:

El juego se ha cargado.

Secuencia normal:

1. El jugador redimensiona la ventana de juego, y al hacerlo envía las nuevas dimensiones de su área visible del tablero al juego.
2. El juego establece para el jugador su nuevo área visible del tablero.

Postcondiciones:

El área visible del tablero para el jugador queda cambiada a la nueva dimensión establecida al redimensionar la ventana de juego.

Etiqueta: CU-4

Nombre: Mostrar partida

Requisito con el que se corresponde: 3

Descripción:

Se muestra gráficamente al jugador todos los elementos que hay en un momento de la partida en su área visible del tablero de juego (las bolas, el jugador propio, los otros jugadores, la lista de jugadores ordenada por puntos, las bombas, el cronómetro de la partida y los resultados de la partida).

Actores:

El jugador y el juego.

Precondiciones:

El juego debe haberse cargado.

Secuencia normal:

1. El juego muestra gráficamente y el jugador visualiza los elementos que hay actualmente en la partida: (casos de uso: CU-6 a CU-12).

Secuencia alternativa 1:

1. Como no hay en curso ninguna partida, se muestra al jugador un mensaje de que espere a la siguiente partida si el jugador está logueado.

Postcondiciones:

Se ha mostrado gráficamente la partida al jugador o se ha mostrado un mensaje de que espere a la siguiente partida.

Etiqueta: CU-5
Nombre: Acciones de juego
Requisito con el que se corresponde: 4
Descripción:
El jugador ejecuta una acción de juego una vez que está en la partida, de entre las posibles acciones de juego que hay: cambiar la dirección y velocidad del jugador, chatear por voz, lanzar una bomba, resucitar al jugador y salir de la partida.
Actores:
El jugador y el juego.
Precondiciones:
El jugador se ha logueado en el juego.
Secuencia normal:

1. El jugador ejecuta una acción de juego que se encuentra entre los casos de uso del CU-16 al CU-20.

Postcondiciones:
La acción de juego es ejecutada.

2.2.2.2 Diagrama de casos de uso de mostrar partida

A continuación se expone el diagrama de casos de uso que incluye el caso de uso descrito anteriormente llamado “mostrar partida”. El caso de uso mostrar partida incluye los casos de uso mostrar bolas, mostrar jugador, mostrar jugadores, mostrar lista de jugadores, mostrar bomba, mostrar cronómetro y mostrar resultados. El caso de uso “Mostrar jugadores” incluye al caso de uso “Mostrar jugador” que representa la acción de mostrar el jugador del usuario. Y también incluye al caso de uso “Mostrar bomba” que representa la acción de mostrar una bomba, que en realidad es un jugador de la partida. En todos estos casos de uso participa el actor jugador y el actor juego, además del actor tiempo en el caso de uso “mostrar resultados” porque los resultados solo se muestran cuando transcurre un tiempo. Los casos de uso “mostrar jugador” y “mostrar resultados” requieren del logueo previo del jugador, de ahí que incluyan el caso de uso “Login de jugador”:

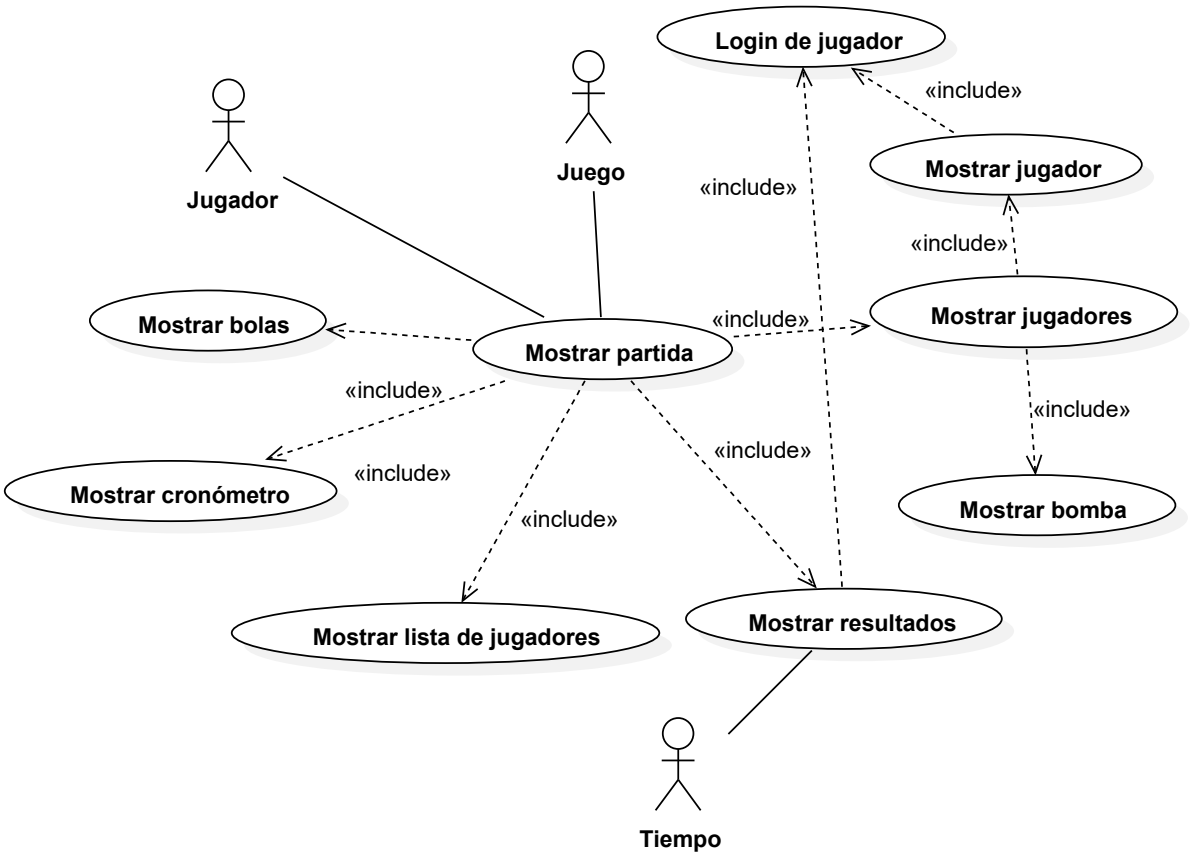


Figura 2.2. Diagrama de casos de uso de mostrar partida

La descripción textual de estos nuevos casos de uso es la siguiente:

Etiqueta: CU-6

Nombre: Mostrar bolas

Requisito con el que se corresponde: 3.1

Descripción:

Muestra gráficamente al jugador las bolas que hay en su área visible del tablero en un momento de la partida.

Actores:

El jugador y el juego.

Precondiciones:

El juego debe haberse cargado.

Secuencia normal:

1. El juego muestra gráficamente las bolas que hay en el área visible del tablero para el jugador.
2. El jugador visualiza el gráfico mostrado.

Postcondiciones:

Se ha mostrado gráficamente las bolas del momento de la partida al jugador.

Etiqueta: CU-7

Nombre: Mostrar jugador

Requisito con el que se corresponde: 3.2

Descripción:

Muestra gráficamente al jugador la representación gráfica de él mismo.

Actores:

El jugador y el juego.

Precondiciones:

El jugador debe haberse logueado.

Secuencia normal:

1. El juego muestra gráficamente la representación gráfica de él mismo en el centro de la pantalla. La representación gráfica será un círculo con el nick del jugador en su interior, y un color de fondo aleatorio para el círculo si el jugador no tiene avatar. Si tiene avatar, el fondo del círculo será la imagen del avatar.
2. El jugador visualiza el gráfico mostrado.

Secuencia alternativa 1:

1. El juego no muestra al jugador porque está muerto.

Postcondiciones:

Se ha mostrado gráficamente la representación de él mismo al jugador, o no se ha mostrado nada.

Etiqueta: CU-8

Nombre: Mostrar jugadores

Requisito con el que se corresponde: 3.3

Descripción:

Muestra gráficamente al jugador los jugadores que hay en el área visible del tablero.

Actores:

El jugador y el juego.

Precondiciones:

El juego debe haberse cargado.

Secuencia normal:

1. El juego muestra gráficamente los jugadores que hay en el área visible del tablero para el jugador implicado.
2. El jugador visualiza el gráfico mostrado.

Postcondiciones:

Se ha mostrado gráficamente los jugadores visibles de la partida al jugador.

Etiqueta: CU-9

Nombre: Mostrar lista de jugadores

Requisito con el que se corresponde: 3.6

Descripción:

Muestra una lista de los jugadores ordenada de mayor a menor por puntos, y si hay varios que tienen los mismos puntos, ordenada de menor a mayor por el nick.

Actores:

El jugador y el juego.

Precondiciones:

El juego se ha cargado.

Secuencia normal:

1. El juego ordena la lista de jugadores de mayor a menor por puntos, y si tienen los mismos puntos, de menor a mayor por su nick.
2. El jugador visualiza la lista de jugadores, con su nick resaltado en negrita.

Postcondiciones:

El jugador visualiza la lista de jugadores ordenada de mayor a menor por puntos.

Etiqueta: CU-10

Nombre: Mostrar bomba

Requisito con el que se corresponde: 3.4

Descripción:

Muestra gráficamente al jugador una bomba que haya en su área visible del tablero.

Actores:

El jugador y el juego.

Precondiciones:

El juego debe haberse cargado.

Secuencia normal:

1. El juego muestra gráficamente una bomba que haya en el área visible del tablero para el jugador implicado. La bomba se mostrará gráficamente como un círculo de color negro en cuyo interior se mostrará el tiempo que queda para que explote en segundos. Si la bomba ha explotado, se mostrará como una circunferencia su onda expansiva.
2. El jugador visualiza el gráfico mostrado.

Postcondiciones:

Se ha mostrado gráficamente una bomba de la partida al jugador.

Etiqueta: CU-11

Nombre: Mostrar cronómetro

Requisito con el que se corresponde: 3.5

Descripción:

Muestra gráficamente al jugador el cronómetro de la partida que indica cuánto tiempo falta para que termine la partida.

Actores:

El jugador y el juego.

Precondiciones:

El juego debe haberse cargado.

Secuencia normal:

1. El juego muestra gráficamente el tiempo actual de la partida como un cronómetro
2. El jugador visualiza el cronómetro.

Postcondiciones:

Se ha mostrado gráficamente el cronómetro al jugador.

Etiqueta: CU-12

Nombre: Mostrar resultados

Requisito con el que se corresponde: 3.7

Descripción:

Muestra los resultados de la partida que acaba de finalizar, divididos en los resultados del jugador, los resultados de la partida, y los resultados globales.

Actores:

El jugador, el juego y el tiempo.

Precondiciones:

El jugador está logueado y hay una partida en curso.

Secuencia normal:

1. Muestra los resultados del jugador, los resultados de la partida y los resultados globales transcurrido el tiempo de la partida (casos de uso CU-13 al CU-15)

Postcondiciones:

El jugador ha visualizado gráficamente los resultados.

2.2.2.3 Diagrama de casos de uso de mostrar resultados

El caso de uso “Mostrar resultados” (CU-12) incluye otros tres casos de uso que se llaman “Mostrar resultados jugador”, “Mostrar resultados partida” y “Mostrar resultados globales”. En todos los casos de uso actúa el jugador, porque es quien visualiza los resultados, el juego que es quien los calcula, y el tiempo que es quien los activa cuando finaliza el tiempo de la partida. El diagrama donde se muestra la relación entre estos casos de uso es el siguiente:

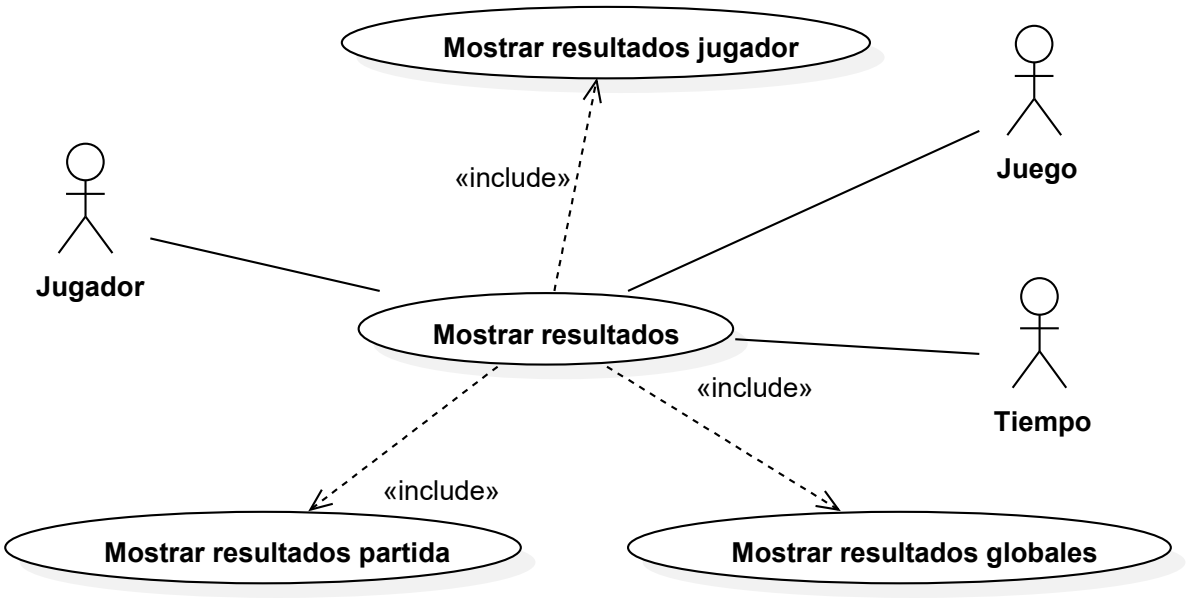


Figura 2.3. Diagrama de casos de uso de mostrar resultados

Las descripciones textuales de cada uno de los nuevos casos de uso son las siguientes:

Etiqueta: CU-13

Nombre: Mostrar resultados jugador

Requisito con el que se corresponde: 3.7.1

Descripción:

Cuando finaliza una partida, muestra los resultados del jugador en ella: nº de puntos, nº de bolas comidas, nº de jugadores comidos, nº de jugadores matados con bombas, y nº de veces que ha muerto.

Actores:

El juego, el jugador y el tiempo.

Precondiciones:

El jugador está logueado y hay una partida en curso.

Secuencia normal:

1. El tiempo de la partida actual finaliza, y se lo comunica al juego.
2. El juego envía los resultados del jugador al jugador. Estos resultados están organizados en nº de puntos, de bolas comidas, de jugadores comidos, de jugadores matados con bombas y de veces que ha muerto.

3. El jugador visualiza gráficamente estos resultados.

Postcondiciones:

El jugador ha visualizado gráficamente sus resultados en la partida.

Etiqueta: CU-14

Nombre: Mostrar resultados partida

Requisito con el que se corresponde: 3.7.2

Descripción:

Muestra los resultados de la partida, es decir, muestra el nick de quién tiene el máximo en las siguientes categorías: nº de puntos, bolas comidas, jugadores comidos, jugadores matados con bombas, y nº de veces que ha muerto.

Actores:

El juego, el jugador y el tiempo.

Precondiciones:

El jugador está logueado y hay una partida en curso.

Secuencia normal:

1. El tiempo de la partida actual finaliza, y se lo comunica al juego.
2. El juego calcula y envía los resultados de la partida al jugador. Estos resultados están organizados en quién tiene el máximo en la partida en nº de puntos, de bolas comidas, de jugadores comidos, de jugadores matados con bombas y de veces que ha muerto.
4. El jugador visualiza gráficamente estos resultados.

Postcondiciones:

El jugador ha visualizado gráficamente los resultados de la partida.

Etiqueta: CU-15

Nombre: Mostrar resultados globales

Requisito con el que se corresponde: 3.7.3

Descripción:

Muestra quién tiene el máximo, sumando todas las partidas jugadas, en las siguientes categorías: nº de puntos, bolas comidas, jugadores comidos, jugadores matados con bombas, y nº de veces que ha muerto.

Actores:

El juego, el jugador y el tiempo.

Precondiciones:

El jugador está logueado y hay una partida en curso.

Secuencia normal:

1. El tiempo de la partida actual finaliza, y se lo comunica al juego.
2. El juego calcula y envía los resultados globales al jugador. Estos resultados están organizados en quién tiene el máximo, sumando los resultados en todas la partidas, en nº de puntos, de bolas comidas, de jugadores comidos, de jugadores matados con bombas y de veces que ha muerto.
3. El jugador visualiza gráficamente estos resultados.

Postcondiciones:

El jugador ha visualizado gráficamente los resultados globales.

2.2.2.4 Diagrama de casos de uso de acciones de juego

Anteriormente se definió un caso de uso llamado “Acciones de juego”. Este caso de uso incluye otros casos de uso más, que son los casos de uso que se corresponden con las acciones que puede realizar el jugador para jugar en el juego una vez que se ha logueado y entrado en la partida. En todos estos casos de uso está involucrado como actor el jugador que es quien los acciona, el juego que es quien recibe las acciones y las ejecuta, y el tiempo también en el caso de uso “Lanzar bomba” y “Chatear por voz”. En el siguiente diagrama se pueden visualizar estos nuevos casos de uso y la relación entre ellos, y a continuación se detalla la descripción textual de cada uno de ellos:

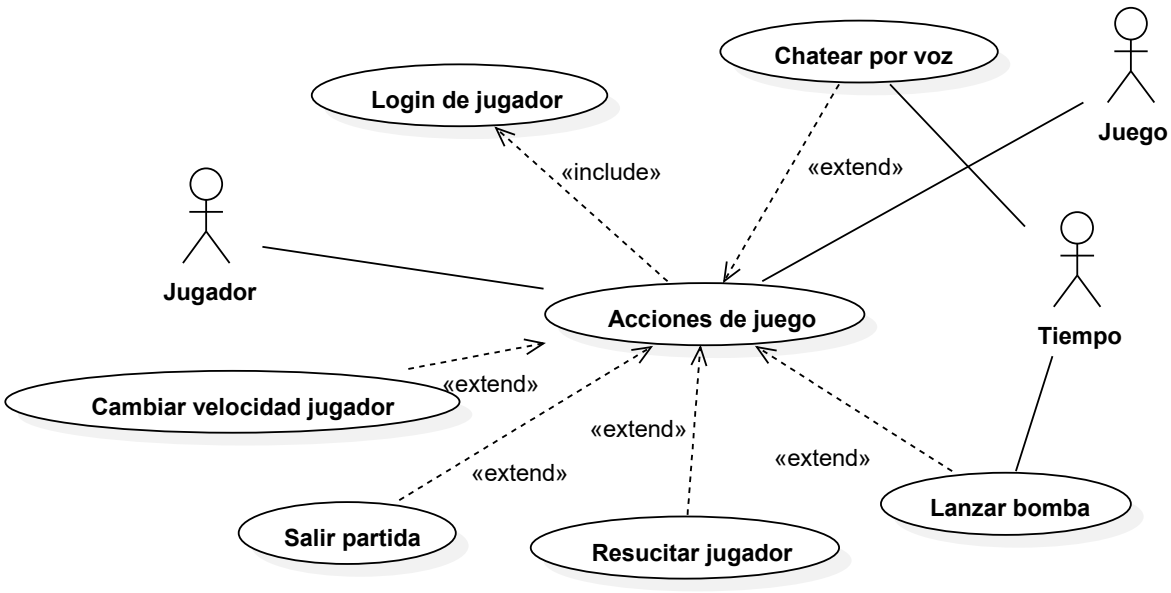


Figura 2.4. Diagrama de casos de uso de acciones de juego

Etiqueta: CU-16

Nombre: Chatear por voz

Requisito con el que se corresponde: 4.7

Descripción:

El jugador podrá grabar su voz durante un tiempo y enviarla a los demás jugadores a través del juego, que escucharán su voz y verán el nick de quién les habla.

Actores:

El jugador, el juego, el resto de jugadores y el tiempo.

Precondiciones:

El jugador debe haberse logueado en el juego.

Secuencia normal:

1. El jugador pulsa el botón de grabar una nota de audio y habla.
2. El juego graba la voz mientras se visualiza una cuenta atrás del tiempo que queda para que termine la grabación.
3. Transcurrido un tiempo, el juego envía la nota de audio grabada a todos los jugadores que estén conectados salvo a él mismo, además del nick del que ha hablado.
4. Los jugadores que han recibido la nota de audio, la escuchan y ven el nick del jugador que ha hablado.

Postcondiciones:

Los otros jugadores conectados al juego escuchan la voz y ven el nick del jugador que ha hablado.

Etiqueta: CU-17

Nombre: Cambiar velocidad jugador

Requisito con el que se corresponde: 4.1

Descripción:

El jugador le indica al juego la velocidad y la dirección que debe tener.

Actores:

El jugador y el juego.

Precondiciones:

El jugador debe haberse logueado.

Secuencia normal:

1. El jugador indica al juego la velocidad y la dirección a la que se quiere mover. En el PC lo indica moviendo el cursor del ratón, mientras más lo aleje de su jugador, mayor velocidad le dará. Y en el dispositivo Android, la velocidad la establecerá el acelerómetro.
2. El juego cambia la velocidad y la dirección del jugador.

Postcondiciones:

El jugador cambia su velocidad y dirección de movimiento.

Etiqueta: CU-18

Nombre: Resucitar jugador

Requisito con el que se corresponde: 4.6

Descripción:

El jugador se resucita a sí mismo después de estar muerto.

Actores:

El jugador y el juego.

Precondiciones:

El jugador debe haberse logueado y estar muerto.

Secuencia normal:

1. El jugador indica al juego que quiere resucitarse a sí mismo, haciendo click en el botón de la ventana que aparece preguntando si quiere continuar en la partida después de morir.
2. El juego pone el estado del jugador a vivo.

Postcondiciones:

El jugador tiene su estado puesto a vivo.

Etiqueta: CU-19

Nombre: Lanzar bomba

Requisito con el que se corresponde: 4.4

Descripción:

Un jugador lanza una bomba.

Actores:

El jugador, el juego y el tiempo.

Precondiciones:

El jugador se ha logueado.

Secuencia normal:

1. El jugador indica al juego que quiere lanzar una bomba pulsando la barra espaciadora si está jugando con un PC o pulsando el botón de menú si está en un dispositivo móvil.
2. El juego comprueba si ha transcurrido el tiempo suficiente para poder lanzar la bomba, y ve que es posible.
3. El juego añade la bomba al juego en una posición cercana al jugador y con la misma velocidad y dirección que llevaba el jugador.

Secuencia alternativa 1:

1. Paso 1 de la secuencia normal.
2. El juego comprueba si ha transcurrido el tiempo suficiente para poder lanzar la bomba, y ve que no es posible porque aún no ha pasado el tiempo.
3. El juego no añade la bomba.

Secuencia alternativa 2:

1. Paso 1 de la secuencia normal.
2. El juego comprueba que la posición en la que el jugador quiere añadir la bomba, está fuera del tablero, así que no la añade y no hace nada.

Postcondiciones:

La bomba se añade al juego o no se añade porque no ha transcurrido el tiempo suficiente o se sale del tablero.

Etiqueta: CU-20

Nombre: Salir partida

Requisito con el que se corresponde: 4.9

Descripción:

El jugador sale de la partida.

Actores:

El jugador y el juego.

Precondiciones:

El jugador se ha conectado al juego.

Secuencia normal:

1. El jugador indica al juego que quiere salir de él.
2. El juego elimina al jugador de su lista de jugadores de la partida.

Postcondiciones:

El jugador queda eliminado de la lista de jugadores de la partida.

2.2.2.5 Diagrama de casos de uso del juego

En este apartado se muestran los casos de usos correspondientes a los que ejecuta el actor Juego en solitario sin la intervención del jugador. El juego por sí solo se carga a sí mismo (caso de uso "Cargar juego"), y al hacerlo carga también los bots que habrá por defecto en todas las partidas (caso de uso "Añadir bots"), por eso el caso de uso "Cargar juego" incluye a "Añadir bots". También, el juego puede iniciar y parar partidas (casos de uso "Iniciar partida" y "Parar partida"), y añadir bolas y hacer jugadas mientras está en una partida (casos de uso "Añadir bola" y "Hacer jugada"). En el siguiente diagrama se muestran los casos de uso anteriores y las relaciones entre ellos, y a continuación una descripción textual de cada uno:

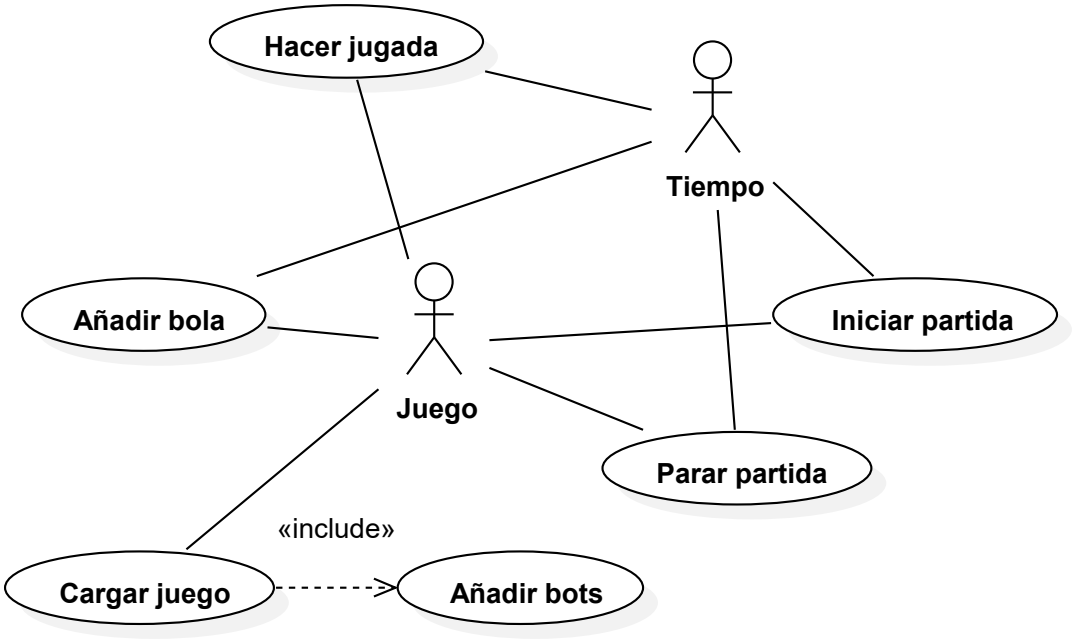


Figura 2.5. Diagrama de casos de uso del juego

Etiqueta: CU-21

Nombre: Cargar juego

Requisito con el que se corresponde: 6

Descripción:

El juego se carga con todos sus datos.

Actores:

El juego.

Precondiciones:

Ninguna.

Secuencia normal:

1. El juego se carga inicializando todos su datos y añadiendo los bots (caso de uso CU-22)

Postcondiciones:

El juego queda cargado.

Etiqueta: CU-22

Nombre: Añadir bots

Requisito con el que se corresponde: 6

Descripción:

Se añaden los bots al juego, que se incorporarán a todas las partidas.

Actores:

El juego.

Precondiciones:

Ninguna.

Secuencia normal:

1. Se añade un número determinado de bots al juego.

Postcondiciones:

Los bots quedan añadidos al juego.

Etiqueta: CU-23

Nombre: Añadir bola

Requisito con el que se corresponde: 6

Descripción:

El juego añade una bola en la partida en una posición aleatoria del tablero.

Actores:

El juego y el tiempo.

Precondiciones:

Hay una partida del juego en curso.

Secuencia normal:

1. Transcurre el tiempo determinado para que se pueda añadir una bola.
2. El juego comprueba si se ha alcanzado el número máximo de bolas establecido.
3. Si no ha alcanzado dicho límite, añade una bola en una posición aleatoria del tablero.

Postcondiciones:

Hay una bola añadida en una posición aleatoria del tablero.

Etiqueta: CU-24

Nombre: Iniciar partida

Requisito con el que se corresponde: 6

Descripción:

Se inicia una nueva partida en el juego.

Actores:

El juego y el tiempo.

Precondiciones:

El juego se ha cargado.

Secuencia normal:

1. Transcurre una cantidad de tiempo determinada desde la finalización de la anterior partida.
2. El juego resetea los jugadores y el contador de tiempo para la finalización de la partida.
3. El juego oculta al jugador la ventana de resultados de la anterior partida y el mensaje que indicaba que esperase a que comenzase la partida, si lo había.

Postcondiciones:

El juego inicia una nueva partida y oculta las ventanas de resultados y de mensaje de espera.

Etiqueta: CU-25

Nombre: Parar partida

Requisito con el que se corresponde: 6

Descripción:

Se para la partida actual y se guardan las estadísticas de la partida.

Actores:

El juego y el tiempo.

Secuencia normal:

1. Transcurre una cantidad de tiempo determinada desde el inicio de la partida.
2. El juego guarda los datos de la partida, e inicia el contador para iniciar la siguiente partida.
3. El juego envía un mensaje al jugador de que espere a que reciba los resultados de la partida que acaba de terminar.

Postcondiciones:

El juego para la partida y las estadísticas de la partida quedan guardadas.

Etiqueta: CU-26

Nombre: Hacer jugada

Requisito con el que se corresponde: 4

Descripción:

El juego cada cierto tiempo calcula una jugada, es decir, para cada jugador, lo mueve, le hace comer bolas, comer otros jugadores, los mata o explota las bombas.

Actores:

El juego y el tiempo.

Precondiciones:

El juego se ha cargado y hay una partida en curso.

Secuencia normal:

1. Transcurre el tiempo determinado entre jugadas del juego.
2. El juego selecciona un jugador, un bot o una bomba de entre la lista de jugadores, y ejecuta una jugada o varias de entre los casos de uso CU-27 al CU-31.
3. Si ha terminado de ejecutar una jugada con todos los jugadores de la partida, terminar el caso de uso. Si no, ir al paso 2.

Postcondiciones:

La jugada del juego es ejecutada.

2.2.2.6 Diagrama de casos de uso de hacer jugada

El caso de uso descrito anteriormente llamado “Hacer jugada” incluye otros casos de uso que se van a describir en este apartado. Cuando el juego hace una jugada, calcula la situación de todos los jugadores en la partida en un instante de tiempo determinado. Para ello, hace la jugada de cada jugador teniendo en cuenta a los otros jugadores, y de esta forma en una jugada del juego cada jugador puede moverse (caso de uso “Mover jugador”), o comer bolas (caso de uso “Comer bolas”), o comer jugadores (caso de uso “Comer jugadores”), explotar las bombas (caso de uso “Explotar bomba”) y matar a uno de los jugadores (caso de uso “Matar jugador”). En el siguiente diagrama se pueden ver estos casos de uso y las relaciones entre ellos. Y a continuación se expone una descripción textual de cada uno de ellos:

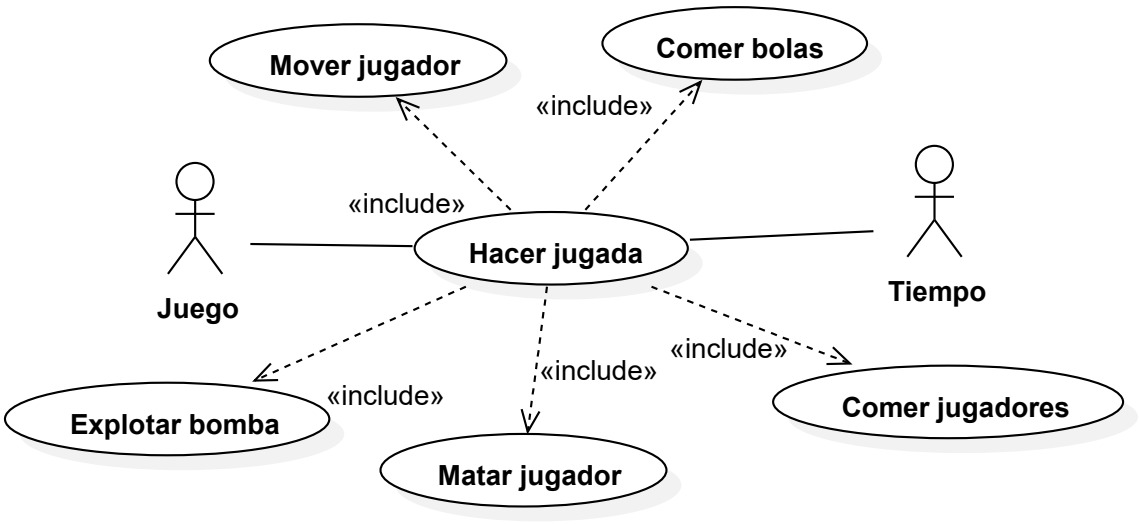


Figura 2.6. Diagrama de casos de uso de hacer jugada

Etiqueta: CU-27

Nombre: Mover jugador

Requisito con el que se corresponde: 4.1

Descripción:

Mueve un jugador según la velocidad y la dirección que tenía establecida.

Actores:

El juego.

Precondiciones:

Hay una partida en curso, y el jugador está vivo.

Secuencia normal:

1. Si el jugador es un bot, calcula su velocidad y su dirección según el algoritmo establecido en los requisitos.
2. El juego calcula la nueva posición del jugador según la velocidad y la dirección que tiene establecidos.
3. El juego cambia la posición del jugador a esta nueva posición.

Secuencia alternativa:

1. Pasos 1 y 2 de la secuencia normal.
2. El juego no cambia de posición al jugador, porque su nueva posición está fuera del tablero.

Postcondiciones:

El jugador a mover se mueve a su nueva posición, excepto si esa nueva posición está fuera del

tablero.

Etiqueta: CU-28

Nombre: Comer bolas

Requisito con el que se corresponde: 4.2

Descripción:

El juego comprueba si un jugador puede comer bolas, y si es así, se las come.

Actores:

El juego.

Precondiciones:

Hay una partida en curso y el jugador implicado está vivo.

Secuencia normal:

1. El juego comprueba si hay bolas que el jugador pueda comerse a su alrededor, y si es así, se las come.
2. El juego elimina las bolas comidas del juego, aumenta el área del jugador en la suma del área de las bolas que se ha comido, aumenta el nº de puntos en la partida en el nº de puntos establecido por comer una bola multiplicado por el nº de bolas comidas y aumenta el valor del nº de bolas comidas.

Postcondiciones:

Las bolas comidas quedan eliminadas del juego, aumenta el área del jugador en el área de la bolas que se ha comido, aumenta el nº de puntos en la partida en el nº de puntos establecido por comer una bola multiplicado por el nº de bolas comidas , y aumenta el valor del nº de bolas comidas.

Etiqueta: CU-29

Nombre: Comer jugadores

Requisito con el que se corresponde: 4.3

Descripción:

El juego comprueba si un jugador puede comerse a otros jugadores, y si es así, se los come.

Actores:

El juego.

Precondiciones:

Hay una partida en curso y los jugadores implicados están vivos.

Secuencia normal:

1. El juego comprueba si para un jugador hay jugadores que pueda comerse a su alrededor, y si es así, se los come. Puede comerse todos los jugadores que estén cerca de él y sean de menor tamaño que él.
2. El juego mata a los jugadores comidos, aumenta el área del jugador en la suma del área de los jugadores que se ha comido, aumenta el nº de puntos en la partida en la suma del nº de puntos que tenían los jugadores comidos y aumenta el valor del nº de jugadores comidos.

Postcondiciones:

Los jugadores comidos están muertos, aumenta el área del jugador en la suma del área de los jugadores que se ha comido, aumenta el nº de puntos en la partida en la suma del nº de puntos que tenían los jugadores que se ha comido , y aumenta el valor del nº de jugadores comidos.

Etiqueta: CU-30

Nombre: Explotar bomba

Requisito con el que se corresponde: 4.4

Descripción:

Una bomba explota si se cumplen ciertas condiciones, y su onda expansiva provoca daños en su área.

Actores:

El juego y el tiempo.

Precondiciones:

Un jugador ha lanzado dicha bomba.

Secuencia normal:

1. El juego comprueba si la bomba toca a una bola, un jugador, o otra bomba.
2. Si no toca a nadie, comprueba si ha transcurrido el tiempo de explosión. Si no ha transcurrido ese tiempo, terminar el caso de uso. Si no, continuar al paso 3.
3. La bomba ataca a las bolas, jugadores y otras bombas que están dentro de su radio de explosión en el juego. Las bolas atacadas se destruyen, los jugadores atacados disminuyen su área y las bombas atacadas explotan también. Si el área de un jugador disminuye del mínimo, éste muere, y aumenta en uno el nº de jugadores matados por bombas del jugador que lanzó esa bomba.
4. Transcurre una cantidad de tiempo.

5. Si el radio de la explosión es mayor de un límite, ir al paso 7. Si no, ir al paso 6.
6. El radio de la explosión aumenta. Ir al paso 3.
7. El juego elimina la bomba.

Postcondición:

Si la bomba ha explotado, queda eliminada del juego, las bolas atacadas quedan también eliminadas del juego, los jugadores atacados o han disminuido su área o han muerto, y las bombas atacadas han explotado.

Etiqueta: CU-31

Nombre: Matar jugador

Requisito con el que se corresponde: 4.5

Descripción:

El juego mata a un jugador que debe estar muerto porque ha sido comido por otro jugador o lo ha matado una bomba.

Actores:

El juego.

Precondiciones:

El jugador que debe morir se encuentra en una o varias de estas condiciones:

1. Un jugador de mayor tamaño se lo ha comido.
2. Ha intentado disminuir del tamaño mínimo cuando la explosión de una bomba le ha afectado.

Secuencia normal:

1. El juego cambia el estado del jugador a muerto.
2. El juego aumenta en uno su nº de muertes en la partida.

Postcondiciones:

El jugador ha muerto y ha aumentado su nº de muertes en la partida en uno.

CAPITULO 3: DISEÑO E IMPLEMENTACIÓN

En el capítulo de análisis se han analizado todas las funcionalidades del software del proyecto.
En este capítulo se va a exponer el diseño y la implementación, es decir, la solución que se ha propuesto para solucionar los problemas planteados en el análisis.

Para diseñar la aplicación, se ha seguido el patrón de diseño cliente/servidor. Es decir, el software se divide en dos partes: una se llama cliente y la otra servidor. La parte cliente se ejecuta en el dispositivo que usan los jugadores para jugar al juego (PC o dispositivo móvil) y la parte servidor se ejecuta en la máquina servidor que normalmente está alojada en un servidor de Internet. Los clientes y los servidores se comunican entre sí usando las comunicaciones de red. Todo esto se puede ver en el siguiente diagrama:

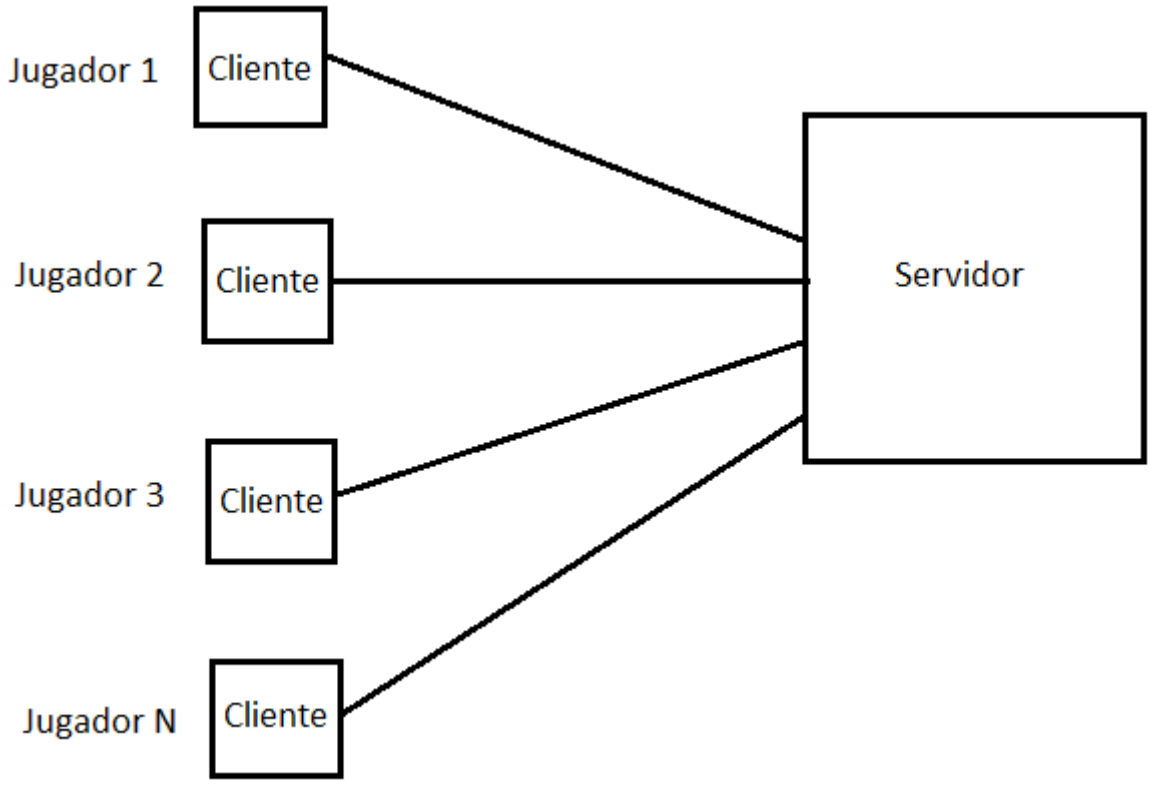


Figura 3.1. Patrón de diseño cliente/servidor

En los subcapítulos siguientes se va a exponer el diseño e implementación de cada una de estas dos partes, divididas en los capítulos “diseño e implementación de la parte servidor” y “diseño e implementación de la parte cliente”. En cada capítulo se hablará primero del diseño de esa parte, explicando la estructura diseñada mediante un diagrama de clases en UML y explicando su dinámica o funcionamiento mediante diagramas de actividad en UML. A continuación se explicará cómo se ha hecho la implementación de ese diseño y qué tecnologías se han usado para implementarlo.

3.1 DISEÑO E IMPLEMENTACIÓN DE LA PARTE SERVIDOR

3.1.1 Diseño de la parte servidor

3.1.1.1 Diagrama de clases

Para explicar el diseño de la parte servidor del software, primero se va a exponer su estructura mediante el siguiente diagrama de clases y explicaciones textuales del mismo:

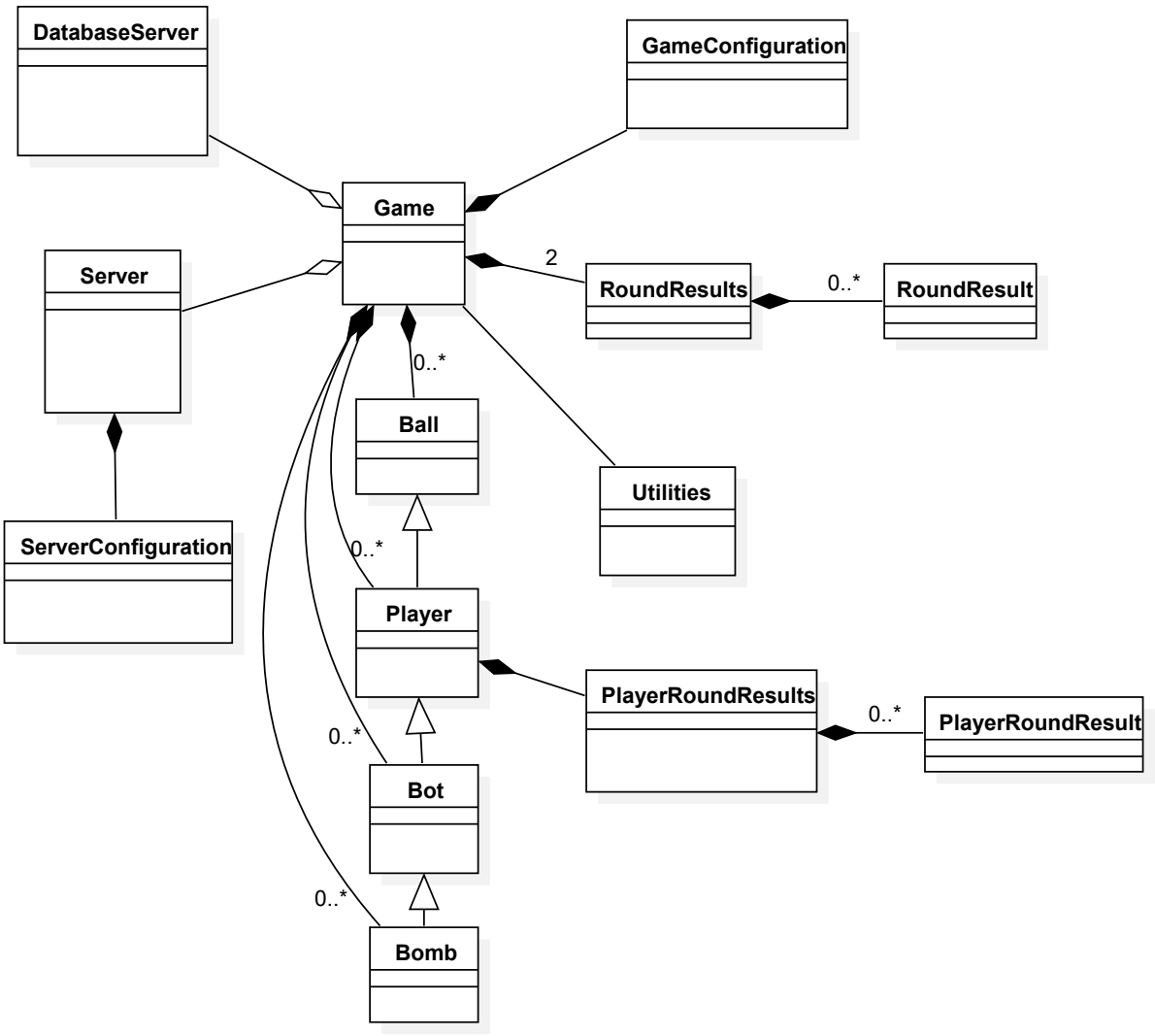


Figura 3.2 Diagrama de clases del diseño de la parte servidor

Explicación textual del diagrama de clases:

Como se puede observar en el diagrama, la clase principal es Game, que representa al actor "Juego" en el diagrama de casos de uso explicado en el capítulo anterior. El juego tiene unos componentes fundamentales, que se representan en UML por la relación de composición, y que son:

- **Su configuración**. Representada por la clase "GameConfiguration". Esta clase contiene los atributos del juego tales como el tamaño de las bolas, de los jugadores, los puntos que se dan al comerse una bola, el tiempo de la partida, etc.
- **Sus bolas y sus jugadores**. Cada juego puede tener ninguna o más bolas (clase Ball) y ninguno o más de un jugador (clase Player para los jugadores activados por humano, clase Bot para los jugadores activados automáticamente y clase Bomb para los jugadores que son bombas). La clase Player hereda de la clase Ball porque un jugador activado por humano en realidad es una bola que se puede mover por un jugador humano. La clase Player se compone mediante la relación de composición en UML, de la clase PlayerRoundResults que representa los resultados que ha tenido ese jugador en la partida actual. A su vez, la clase PlayerRoundResults se compone de cero o más tipos de resultado que vienen representados por la clase PlayerRoundResult. La clase Bot hereda de la clase Player porque un bot es un jugador que solo se diferencia en que se mueve automáticamente. Y la clase Bomb hereda de la clase Bot porque una bomba en realidad es un bot que tiene una forma particular de moverse y de atacar.
- **Sus resultados de partida y sus resultados globales**. Representados por la clase RoundResults. En una partida puede haber cero o más de un tipo de resultado, de ahí que tenga una relación de composición con la clase RoundResult que representa un tipo de resultado de la partida (nº de puntos, de bolas comidas, etc).

Además, el juego se relaciona con otros elementos del servidor como son los siguientes:

- **El servidor web y de websockets**. El juego se comunica con un servidor web y de websockets para poder comunicarse con los clientes, y esto lo hace a través de la clase Server. La clase Server a su vez se compone de una clase ServerConfiguration donde viene la configuración del servidor, como puede ser el número de puerto en el que escuchan las

peticiones de los clientes.

- **El gestor de bases de datos**. El juego se comunica con un gestor de base de datos a través de la clase DatabaseServer.

Aparte, existe una clase llamada Utilities que contiene métodos de utilidades que usa el juego para por ejemplo leer imágenes del sistema de archivos y otras utilidades.

Explicación textual de cada clase:

Clase Game:

Esta clase representa al actor "Juego" especificado en el diagrama de casos de uso explicado en el capítulo de análisis. El juego tiene acceso al servidor para poder comunicarse con el cliente (atributo "_server"), tiene una configuración de sus parámetros (atributo "_gameConfiguration"), el acceso al gestor de base de datos (atributo "_databaseModule"), las bolas, jugadores, imágenes de jugadores, y nº de bots (atributos "_balls", "_players", "_playersImgString", y "_numBots") y también tiene los datos sobre los resultados de la partida actual ("_activeRoundId", "_roundNumPlayers" y "_roundResults") y los datos sobre los resultados globales "_globalRoundResults".

De entre los métodos que tiene, cabe destacar los siguientes, que se corresponden con casos de uso:

- Método "registerPlayer": se corresponde con la gestión que se hace del caso de uso "Registro de jugador" en la parte de servidor.
- Método "loginPlayer": se corresponde con la gestión que se hace del caso de uso "Login de jugador" en la parte de servidor.
- Método "setPlayerWindowSize": se corresponde con la gestión que se hace del caso de uso "Cambiar tamaño área" en la parte de servidor.
- Método "broadcastChatAudio": se corresponde con la gestión que se hace del caso de uso "Chatear por voz" en la parte de servidor.
- Método "exitPlayer": se corresponde con la gestión que se hace del caso de uso "Salir partida" en la parte de servidor.
- Método "addBall": se corresponde con la gestión que se hace del caso de uso "Añadir bola"

en la parte de servidor.

- Método “addBots”: se corresponde con la gestión que se hace del caso de uso “Añadir bots” en la parte de servidor.
- Método “play”: se corresponde con la gestión que se hace del caso de uso “Hacer jugada” en la parte de servidor.
- Método “startRound”: se corresponde con la gestión que se hace del caso de uso “Iniciar partida” en la parte de servidor.
- Método “stopRound”: se corresponde con la gestión que se hace del caso de uso “Parar partida” en la parte de servidor.

Clase GameConfiguration:

Representa la configuración del juego (clase Game). Los atributos que contiene son los valores establecidos, por ejemplo, para el ancho y el alto del tablero de juego (_boardWidth y _boardHeight), la configuración correspondiente a las bolas (radio de las bolas “_ballRadius”, nº máximo de bolas “_maxNumBalls”, tiempo entre la adición de bolas “_addBallTime”, y puntos por comerse una bola “_ballPoints”), la configuración de los jugadores (radio inicial “_playerInitialRadius”, límite de velocidad “_playerSpeedLimit”, nº máximo de jugadores “_maxNumPlayers”, nº máximo de bots “_maxNumBots”), el espacio de tiempo entre jugadas del juego “_playTime”, la configuración de las bombas (radio de las bombas “_bombRadius” y tiempo para que una bomba explote “_bombExplosionTime”) y la configuración de las partidas (tiempo de una partida “_roundTime” y tiempo entre partidas “_nextRoundTime”).

Clase Ball:

Representa una bola en el juego. Se define por su posición en el tablero (atributos _posX y _posY), por su radio (_radius) y su color (_color). Un juego puede tener cero o más bolas, y esto queda representado por la relación que hay entre la clase Game y la clase Ball.

Clase Player:

Representa un jugador accionado por un ser humano. Hereda de la clase Ball porque un jugador en realidad es una bola, con los mismos atributos de posición, de radio y de color, pero además, añade la posibilidad de que se pueda mover por un usuario del juego. Por eso, a los atributos de la clase Ball, añade atributos como el nick (_nick), la ruta de la imagen de su avatar (_imgLocation), el área

del tablero que puede visualizar el usuario de ese jugador (_visibleAreaX1, _visibleAreaY1, _visibleAreaX2 y _visibleAreaY2), la velocidad que lleva en sus componentes horizontal y vertical (_horizontaSpeed y _verticalSpeed), y sus resultados en la partida actual (_roundResults).

Los métodos más relevantes de la clase son:

- Método “accelerate”: se corresponde con la gestión que se hace del caso de uso “Cambiar velocidad jugador” en la parte de servidor.
- Método “resurrect”: se corresponde con la gestión que se hace del caso de uso “Resucitar jugador” en la parte de servidor.
- Método “play”: se corresponde con la gestión del caso de uso “Hacer jugada” pero para un jugador en concreto. Es decir este método efectúa una jugada para este jugador, haciendo uso de los métodos “move” (caso de uso “Mover jugador”), “attackBall” (caso de uso “Comer bolas” pero para comer sólo una bola), “attackPlayer” (caso de uso “Comer jugadores” pero para comer sólo un jugador), “attackBot” (lo mismo que el anterior pero para el caso de que el jugador sea un bot) y “kill” (caso de uso “Matar jugador”).
- Método “throwBomb”: se corresponde con la gestión del caso de uso “Lanzar bomba” en la parte de servidor.

Clase Bot:

Esta clase representa a un bot. Un bot es un jugador que está accionado automáticamente sin la intervención de un ser humano. Por eso esta clase hereda de la clase “Player” porque un bot es un jugador con la diferencia de que se mueve automáticamente. Por lo tanto, sobrescribe los métodos de Player que podía ejecutar un ser humano, como por ejemplo “accelerate” (caso de uso “Cambiar velocidad jugador”), “play” (caso de uso “Hacer jugada”) y “throwBomb” (caso de uso “Lanzar bomba”).

Clase Bomb:

Esta clase representa una bomba. Una bomba en realidad es un bot, porque se mueve y actúa de forma autónoma sin la intervención humana, pero tiene un comportamiento diferente al de un bot. Por eso esta clase hereda de la clase Bot y sobrescribe algunos métodos: los referentes a cómo se mueve, cómo ataca a los demás elementos del juego y cómo explota. Sobrescribe el método

"accelerate" (caso de uso "Cambiar velocidad jugador"), los métodos "attackBall", "attackPlayer" y "attackBot" (casos de uso "Comer bolas" y "Comer jugadores") y el método "play" (caso de uso "Hacer jugada"). Y añade otro método propio de las bombas que es "explode" y que se corresponde con el caso de uso "Explotar bomba".

Clase RoundResults:

Esta clase representa los resultados de la partida actual en el juego. Por eso es un atributo de la clase Game. Es un conjunto de resultados de la partida, donde un resultado de la partida es un objeto de la clase RoundResult.

Clase RoundResult:

Esta clase representa un resultado en la partida actual en el juego.

Clase PlayerRoundResults

Esta clase es similar a RoundResults, porque representa los resultados de la partida en un juego, pero para un jugador en concreto. Por eso es atributo de la clase Player. Es un conjunto de resultados en la partida, donde cada resultado es un objeto de la clase PlayerRoundResult.

Clase PlayerRoundResult:

Esta clase representa un resultado de la partida para un jugador.

Clase DatabaseServer:

Representa la clase que permite el acceso al gestor de base de datos. Los métodos que contiene son para registrar jugadores en la base de datos, buscarlos y guardar y cargar los resultados de las partidas de los jugadores.

Clase Server:

Representa la clase que permite el acceso al servidor web y de websockets.

Clase ServerConfiguration:

Representa la configuración del servidor, representada por la clase Server. El atributo que contiene es el puerto donde el servidor escucha las conexiones.

Clase Utilities:

Contiene cuatro métodos usados por las demás clases para leer y escribir imágenes (métodos “readImage” y “writeImage”), calcular la distancia entre dos puntos (método “euclideanDistance”) y para generar un color aleatorio en formato hexadecimal (método “randomColor”).

3.1.1.2 Diagramas de actividad

En este apartado se va a describir la dinámica o funcionamiento de la parte de servidor, con la ayuda de diagramas de actividad en UML. En primer lugar se presenta el siguiente diagrama que describe la dinámica del funcionamiento del juego en su parte servidor de forma general, es decir, en el nivel de abstracción más alto:

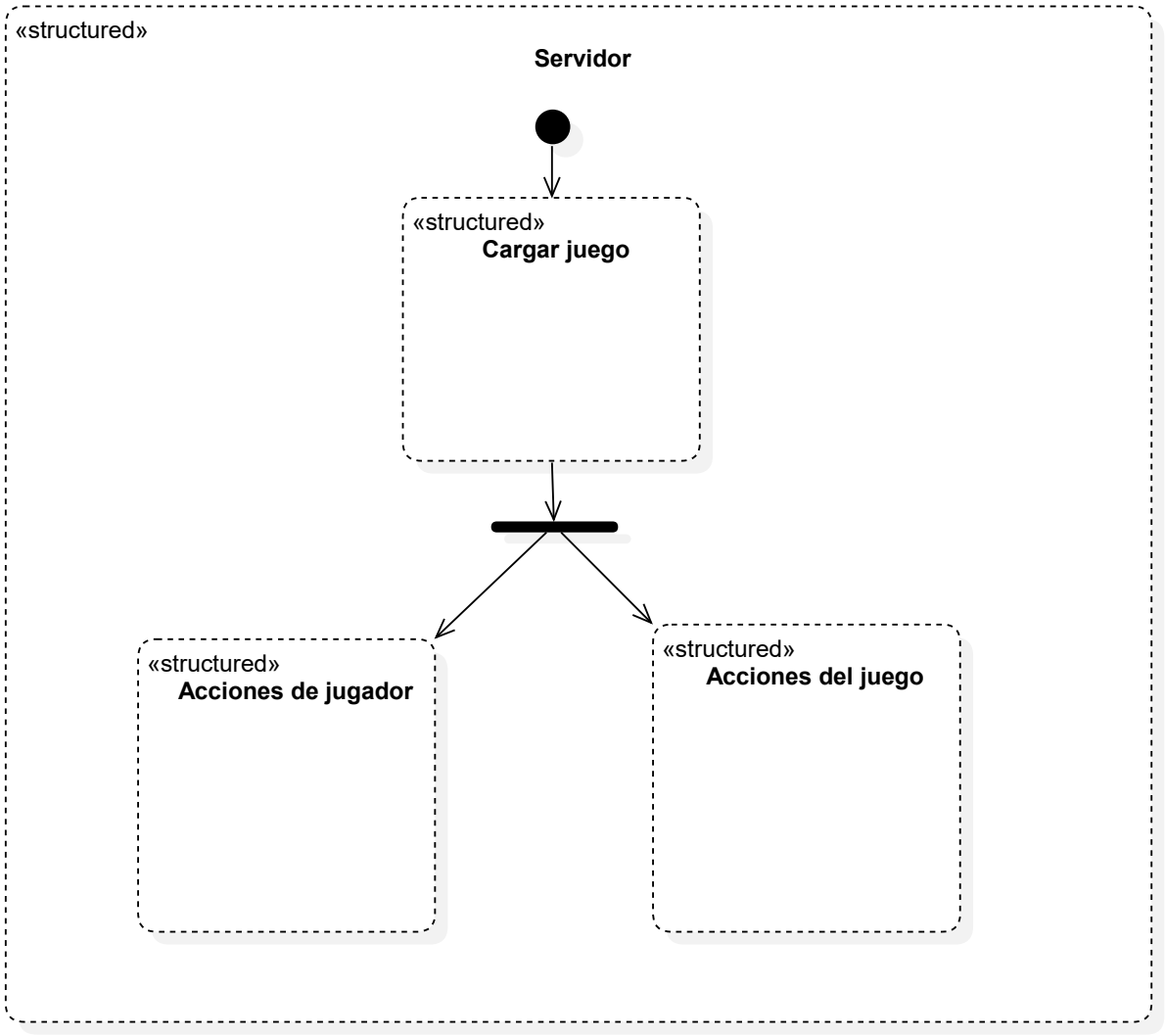


Figura 3.3 Diagrama de actividad general del servidor

Como se puede observar, el funcionamiento del juego en el servidor empieza con la actividad "Cargar juego". Esta actividad coincide con el caso de uso del mismo nombre, en el que se carga el juego. De esta actividad parten dos ramas de actividades que se pueden ejecutar a la vez: "Acciones de jugador" y "Acciones del juego". "Acciones de jugador" contiene la dinámica de los casos de uso en los que interviene el actor "jugador" y "Acciones del juego" contiene la dinámica de los casos de uso en los que solo interviene el actor "juego".

Se empieza explicando la actividad "Cargar juego" con un diagrama de actividad y una breve explicación:

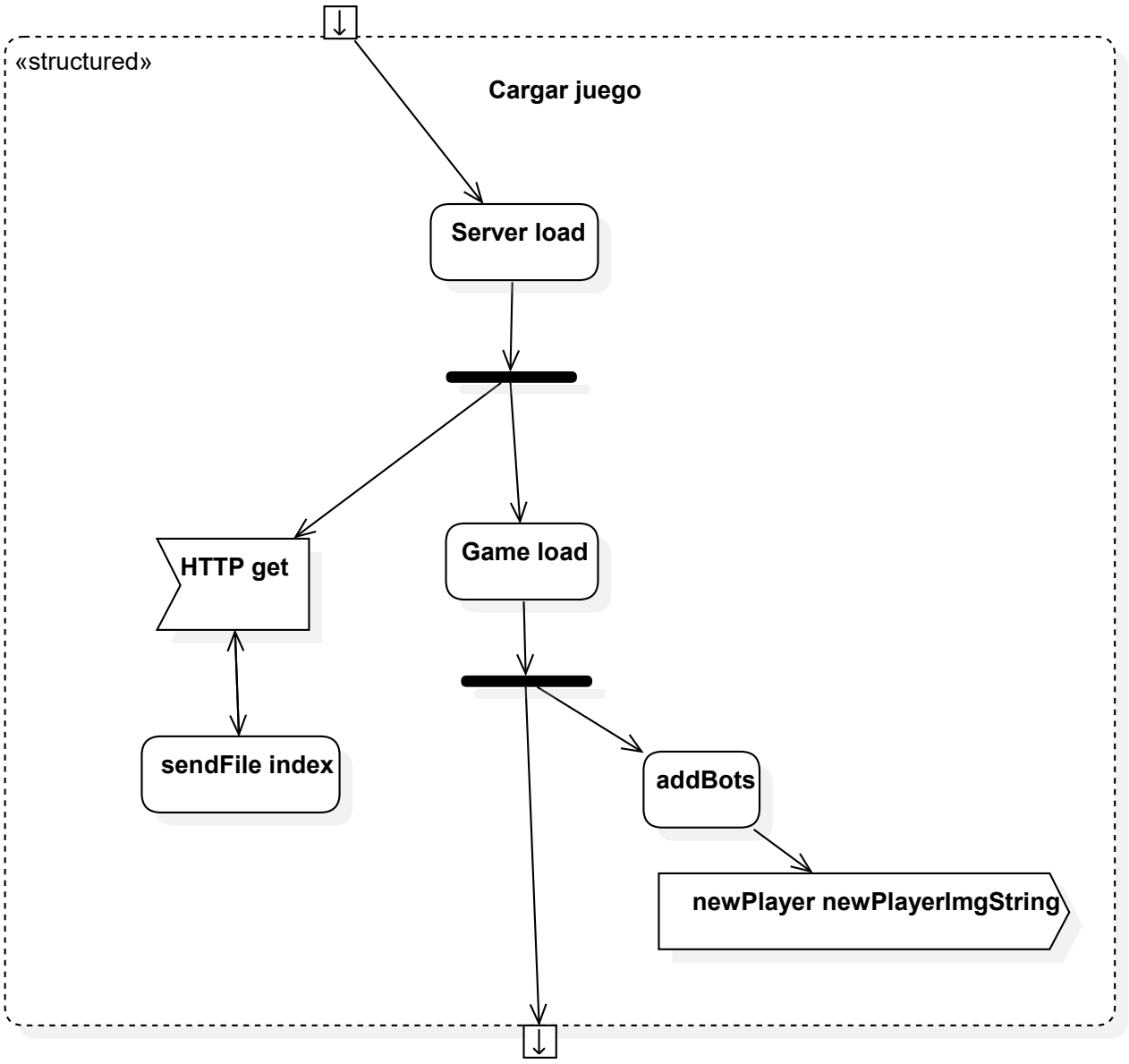


Figura 3.4 Diagrama de actividad de cargar juego

En el diagrama se puede ver que lo primero que se hace es cargar el servidor web (método load de la clase Server). Luego, crea una rama de ejecución que se activa al recibir un mensaje de la parte cliente de conexión al servidor web (en el diagrama, HTTP get). Si en algún momento recibe una petición HTTP, se enviaría la parte cliente del software al usuario o jugador que hizo la petición (método sendFile de la clase Server), y volvería a esperar a recibir una petición HTTP para volver a hacer lo mismo cuando se recibiese de nuevo.

La ejecución continúa con la carga de los elementos del juego (método load de la clase Game) y se crea una nueva rama de ejecución en la cual se añaden los bots (método addBots de la clase Game). Cuando se añaden bots, se envían mensajes al cliente indicándole qué bots se han añadido (mensajes "newPlayer" y "newPlayerImgString").

A partir de aquí la ejecución continúa con la ejecución en dos ramas donde en cada una se ejecutan las "Acciones del jugador" y las "Acciones del juego" respectivamente, como se vio en el diagrama de actividad general de la parte servidor. A continuación se describe la actividad "Acciones de jugador" con un diagrama y una breve descripción textual:

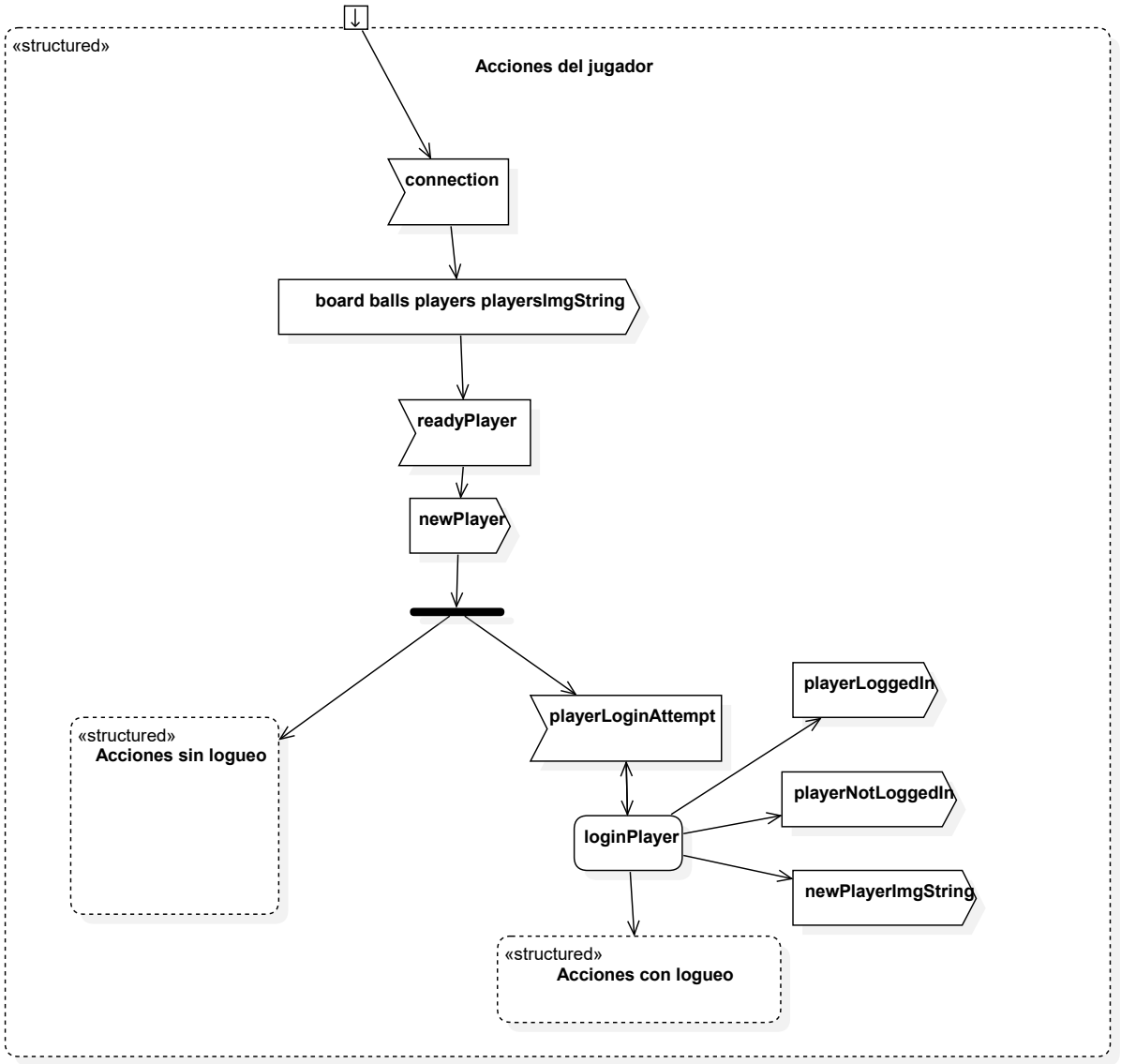


Figura 3.5 Diagrama de actividad de acciones del jugador

Las acciones del jugador son todas las acciones que puede realizar un jugador cuando se conecta al servidor del juego. Por eso, tal y como se puede ver en el diagrama, lo primero que se hace es esperar a la recepción de un mensaje de conexión de un jugador (mensaje “connection” en el diagrama).

Una vez que ha recibido la conexión de un jugador, envía los datos del tablero, bolas y jugadores a través de mensajes al cliente del usuario que ha hecho la conexión (mensajes “board”, “balls”, “players” y “playersImgString” en el diagrama). Luego se espera a recibir la orden del usuario de que está preparado para jugar (mensaje “readyPlayer”). Una vez que la ha recibido, añade el jugador al juego todavía sin loguear, indicándolo con un mensaje al cliente (mensaje “newPlayer”).

Después de hacer esto, se crean dos ramas de ejecución que se pueden ejecutar a la vez. En una de ellas se ejecutan las “Acciones sin logueo” que son las acciones que puede realizar el usuario en el juego sin haberse logueado todavía como jugador. Y en la otra de las ramas se espera a recibir un mensaje del cliente de intento de logueo de un jugador (mensaje “playerLoginAttempt”). Cuando se recibe este mensaje, intenta loguear al jugador en el juego (método “loginPlayer” de la clase Game). Si el logueo es correcto, envía dos mensajes al cliente: uno indicando que se ha logueado correctamente (mensaje “playerLoggedIn”) y otro con la imagen de su avatar (mensaje “newPlayerImgString”). Después de enviar estos dos mensajes, ejecuta las “Acciones con logueo”, que son las acciones que puede realizar el jugador una vez logueado en el juego. En cambio, si el logueo es incorrecto, se envía un mensaje al cliente de que no se ha podido loguear (mensaje “playerNotLoggedIn”).

A continuación se describen cada una de las dos actividades estructuradas que aparecen en el diagrama: “Acciones sin logueo” y “Acciones con logueo”.

Veamos primero las “Acciones sin logueo”:

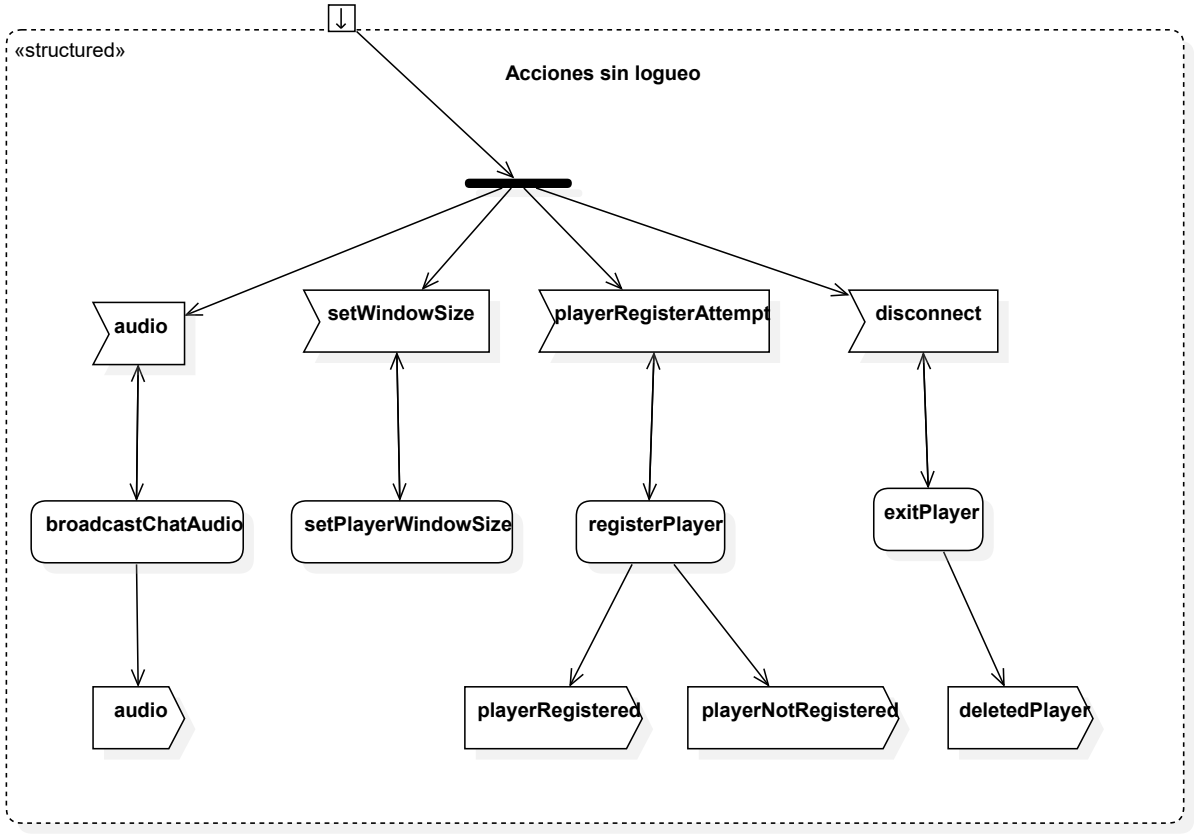


Figura 3.6 Diagrama de actividad acciones sin logueo

El diagrama representa las acciones que puede realizar el jugador sin necesidad de loguearse.
Cada acción coincide con un caso de uso y a la vez con una rama de ejecución que se ejecuta cuando recibe el mensaje correspondiente enviado por el cliente/jugador. Cuando una acción se ejecuta, vuelve a esperar por la recepción del mismo mensaje y a hacer lo mismo cuando lo vuelve a recibir.

Si recibe el mensaje “audio” se ejecuta el caso de uso “Chatear por voz” (método “broadcastChatAudio” de la clase Game), enviando el audio recibido al resto de clientes/jugadores con un mensaje. Lo mismo ocurre cuando recibe el mensaje “setWindowSize” y ejecuta el caso de uso “Cambiar tamaño área” (método “setPlayerWindowSize” de la clase Game), cuando recibe el mensaje “playerRegisterAttempt” y se ejecuta el caso de uso “Registro de jugador” (método “registerPlayer” de la clase Game), y cuando recibe el mensaje “disconnect” y se ejecuta el caso de uso “Salir partida” (método “exitPlayer” de la clase Game).

Cuando se ejecuta el caso de uso “Registro de jugador” se pueden enviar los mensajes “playerRegistered” que indica al cliente que el jugador se ha registrado correctamente, y “playerNotRegistered”, que indica al cliente que el jugador no se ha registrado correctamente. Y cuando se ejecuta el caso de uso “Salir partida”, se envía el mensaje “deletedPlayer” que indica al cliente que un jugador ha salido de la partida.

Ahora veamos las “Acciones con logueo”:

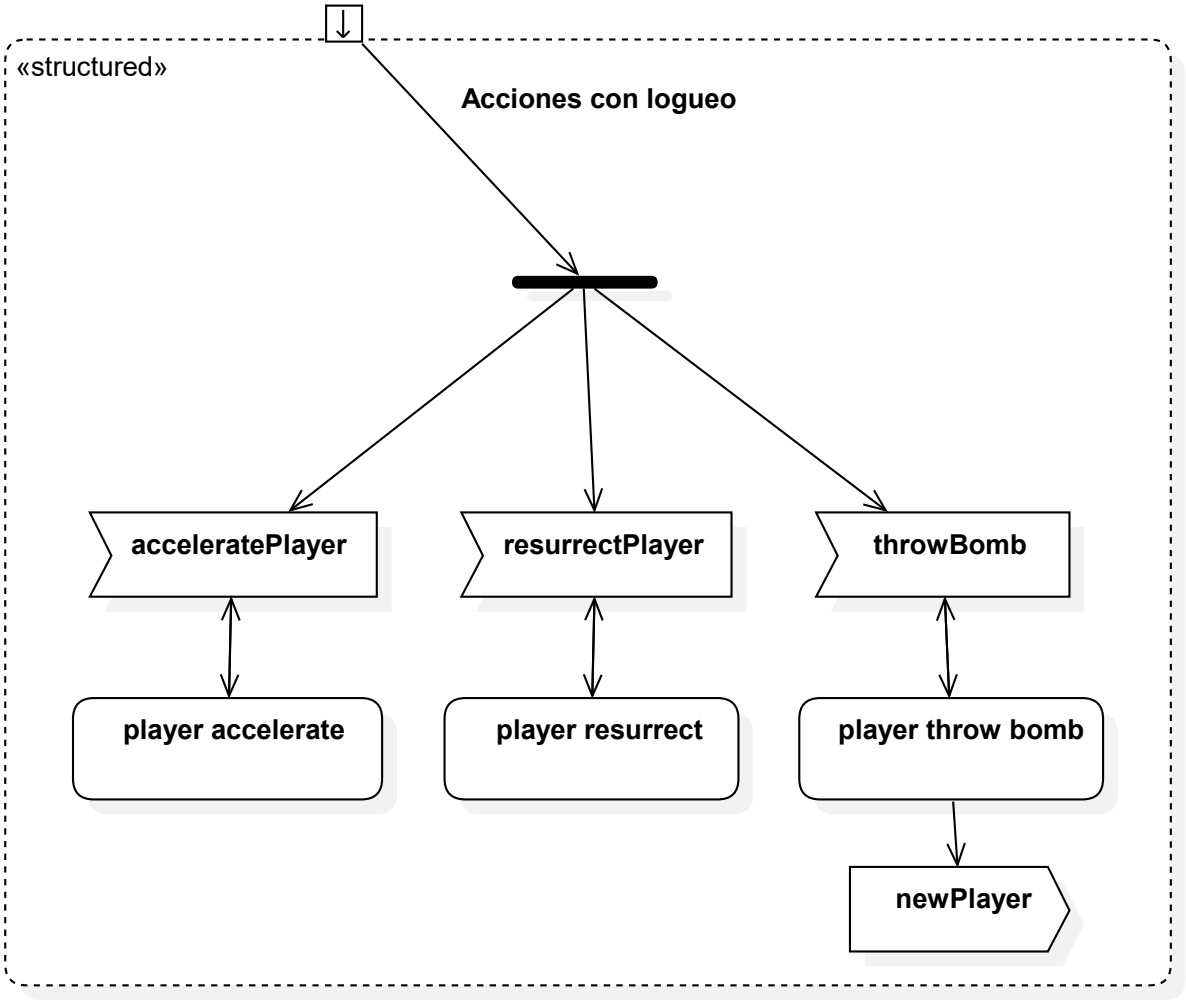


Figura 3.7 Diagrama de actividad acciones con logueo

En este diagrama se puede visualizar cómo es la ejecución de las acciones del jugador que requieren logueo. Igual que en el caso anterior, cada acción se corresponde con un caso de uso y se ejecuta en una rama de ejecución distinta. Cada acción se ejecuta cuando recibe el mensaje correspondiente por parte del cliente/jugador, y una vez que se ejecuta, vuelve a esperar a recibir el mismo mensaje

y a hacer lo mismo cuando lo recibe.

El mensaje “acceleratePlayer” ejecuta la acción que corresponde al caso de uso “Cambiar velocidad jugador” (método “accelerate” de la clase Player), el mensaje “resurrectPlayer” ejecuta la acción correspondiente al caso de uso “Resucitar jugador” (método “resurrect” de la clase Player), y el mensaje “throwBomb” ejecuta la acción correspondiente al caso de uso “Lanzar bomba” (método “throwBomb” de la clase Player) que envía el mensaje “newPlayer” al cliente/jugador indicándole que se ha añadido un nuevo jugador (la bomba añadida).

Por último veamos la actividad “Acciones del juego”:

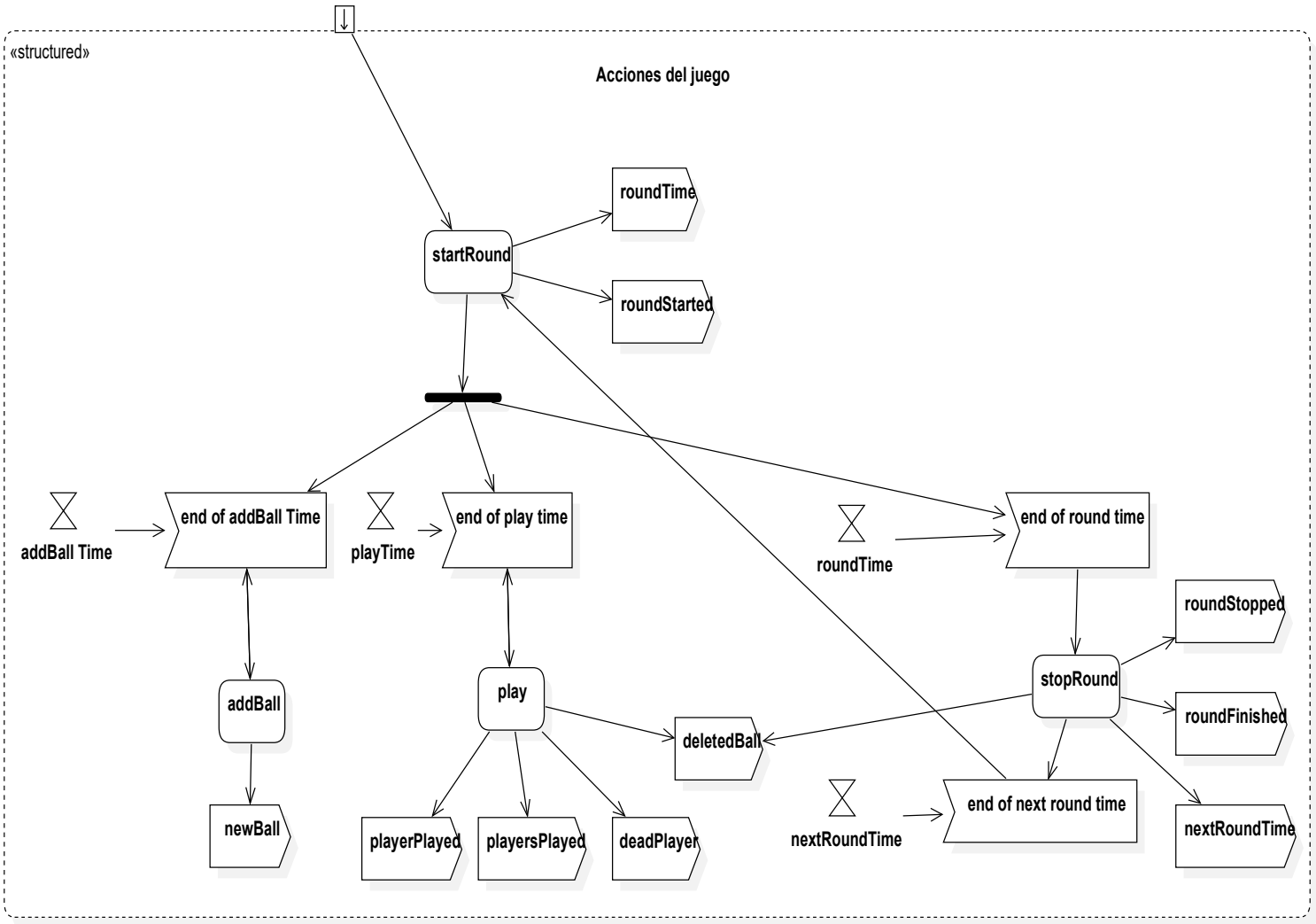


Figura 3.8 Diagrama de actividad de acciones del juego

En este diagrama se puede observar que el juego en primer lugar lo que hace es iniciar una partida (caso de uso “Iniciar partida” y método “startRound” de la clase Game). Cuando inicia una partida, envía continuamente un mensaje al cliente en el que consta el tiempo que queda para la finalización de la partida (mensaje “roundTime”), a la vez que envía el mensaje avisando al cliente de que la partida ya ha comenzado (mensaje “roundStarted”).

Una vez hecho esto, crea tres ramas de ejecución que se pueden ejecutar a la vez. Cada rama de ejecución corresponde con la ejecución de un caso de uso por parte del actor Juego.De forma diferente a lo que ocurría en las anteriores actividades, las acciones del juego no se activan por mensajes del cliente, porque son acciones en las que no interviene el jugador, sino solo el juego y el tiempo. Así que las acciones se activan por el tiempo.

El evento “end of addBall Time“ representa la finalización del tiempo que el juego espera para añadir una bola, así que cuando ese evento ocurre, ejecuta el caso de uso “Añadir bola” (método addBall de la clase Game). Una vez ejecutado, vuelve a esperar a recibir el evento y a ejecutar lo mismo cuando se vuelve a recibir. El evento “end of play time” representa la finalización del tiempo que el juego espera para hacer una jugada, y cuando sucede este evento, ejecuta el caso de uso “Hacer jugada” (método “play” de la clase Game). Cuando se ejecuta una jugada, se pueden enviar distintos mensajes al cliente avisando de que una bola ha sido destruida por una bomba o comida (mensaje “deletedBall”), o de que un jugador ha sido matado por una bomba o comido por un jugador (mensaje “deadPlayer”). Cuando un jugador ha ejecutado su jugada, se avisa al cliente con el mensaje “playerPlayed” y cuando todos los jugadores han ejecutado su jugada, avisa al cliente con el mensaje “playersPlayed”. Una vez ejecutada una jugada, vuelve a esperar a recibir el mismo evento y a ejecutar lo mismo cuando lo recibe. Por último, la finalización del tiempo de la partida (evento “end of round time”) activa la ejecución del caso de uso de parar la partida (método “stopRound” de la clase Game), el cual envía mensajes al cliente para avisar de que la partida ha finalizado (mensajes “roundStopped” y “roundFinished”) y envía continuamente el mensaje “nextRoundTime” que indica al cliente el tiempo que falta para que empiece una nueva partida. Cuando termina el tiempo de espera para la siguiente partida (“end of next round time”) vuelve a ejecutar al caso de uso “Iniciar partida” (método “startRound” de la clase Game), el cual inicia otra partida.

3.1.2 Implementación de la parte servidor y tecnologías utilizadas

En este apartado se va a explicar cómo se ha implementado el diseño de la parte servidor del videojuego explicado en el apartado anterior y cómo se relaciona con otros elementos de la parte servidor. Para ello, primero se va a exponer el diagrama UML del despliegue, que representa la implementación de todo el software que se ejecuta en la parte servidor:

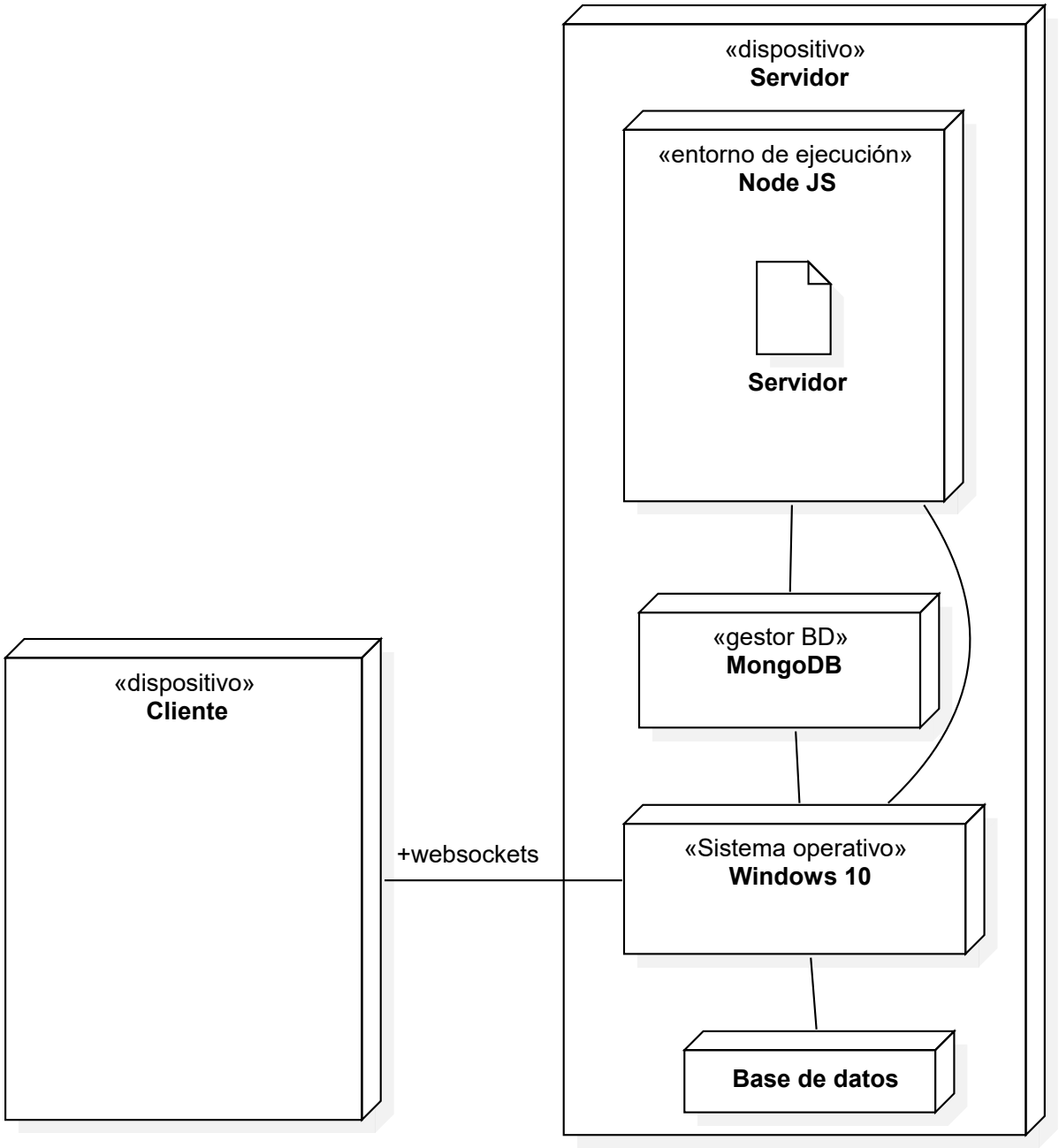


Figura 3.9 Diagrama de despliegue de la parte servidor

Como se puede ver en el diagrama, existe un nodo dispositivo que es la máquina donde se ejecuta todo el software de la parte servidor. Dentro del dispositivo existen otros nodos. Uno de ellos es el sistema operativo, que se trata de Windows 10, y que es desde donde se envían y reciben los mensajes que van desde el cliente y hacia el cliente, utilizando la tecnología de los websockets. También desde el sistema operativo se accede a la base de datos y al sistema de archivos. Con el sistema operativo se comunican el entorno de ejecución de Node JS y el gestor de bases de datos MongoDB. El gestor de bases de datos se comunica con el sistema operativo para acceder a la base de datos. Y a su vez, el gestor de base de bases de datos y el sistema operativo se comunican con el entorno de ejecución Node JS.

A continuación en los siguientes subapartados se van a explicar los siguientes elementos del diagrama:

- El artefacto Servidor y el entorno de ejecución Node JS.
- La base de datos y el gestor de base de datos MongoDB.

3.1.2.1 El artefacto Servidor y el entorno de ejecución Node JS

Node js es un entorno de ejecución para el lenguaje de programación JavaScript, que sirve para ejecutar programas escritos en este lenguaje en la parte de servidor. El código escrito en JavaScript para Node Js se estructura en paquetes, y dentro de cada paquete, en módulos. Node Js ya viene con algunos módulos incluidos por defecto para por ejemplo acceder al sistema de ficheros. El artefacto Servidor que se puede ver en el diagrama de despliegue de la parte servidor, es un paquete de Node JS que contiene el software del videojuego en su parte servidor del cual se explicó su diseño en el capítulo de diseño anterior, y que se divide a su vez en varios paquetes que se muestran en el siguiente diagrama:

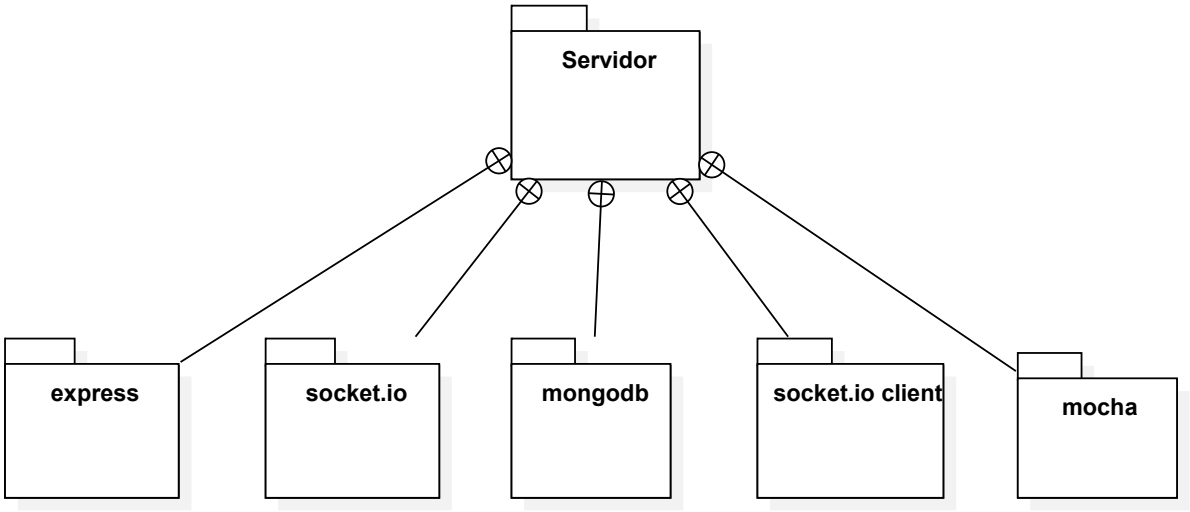
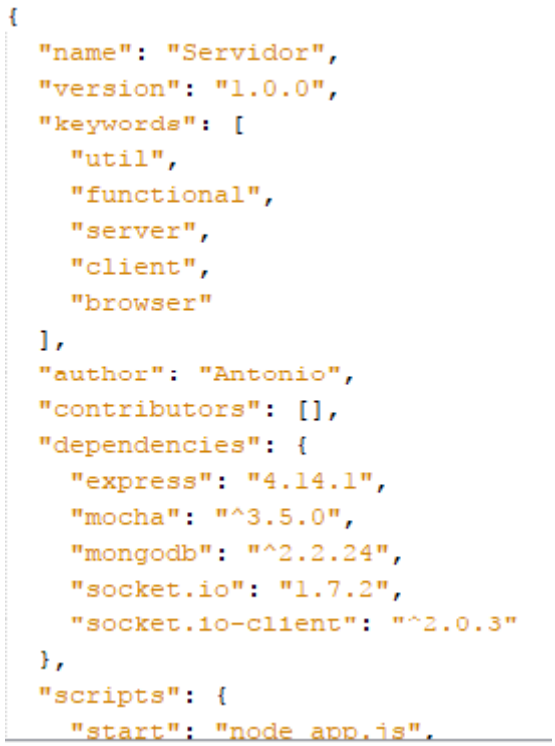


Figura 3.10 Diagrama de paquetes del artefacto Servidor

La descripción del paquete viene en un archivo que acompaña a todos los paquetes Node Js que se llama "package.json". Es un archivo escrito en lenguaje JSON (JavaScript Object Notation), es decir, escrito en la forma en la que se escriben los objetos en JavaScript. En la siguiente imagen se puede visualizar el contenido del archivo "package.json":



```
{
  "name": "Servidor",
  "version": "1.0.0",
  "keywords": [
    "util",
    "functional",
    "server",
    "client",
    "browser"
  ],
  "author": "Antonio",
  "contributors": [],
  "dependencies": {
    "express": "4.14.1",
    "mocha": "^3.5.0",
    "mongodb": "^2.2.24",
    "socket.io": "1.7.2",
    "socket.io-client": "^2.0.3"
  },
  "scripts": {
    "start": "node app.js",
```

Figura 3.11 Descripción del paquete Node JS en el archivo package.json

Como se puede observar, contiene un atributo llamado “dependencies”. Ahí están escritos los nombres de los paquetes que usa el paquete. Estos paquetes se pueden descargar de Internet usando una herramienta que viene con la instalación de Node Js, que se llama “NPM” (Node Packet Manager). La forma de usarlo es escribir en una línea de comandos “npm install nombredelpaquete –save”. Esto instala “nombredelpaquete” como dependencia en el paquete guardado en la carpeta en la que se ejecuta el comando.

Como se puede ver en el diagrama anterior, este paquete contiene los paquetes “express” (que es el framework Express donde está implementado el servidor web del que se hablará más adelante), “socket.io” (que es un framework para usar websockets en el servidor web Express), “mongodb” (que es un framework para acceder al gestor de base de datos mongoDB desde Node Js), socket.io-client, que se usa para hacer pruebas del software simulando a la parte cliente y mocha que es un paquete para hacer tests en JavaScript.

Dentro de cada paquete, éste está dividido en módulos. Un módulo contiene un fichero escrito en javascript y puede usar otros módulos, y un fichero en javascript solo puede pertenecer a un módulo. Además, un fichero en JavaScript debería contener la descripción de una clase (prototipo en JavaScript).

De esta forma, en nuestro caso el paquete Servidor, además de contener los paquetes mencionados anteriormente, contiene módulos en los que cada uno de ellos corresponde con una clase especificada en el diseño del software. Un ejemplo sería la clase/prototipo Ball. Esta clase se define en un solo archivo escrito en JavaScript llamado Ball.js. Al final de este archivo, se incluye la siguiente línea de código: module.exports = Ball; . Esto convierte a Ball en un módulo que se puede usar en otros módulos del paquete, o de paquetes que contengan a este paquete.

A continuación se va a explicar cómo se ha implementado este artefacto Servidor, explicando en primer lugar el servidor web y de websockets que está contenido dentro de él y explicando en segundo lugar el lenguaje de programación JavaScript en el cual se ha programado y su modelo de ejecución.

3.1.2.1.1 El servidor web y de websockets

Para la implementación de la parte de servidor del videojuego es importante disponer de un servidor web. Un servidor web es un software que implementa el protocolo HTTP en su parte servidor, y el protocolo HTTP es el protocolo estandarizado que se usa en una comunicación entre el navegador web y el software de la parte servidor (servidor web). Como entre los objetivos de este proyecto no está el de desarrollar un servidor web, se va a usar uno ya implementado en JavaScript y empaquetado para su ejecución en Node js: se llama Express.

Express es un framework que permite crear un servidor web de forma muy sencilla, usándolo como paquete de Node Js. Cuando se usa Express, solo hay que especificar cuál es la carpeta y el archivo index.html al que se debe acceder por defecto, la ruta de la carpeta donde está, y qué es lo que debe hacer cuando recibe las peticiones HTTP o HTTPS (HTTP + SSL). En el servidor web de este proyecto, se usa el protocolo HTTP junto con el protocolo de seguridad SSL, por eso también se especifica la ruta donde está el certificado de clave pública y la clave privada. Al no disponer de un

hosting de Internet, se ha usado como hosting un ordenador personal con un certificado de clave pública y una clave privadas autofirmadas con OpenSSL. En una implementación desplegada en un hosting de Internet, solo habría que poner el certificado y la clave privada dadas por el hosting en su lugar. Se tiene que usar HTTPS porque así lo exigen los navegadores web de la parte cliente, para proteger a sus usuarios de que sean espiados por servidores falsos que reciban audio grabado y enviado por sus navegadores web.

Se precisa que este servidor web sea también un servidor de websockets. Los websockets son un protocolo de comunicaciones entre un navegador web y un servidor web, aunque también se pueden usar en aplicaciones que no sean web. Se utilizan sobre el protocolo HTTP y permite que haya una conexión bidireccional y full-duplex entre el navegador y el servidor web, con una comunicación mediante mensajes. Este tipo de comunicación es el que se utiliza para comunicar la parte cliente y servidor del videojuego. El protocolo HTTP, por sí solo, lo que hace es transferir archivos de hipertexto entre el navegador y el servidor web. La conexión dura lo justo para que se transfiera el archivo y luego se desconecta. Pero en un videojuego, se requiere que la conexión con el servidor sea continua, y que sea el usuario el que decida cuándo desconectarse. Por eso se ha estandarizado un protocolo que se llama websocket que permite hacer esto.

Para que este servidor web sea también un servidor de websockets, se debe implementar en JavaScript el protocolo de websockets para el servidor web anterior. Como entre los objetivos de este proyecto no está el de implementar un servidor de websockets, se va a usar uno ya implementado en JavaScript y empaquetado para su uso en Node Js: se llama Socket.io. Socket.io hace que se puedan usar muy fácilmente los websockets. Cuando el servidor requiere enviar un mensaje de websocket al cliente, solo debe invocar a la función “emit” del objeto socket creado con el módulo socket.io. En el primer argumento se envía el nombre del mensaje y como segundo argumento se envían los datos que van en el websocket. Para esperar a la recepción de un mensaje de websocket de parte del cliente, se hace uso de la función asíncrona “on”, que recibe como primer argumento el nombre del mensaje y como segundo argumento la función que lo maneja.

En la clase Server descrita en el diagrama de clases, está implementado el acceso y uso del servidor web implementado en Express y Socket.io.

3.1.2.1.2 Implementación en JavaScript del artefacto Servidor

JavaScript es un lenguaje de programación débilmente tipado, orientado a objetos, basado en prototipos (algo similar a las clases) y con un modelo de ejecución basado en eventos cuyos manejadores se activan de forma asíncrona.

Es débilmente tipado porque las variables que se usan en el lenguaje solo pueden tener los tipos primitivos “boolean”, “number” y “string” o los tipos más complejos “function” (las funciones se pueden manejar como variables) y los objetos con notación JSON. Estos tipos no se declaran y el tipo de una variable puede cambiar de forma dinámica.

Es orientado a objetos y éstos se escriben en una notación denominada JSON, que consiste en un array asociativo basado en pares clave-valor. Cada clave es el nombre de una variable, que representa a un atributo o a un método del objeto. Los atributos son las variables cuyo valor es un tipo primitivo u otro objeto, y los métodos son las variables cuyo valor es una función.

JavaScript está basado en prototipos. Un prototipo es algo similar a una clase en otros lenguajes, pero no es lo mismo. Cuando se quiere crear una clase, lo que se hace es crear una función. Cada función está definida por un prototipo, que es un objeto cuyos atributos y métodos se crean usando la palabra reservada “this” o accediendo a la propiedad “prototype” de la función. Para crear un objeto basado en una clase/prototipo, se hace uso de la palabra reservada “new” seguida del nombre de la función.

Un ejemplo de una clase/prototipo en JavaScript es la clase Ball del proyecto que se ve en la siguiente imagen:

```
function Ball(game)
{
    this._game = game;

    var uniqid = require('uniqid');
    this._id = uniqid();
    this._posX = null;
    this._posY = null;
    this._radius = null;
    this._color = require("./Utilities.js").randomColor();
}

Ball.prototype.getGame = function()
{
    return this._game;
};

Ball.prototype.getId = function()
{
    return this._id;
};
```

Figura 3.12 Clase/Prototipo Ball en JavaScript

Como se puede observar, la clase Ball se crea usando una función con ese nombre. Y para crear o modificar atributos o métodos de esa clase, se accede al prototipo de Ball mediante la palabra reservada “this” o “Ball.prototype”. Para crear un objeto de la clase Ball, se hace uso de “new Ball”.

El modelo de ejecución de JavaScript es orientado a eventos con manejadores de eventos que se activan de forma asíncrona. Existe un único hilo de ejecución, y una cola de manejadores de eventos. El modelo de ejecución de JavaScript saca un manejador de eventos de la cola, y el único hilo de ejecución lo ejecuta. Cuando termina su ejecución, saca otro manejador de eventos de la cola, y así sucesivamente. El orden en que los manejadores de eventos se sitúan en la cola no es fijo, sino que es el modelo de ejecución de JavaScript quien decide qué manejador va primero, y cual después. Para que un manejador de eventos entre en la cola, tiene que haberse lanzado el evento que lo activa. Dentro de un manejador de eventos se pueden lanzar eventos o crear nuevos manejadores de eventos.

En el siguiente gráfico se puede visualizar el modelo de ejecución de JavaScript explicado anteriormente:

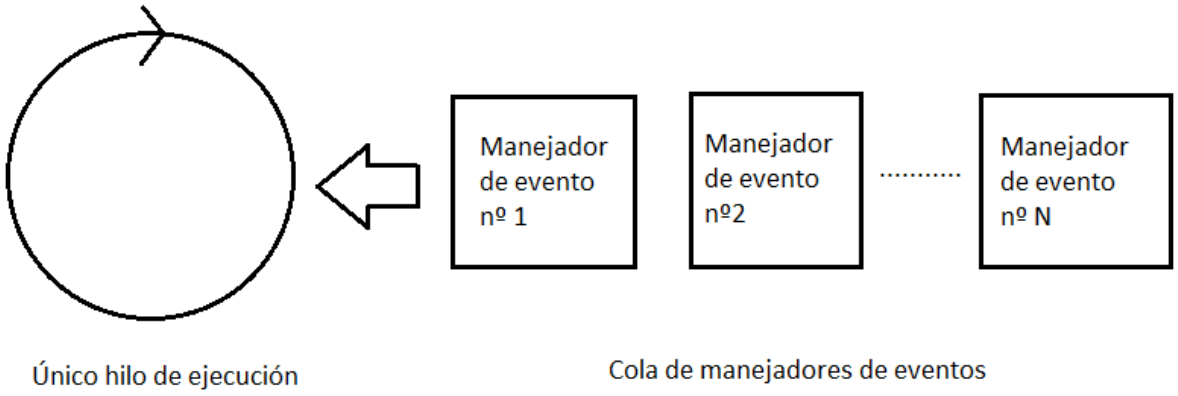


Figura 3.13 Modelo de ejecución de JavaScript

A continuación se va a explicar cómo se aplica este modelo de ejecución al artefacto Servidor.
En el siguiente gráfico se muestra cómo sería la cola de manejadores de eventos del juego en un momento dado:

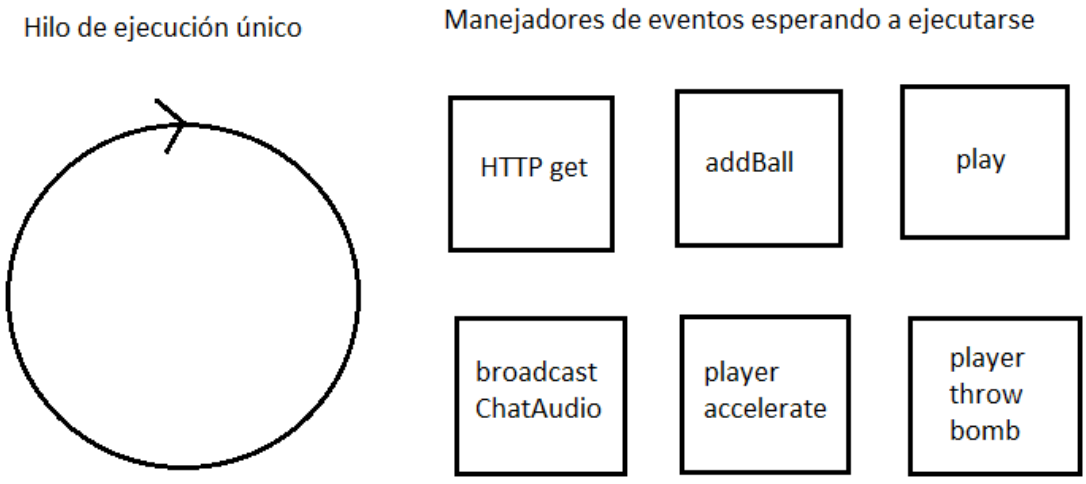


Figura 3.14 Modelo de ejecución de JavaScript aplicado

Los manejadores de eventos que se ven a la derecha se activan por eventos de entrada/salida (la recepción de un mensaje de websocket del cliente, un evento temporal o un evento procedente del gestor de base de datos).

El hilo de ejecución solo puede ejecutar uno de los manejadores de eventos de la derecha a la vez. Es decir, mientras por ejemplo se está enviando el audio que ha grabado un jugador a los demás jugadores (manejador "broadcastChatAudio"), no puede a la vez ejecutar la orden de un jugador de que lance una bomba (manejador "player throw bomb").

Lo que implica esto es muy importante, ya que si un manejador de eventos tiene un tiempo de ejecución grande, bloquearía el sistema entero. Por eso, cada manejador de eventos debería tener un tiempo de ejecución pequeño. De esta forma, al ser pequeños los tiempos de ejecución de cada manejador de eventos, el hilo de ejecución iría saltando rápidamente de un manejador de eventos a otro. Y eso "simularía" una especie de concurrencia como si cada manejador de eventos fuera un hilo de ejecución en un sistema multi hilo.

El problema existe cuando un manejador de eventos tiene que tener un tiempo de ejecución grande, como por ejemplo uno que tenga que hacer accesos a la base de datos, o como por ejemplo el manejador de eventos "play" (hacer jugada) de nuestro sistema. Cuando el juego realiza una jugada, debe por ejemplo mover jugadores, mirar las bolas que se puede comer y eliminar las que se ha comido, comprobar que jugadores atacan a otros.....todo eso tiene mucho tiempo de ejecución, que se incrementa conforme más bolas y jugadores hay en el juego.

Cuando esto ocurre, la solución es dejar que la tarea con mucho tiempo de ejecución se ejecute fuera de Node Js, en otro proceso del sistema operativo diferente, y esperar de forma asíncrona a su terminación, no bloqueando así al resto de manejadores de eventos. Por ejemplo, cuando se accede a la base de datos (algo que suele ocupar mucho tiempo de ejecución) se accede de forma asíncrona, enviando la orden al gestor de base de datos y esperando de forma asíncrona a que termine. La ejecución del manejador de eventos "play" se ha dejado que se haga entera dentro de Node Js. La razón es que se asume que su tiempo de ejecución va a ser pequeño, porque si es grande, el espacio de tiempo entre jugadas será muy grande y el juego se verá muy ralentizado, y no tendrá sentido jugar a un juego en esas condiciones. Para evitar que entren muchos jugadores a la partida, ralentizando el manejador "play", existe un límite establecido de jugadores jugando al juego, y no se deja entrar a los jugadores si la sala está llena.

Existe una amplia discusión sobre si es más conveniente un modelo de ejecución para la parte servidor como el de Node Js (asíncrono y orientado a eventos), o si en cambio, es mejor un modelo multi hilo. Cada uno tiene sus ventajas y sus desventajas.

La desventaja fundamental que tiene Node Js respecto al modelo multihilo, es la mencionada anteriormente de tareas con tiempos de ejecución muy grandes, ya que se tendría que ejecutar fuera de Node JS, en diferentes procesos que ocupan mucha más memoria y más tiempo en el cambio de contexto entre procesos. En cambio, los hilos son más ligeros en memoria y en tiempo de cambio de contexto.

Y la principal ventaja que tiene Node Js respecto a un sistema multihilo, es que en un sistema multihilo hay que implementar la sincronización entre hilos de ejecución, que aumenta la complejidad de la programación y por lo tanto un aumento del número de errores que pueden surgir. Si un hilo de ejecución requiere que ejecute un trozo de código de forma síncrona sin que cambie a otro hilo de forma concurrente, deben implementarse mecanismos de exclusión mutua como los mutex o semáforos. Y esto es algo que no hay que hacer en Node Js porque existe un único hilo. Además, los hilos ocupan tiempo de procesador y memoria aunque en ese momento solo esté en espera de recibir una orden, algo que no ocurre en Node Js.

3.1.2.2 El gestor de bases de datos MongoDB y la base de datos

Otra parte fundamental de la parte servidor del videojuego es la base de datos y el software que lo gestiona, llamado gestor de bases de datos. Primero, hay que decidir qué información sobre el juego se va a guardar de forma permanente en la base de datos. Se ha decidido que sean los datos de los jugadores y sus resultados en cada una de las partidas. Esto se puede ver en el siguiente diagrama de tablas de la base de datos:

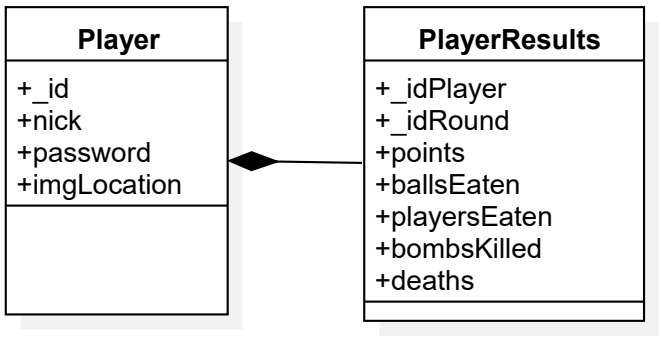


Figura 3.15 Diagrama de tablas de la base de datos

Como se puede observar, existen dos tablas. La tabla Player representa los datos de un jugador (tanto jugador humano como bot) que se van a guardar en la base de datos. Así que tiene como columnas: un "id" de identificación, su nick, su password y la dirección de localización de su avatar "imgLocation". La clave primaria es el "id" de identificación. Y la tabla PlayerResults representa los resultados que tienen los jugadores de la tabla Player en cada una de las partidas. Tiene como columnas: "_idPlayer" que es una clave foránea que apunta a la columna "id" de la tabla Player, "_idRound" que es el identificador que representa a una partida, y después una serie de columnas que representan los resultados que ha tenido el jugador con el identificador "_idPlayer" en la partida "_idRound": "points" que son los puntos obtenidos, "ballsEaten" que es el nº de bolas comidas, "playersEaten" que es el nº de jugadores comidos, "bombsKilled" que es el nº de jugadores que ha matado con una bomba y "deaths" que es el nº de veces que ha muerto en la partida. Las claves primarias de esta tabla son conjuntamente el identificador del jugador "idPlayer" y el identificador de la partida "idRound".

Se puede ver que esta base de datos está normalizada por lo menos hasta la tercera forma normal:

- Cumple la primera forma normal, porque todas las columnas contienen valores atómicos.
- Cumple la segunda forma normal, porque está en primera forma normal y porque en cada tabla, los atributos que no forman parte de la clave primaria dependen funcionalmente totalmente de la clave primaria. En la tabla Player se cumple porque todos los datos del jugador dependen de su identificador. Y en la tabla PlayerResults se cumple porque ningún resultado de la partida depende únicamente o del jugador (idPlayer) o de la partida (idRound), sino de los dos atributos a la vez.
- Cumple la tercera forma normal, porque está en segunda forma normal y porque en cada

tabla, cada atributo que no forma parte de la clave primaria, no depende funcionalmente de otro atributo que no forme parte de la clave primaria. En la tabla Player, el nick, el password y la localización del avatar no dependen entre sí. Y en la tabla PlayerResults, los resultados de la partida no dependen entre sí.

Las bases de datos se suelen normalizar de esta forma para evitar que haya datos repetidos. Y esto es así porque los datos repetidos en una base de datos generan dos problemas:

- Hacen que la base de datos ocupe más y gaste más recursos hardware.
- Hay problemas a la hora de hacer escritura de datos, porque los datos repetidos se deben escribir todos antes de que ningún otro proceso pueda acceder a ellos. Esto provoca lentitud en las escrituras, y también posibles errores de inconsistencia si no se escriben todos los datos repetidos o alguien accede a ellos sin que se termine la escritura.

Ante esto, en este proyecto se plantea la siguiente pregunta: ¿tiene alguna ventaja tener una base de datos NO normalizada? La respuesta es que sí. Puede ser conveniente tener una base de datos no normalizada en el caso de que se tengan bases de datos muy grandes donde haya que leer muchos datos y se escriban pocos datos en relación a la cantidad grande de datos leídos. Porque una base de datos normalizada tiene muchas ventajas, pero la gran desventaja que tiene es la unión de tablas, sobre todo cuando éstas son grandes, porque es una operación con mucho tiempo de ejecución. Por eso, la solución que se propone en este proyecto es usar un gestor de base de datos no normalizadas (también llamado No-SQL porque no usan el lenguaje SQL de las bases de datos relacionales). Se llama MongoDB.

La base de datos del proyecto en MongoDB

En la base de datos relacional y normalizada anterior, existían dos tablas. Si se tienen muchos datos de resultados de partidas, para leer los resultados de un jugador en concreto por ejemplo, habría que hacer una unión de las dos tablas que sería muy costosa en tiempo de ejecución. ¿Entonces qué ocurriría si metemos una de las tablas dentro de otra? La tabla PlayerResults podría estar dentro de la tabla Player, y así no habría que hacer uniones de tablas costosas en tiempo de ejecución.

En MongoDB se puede hacer esto. Una “columna” de una tabla, puede ser otra tabla con más columnas. Para describir las bases de datos en este gestor, se usa el lenguaje de datos de JavaScript descrito anteriormente en la memoria: JSON. Un fragmento de la base de datos en un momento dado es el siguiente:


```
{ "_id" : ObjectId("59d7e9f0bfaee228b9d73a98"), "nick" : "Greece", "imgLocation" : "./img/Greece.png", "rounds" : [ { "r
oundId" : "gtv3l2w8j8gcv3d6", "points" : 10, "ballsEaten" : 28, "playersEaten" : 1, "bombsKilled" : 1, "deaths" : 8 }, {
 "roundId" : "gtv3l2w8j8gd1se6", "points" : 220, "ballsEaten" : 40, "playersEaten" : 0, "bombsKilled" : 1, "deaths" : 8
} ] }
{ "_id" : ObjectId("59d7e9f0bfaee228b9d73a99"), "nick" : "Mexico", "imgLocation" : "./img/Mexico.png", "rounds" : [ { "r
oundId" : "gtv3l2w8j8gcv3d6", "points" : 150, "ballsEaten" : 32, "playersEaten" : 0, "bombsKilled" : 3, "deaths" : 11 },
 { "roundId" : "gtv3l2w8j8gd1se6", "points" : 0, "ballsEaten" : 27, "playersEaten" : 1, "bombsKilled" : 3, "deaths" : 14
 } ] }
{ "_id" : ObjectId("59d7e9f0bfaee228b9d73a9a"), "nick" : "Romania", "imgLocation" : "./img/Romania.png", "rounds" : [ {
"roundId" : "gtv3l2w8j8gcv3d6", "points" : 10, "ballsEaten" : 27, "playersEaten" : 0, "bombsKilled" : 2, "deaths" : 13 }
, { "roundId" : "gtv3l2w8j8gd1se6", "points" : 10, "ballsEaten" : 35, "playersEaten" : 1, "bombsKilled" : 2, "deaths" :
10 } ] }
```


Figura 3.16 Fragmento de la base de datos MongoDB

En MongoDB, las tablas se llaman “colecciones”. Y el equivalente a las filas de una base de datos relacional, es lo que se llama “documento”. En la imagen anterior se puede ver la colección/tabla “players” (jugadores) con tres documentos/filas. Cada fila son los datos correspondientes a un jugador, incluido un campo/columna llamado “rounds” que representa los datos de las partidas de ese jugador. Los datos de las partidas a su vez son un conjunto de varios documentos/filas donde cada uno contiene los datos de una partida del jugador. Al estar los datos de las partidas como columna/campo dentro de un jugador, si se requiere saber por ejemplo cuántos puntos ha obtenido un jugador sumando todas sus partidas, no hay que hacer una unión de tablas buscando el identificador del jugador en la tabla de resultados, sino que directamente se accede a su atributo “rounds”.

En el videojuego, cada vez que termina una partida, se requiere saber por ejemplo la suma de puntos de todas las partidas en las que ha participado el jugador. Cada partida se juega cada poco tiempo, y al cabo de días de que esté el juego en funcionamiento en el servidor, podría haber miles de partidas. Con este diseño de la base de datos se accedería mucho más rápidamente a los datos de miles de partidas de un jugador.

El problema existe, como se ha dicho antes, con las escrituras de datos. MongoDB no da soporte a transacciones y las operaciones son atómicas solo a nivel de documento/fila. Esto quiere decir que cuando una partida termina y se escriben los datos de esa partida para cada jugador, al estar en diferentes documentos, la operación de escritura de los resultados no es atómica, y por lo tanto podrían ocurrir errores de inconsistencia si un hilo de ejecución intenta leer los resultados de las partidas antes de que otro hilo termine de escribirlos. Este tipo de problemas deben solucionarse sincronizando dichos hilos de ejecución para que no se ejecuten de forma concurrente, sino que uno se ejecute después del otro.

En nuestro caso, como se usa la tecnología Node Js que tiene un único hilo de ejecución, no existe este problema, porque la escritura de resultados está en un manejador de eventos, y su lectura en otro, y en Node Js nunca se ejecutan de forma concurrente, sino asíncrona. El problema podría existir si por ejemplo en un futuro se decide por ejemplo hacer salas de partidas, donde habría varios procesos Node Js leyendo la base de datos de forma concurrente. Para evitar el problema de las escrituras en este caso, se podría hacer sincronizando entre sí los procesos de Node Js para que no se lean y se escriban resultados de forma concurrente. Para hacerlo, se podría usar un mecanismo de paso de mensajes como los websockets o algún otro.

Por último, decir que en MongoDB no se usa el lenguaje SQL para hacer las consultas, sino un lenguaje propio. Por ejemplo, para consultar si un jugador con cierto nick y password existe, se usa el siguiente código:

```
DatabaseServer.prototype.findPlayer = function(db, nick, password, callback){
   var collection = db.collection('players');
   collection.findOne({"nick": nick, "password": password},function(err, playerFound) {
      callback(err,playerFound);
   });
};
```

Como se puede ver, el equivalente a leer las filas de una tabla en SQL (SELECT * FROM) es el método “find”, o en nuestro caso, que solo queremos un resultado, el método “findOne”. Y como parámetro de ese método, se introducen los valores que se buscan en cada documento/fila (esto es el

equivalente en SQL a la cláusula WHERE).

Y un ejemplo de escritura en la base de datos es la actualización de datos siguiente:

```
DatabaseServer.prototype.registerPlayer = function(db, nick, password, imgLocation, callback){
   var collection = db.collection("players");
   collection.updateOne({nick: nick, password: password}, {$set: {imgLocation: imgLocation},
$setOnInsert: {nick: nick, password: password}}, {upsert: true}, function(err, playerFound){
      callback(err,playerFound);
   });
};
```

El método “update” en MongoDB es el equivalente al del mismo nombre en SQL. Como parámetro de ese método, se introduce los datos que se buscan para actualizar, es decir, el equivalente a la cláusula WHERE de SQL, y como un objeto JSON de nombre “$set” se introducen los valores que se quieren cambiar (el equivalente a SET columna = valor en SQL).

3.2 DISEÑO E IMPLEMENTACIÓN DE LA PARTE CLIENTE

El diseño de la parte cliente del videojuego se ha realizado usando el patrón de diseño modelo-vista-controlador. La vista representa la interfaz gráfica de usuario, el controlador la dinámica de la parte cliente del videojuego que controla la interfaz gráfica, y el modelo los datos que se utilizan.

A continuación en los dos siguientes subapartados se exponen el diseño de la parte cliente del videojuego, la implementación de ese diseño en un cliente PC y la implementación de ese diseño en un cliente móvil (smartphone o tablet).

3.2.1 Diseño de la parte cliente

3.2.1.1 Diagramas de clases

En el siguiente diagrama de clases se puede ver la estructura de la parte cliente del videojuego en su nivel de abstracción más alto:

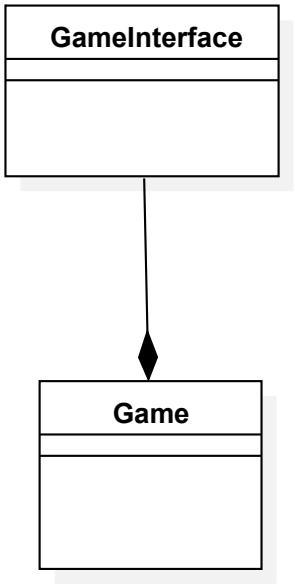


Figura 3.17 Diagrama de clases de la parte cliente del videojuego

La clase Game representa el modelo en el patrón de diseño modelo-vista-controlador. Y la clase GameInterface representa la interfaz de la parte cliente, en la que está tanto la vista como el controlador del patrón de diseño modelo-vista-controlador. A continuación se entra en un nivel más bajo de abstracción para cada una de estas clases del diagrama, exponiendo primero el diagrama de clases de Game y el diagrama de clases de GameInterface.

3.2.1.1.1 Diagrama de clases del modelo (Game)

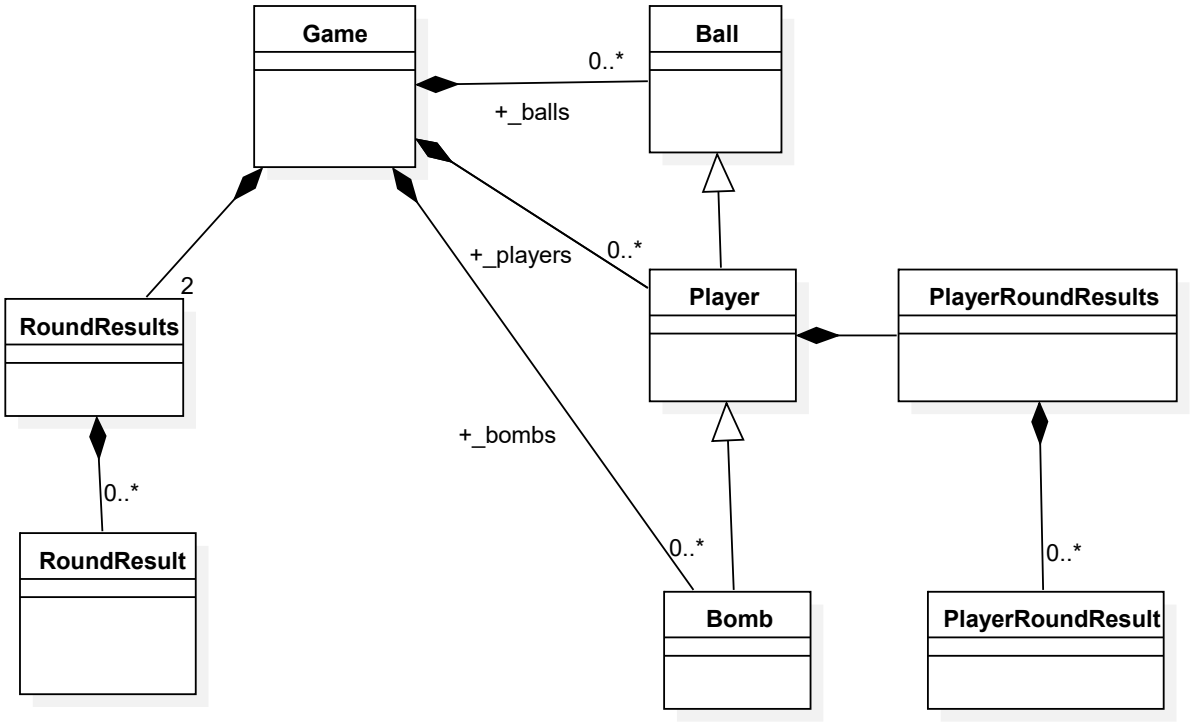


Figura 3.18 Diagrama de clases del modelo

Las clases representadas aquí tienen el mismo significado que en el diagrama de clases de la parte servidor del videojuego. Esto es porque se trata de los mismos datos enviados por el servidor al cliente. Cada jugador, en su máquina cliente va a tener las bolas (Ball), jugadores y bots (Player) y bombas (Bomb) que hay en el tablero de juego en cada momento.

Podría pensarse que el hecho de que cada jugador en su PC o smartphone contenga en cada momento todos los elementos del tablero podría hacer mucho uso de memoria. Pero el juego se ha hecho de tal manera que nunca haya muchos jugadores y bombas en un mismo momento, porque además un número alto de jugadores haría que se ralentizara el juego. Además, hay que tener en

cuenta que si solo se guardase en memoria los datos de los elementos del juego que están en el área visible del jugador, en cada momento el servidor tendría que buscar qué elementos están cerca del jugador, lo cual gasta mucho tiempo de ejecución, y enviarlos a cada momento por la red, lo cual gasta recursos de red.

Las clases del anterior diagrama no tienen los mismos métodos que las clases del diagrama de clases de la parte servidor del videojuego. Prácticamente solo contienen métodos para obtener y cambiar sus atributos, pues al ser el modelo, carece de dinámica. Solo la clase Player tiene los métodos “accelerate” y “resurrect” que se corresponden con la gestión que se hace de los casos de uso “Cambiar velocidad jugador” y “Resucitar jugador” respectivamente en la parte cliente del videojuego.

3.2.1.1.2 Diagrama de clases de la vista y el controlador (GameInterface)

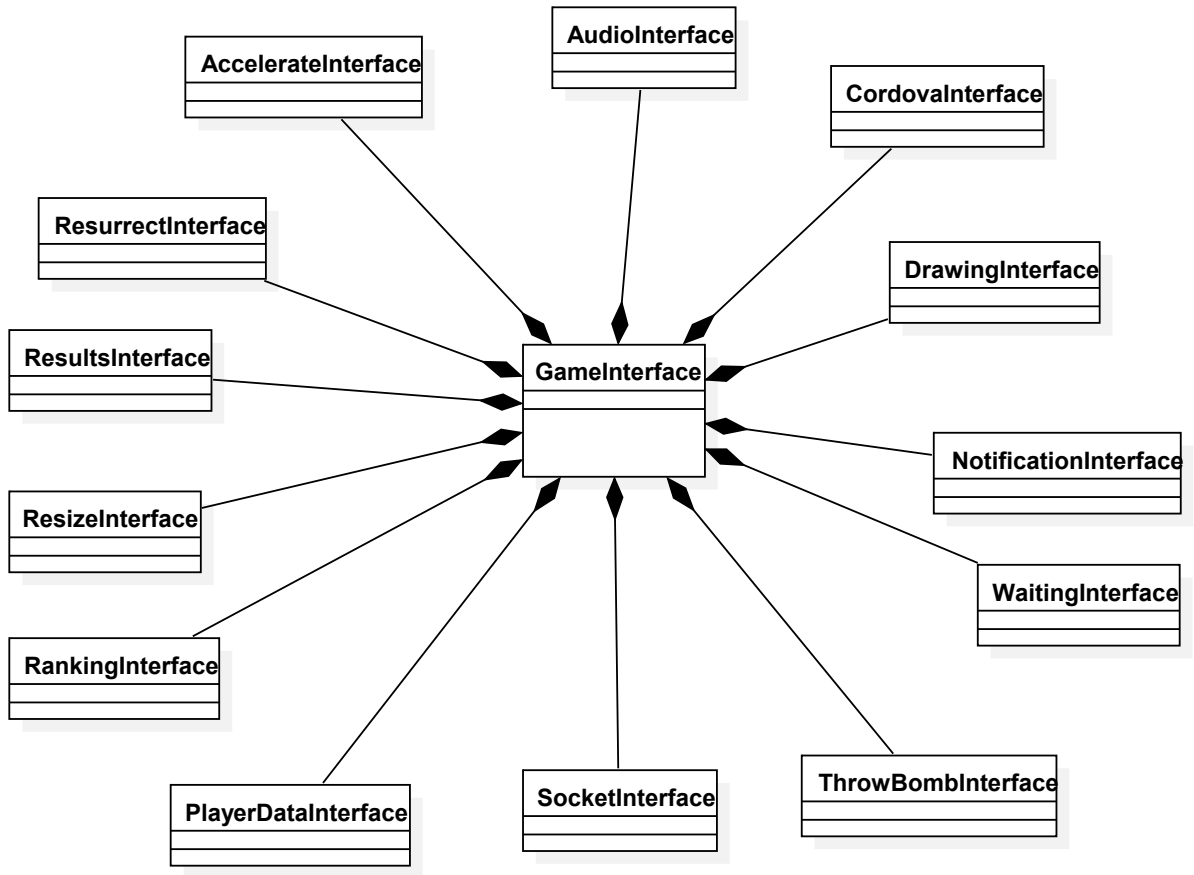


Figura 3.19 Diagrama de clases de la vista y el controlador

Este diagrama se corresponde con las clases que definen la estructura de la interfaz de la parte cliente del videojuego. La interfaz de la parte cliente del videojuego se divide en más interfaces, cada una con una función distinta. Cada una de estas interfaces contiene el controlador y la vista del patrón de diseño.

A continuación se procede a explicar cada una de estas interfaces:

- AccelerateInterface: es la interfaz que trata las órdenes del usuario o jugador correspondientes al caso de uso "Cambiar velocidad jugador".
- AudioInterface: es la interfaz que trata las órdenes del usuario o jugador correspondientes al caso de uso "Chatear por voz". Es la interfaz que se encarga de grabar el audio y enviarlo al servidor.
- CordovaInterface: es la interfaz que trata las órdenes dadas por el usuario o jugador desde el dispositivo móvil usando la tecnología Phonegap o también llamada Cordova. Esta tecnología se explicará más adelante.
- DrawingInterface: es la interfaz que conecta con la interfaz gráfica de los gráficos del videojuego, mostrando los elementos del tablero. En esta interfaz se implementa parte de la gestión del caso de uso "Mostrar partida". De ahí que contenga los métodos: "showBalls" (caso de uso "Mostrar bolas"), "showPlayers" (caso de uso "Mostrar jugadores"), "showPlayer" (caso de uso "Mostrar jugador") y "showBomb" (caso de uso "Mostrar bomba").
- NotificationInterface: es la interfaz que se encarga de mostrar al usuario notificaciones o mensajes importantes, como por ejemplo cuando ha realizado un logueo incorrecto o ha muerto en el juego.
- PlayerDataInterface: es la interfaz que se encarga de tratar las órdenes del usuario correspondientes a los caso de uso "Registro de jugador" y "Login de jugador" en los métodos "initRegisterInterface" y "initLoginInterface" respectivamente.
- RankingInterface: es la interfaz que se encarga de mostrar al jugador la lista de jugadores ordenada de mayor a menor por puntos. Se corresponde con la gestión del caso de uso "Mostrar lista de jugadores".
- ResizeInterface: es la interfaz que se encarga de redimensionar todas las interfaces gráficas cuando el usuario da la orden de hacerlo. En especial la redimensión de los elementos del

juego, correspondiente al caso de uso “Cambiar tamaño área”.

- ResultsInterface: es la interfaz que muestra los resultados del jugador, de la partida y los resultados globales. Para ello contiene los métodos “displayPlayerResults” (caso de uso “Mostrar resultados jugador”), “displayRoundResults” (caso de uso “Mostrar resultados partida”) y “displayGlobalResults” (caso de uso “Mostrar resultados globales”).
- ResurrectInterface: es la interfaz responsable de mostrar al jugador cuándo ha muerto en el juego y de mostrar la interfaz gráfica y enviar al servidor la orden de resucitar a su jugador cuando el jugador da esa orden. Se corresponde con el caso de uso “Resucitar jugador”.
- SocketInterface: es la interfaz que implementa la comunicación por sockets del servidor hacia el cliente y del cliente hacia el servidor. En el caso del proyecto esta comunicación es mediante websockets usando la tecnología Socket.io, pero la implementación podría ser con cualquier otra tecnología que implementase comunicación de mensajes por sockets.
- ThrowBombInterface: es la interfaz que se encarga de tratar la orden del usuario o jugador de que quiere lanzar una bomba en el juego. Se corresponde con el caso de uso “Lanzar bomba”.
- WaitingInterface: es la interfaz que se encarga de mostrar al jugador los mensajes de espera debidos a que debe esperar a recibir algo del servidor. Por ejemplo, cuando tiene que esperar a saber si se ha logueado correctamente, o cuando tiene que esperar a que se muestren los resultados de la partida.

3.2.1.2 Diagramas de actividad

En este subcapítulo se va a exponer la parte del diseño en que se explica la dinámica o funcionamiento de la parte cliente del videojuego mediante diagramas de actividad en UML. Como se vio antes, se ha seguido un patrón de diseño modelo-vista-controlador. Así que para explicar la dinámica de funcionamiento de la parte cliente del videojuego, hay que explicar cómo funciona la parte de controlador, que es la que posee la funcionalidad.

La explicación de esta dinámica se ha dividido en dos partes: acciones desde el cliente al servidor, y acciones desde el servidor al cliente. Las acciones del cliente al servidor representan cómo es la ejecución de las órdenes que da el usuario/jugador a través del controlador de la interfaz y que van hacia el servidor. Y las acciones del servidor al cliente representan cómo es la ejecución de las

órdenes que van desde el servidor al cliente y que maneja el controlador.

3.2.1.2.1 Acciones desde el cliente al servidor

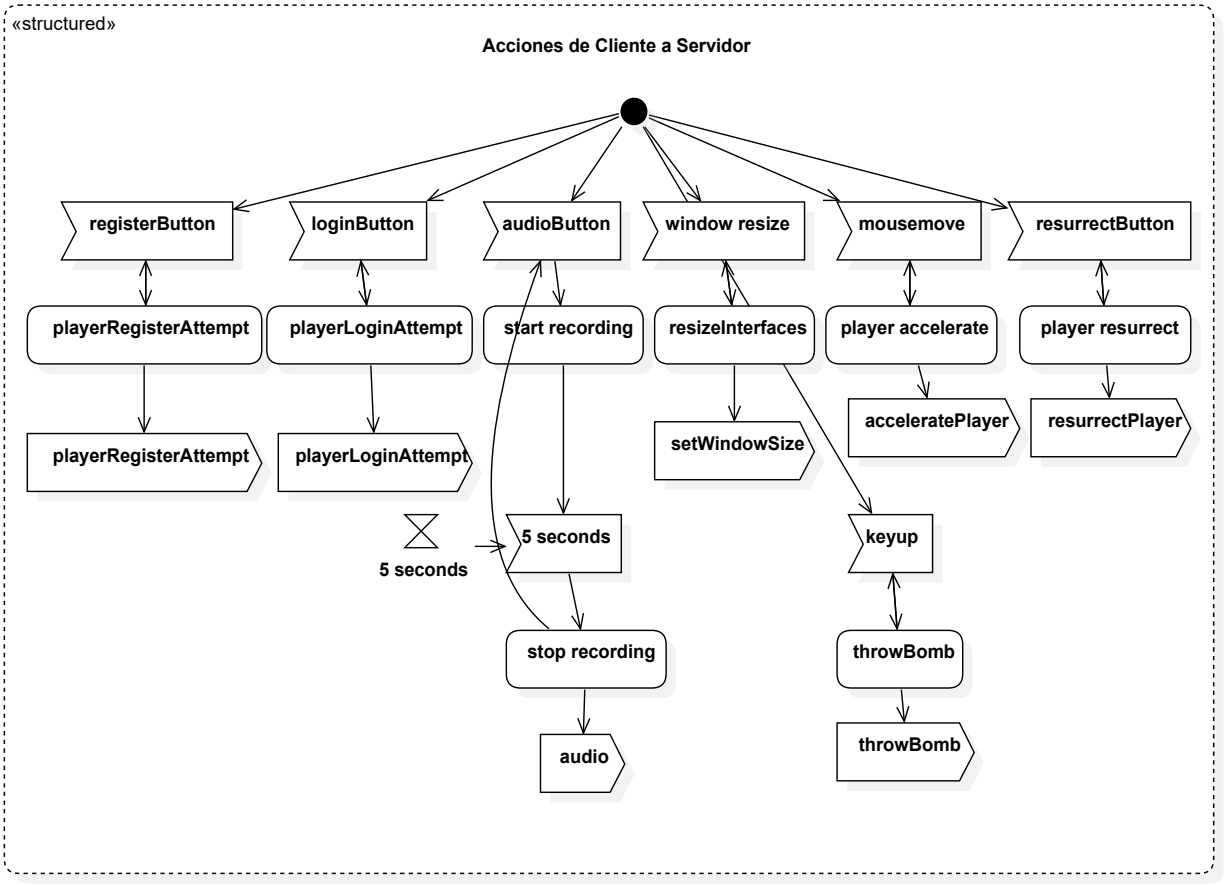


Figura 3.20 Diagrama de actividad. Acciones de Cliente a Servidor.

Cuando se inicia el controlador, se crean varias ramas de ejecución que se pueden ejecutar a la vez. Cada rama de ejecución corresponde con la parte controlador de una de las interfaces vistas anteriormente en el diagrama de clases del diseño.

A continuación se explica cada acción de una rama de ejecución, se dice a qué interfaz corresponde y qué caso de uso implementa.

- PlayerRegisterAttempt. La ejecución de esta acción la hace el controlador de la interfaz "PlayerDataInterface". Cuando se hace click en el botón de registro de esta interfaz (registerButton), se lanza un mensaje que activa la ejecución de esta rama y que implementa

el caso de uso “Registro de jugador” en la parte cliente del videojuego. Al hacerlo, envía un mensaje de nombre “playerRegisterAttempt” al servidor, indicándole que quiere registrar un jugador.

- PlayerLoginAttempt. La ejecución de esta acción la hace el controlador de la interfaz “PlayerDataInterface”. Cuando se hace click en el botón de login de esta interfaz (loginButton), se lanza un mensaje que activa la ejecución de esta rama y que implementa el caso de uso “Login de jugador” en la parte cliente del videojuego. Al hacerlo, envía un mensaje de nombre “playerLoginAttempt” al servidor, indicándole que quiere loguearse en el juego.
- Start recording. La ejecución de esta acción la hace el controlador de la interfaz “AudioInterface”. Cuando se pulsa el botón de grabar audio de esta interfaz para chatear por voz con los demás jugadores (audioButton) se lanza un mensaje que activa la ejecución de esta rama y que implementa el caso de uso “Chatear por voz” en la parte cliente del videojuego. Al hacerlo, va grabando el audio del micrófono. Cuando se lanza un mensaje temporal de que han pasado 5 segundos, se ejecuta la acción “stopRecording” que para la grabación y envía lo grabado mediante un mensaje llamado “audio” al servidor.
- ResizeInterfaces. La ejecución de esta acción la hace el controlador de la interfaz “ResizeInterface”. Cuando se redimensiona la ventana del navegador (window resize) se lanza un mensaje que activa la ejecución de esta rama y que implementa el caso de uso “Cambiar tamaño área”. Al hacerlo, se redimensionan las interfaces gráficas y se envía el mensaje “setWindowSize” al servidor en el que se da la orden de que se cambie el área de juego mostrada al jugador.
- Player accelerate. La ejecución de esta acción la hace el controlador de la interfaz “AccelerateInterface”. Cuando el usuario mueve el ratón se lanza el mensaje “mousemove”, el cual activa la ejecución de esta rama y que implementa el caso de uso “Cambiar velocidad jugador” en su parte cliente del videojuego enviando el mensaje “acceleratePlayer” al servidor para que el jugador cambie su velocidad.
- ThrowBomb. La ejecución de esta acción la hace el controlador de la interfaz “ThrowBombInterface”. Cuando el usuario pulsa y suelta la barra espaciadora del teclado, se lanza el mensaje “keyup” el cual activa la ejecución de esta rama y que implementa el caso de uso “Lanzar bomba”, el cual envía el mensaje “throwBomb” para decirle al servidor que lance una bomba en nombre del jugador.

- Player resurrect. La ejecución de esta acción la hace el controlador de la interfaz "ResurrectInterface". Cuando el usuario pulsa el botón de la interfaz de resucitar a su jugador ("resurrectButton"), envía un mensaje que activa la ejecución de esta rama y que implementa el caso de uso "Resucitar jugador" en la parte cliente. El cual envía el mensaje "resurrectPlayer" al servidor para indicarle que quiere resucitar a su jugador.

Además, cuando el usuario se desconecta del juego cerrando la ventana de juego debe enviarse un mensaje al servidor llamado "disconnect" que hace que el jugador se salga de la partida , implementando así el caso de uso "Salir partida".

Cada vez que se termina de ejecutar cada una de estas acciones, vuelve a esperar a recibir los mensajes que vuelven a activar las acciones cuando se vuelven a recibir.

3.2.1.2.2 Acciones desde el servidor al cliente

En este apartado se explica cómo es el funcionamiento del controlador en las acciones que van desde el servidor hacia el cliente, explicando cómo el controlador trata los mensajes provenientes del servidor. Este controlador forma parte de la interfaz SocketInterface.

Entre los mensajes que recibe el controlador desde el servidor están los siguientes:

- PlayerLoggedIn, PlayerNotLoggedIn, PlayerRegistered y PlayerNotRegistered. Representan mensajes del servidor indicando al jugador si ha podido loguearse en el juego o no, o si ha podido registrarse en el juego o no, respectivamente. Cuando se reciben estos mensajes, se activa la ejecución que muestra mensajes de notificación indicando lo que ha ocurrido mediante la interfaz NotificationInterface.
- RoundStarted, RoundFinished y RoundStopped. Representan mensajes que indican cuando una partida ha empezado, finalizado o parado respectivamente. Cuando se para una partida (RoundStopped) se activa la ejecución que muestra la interfaz WaitingInterface que indica al jugador que espere a que se carguen los resultados de la partida. Cuando se terminan de calcular los resultados, el servidor envía el mensaje "RoundFinished" que se recibe en el cliente y activa la ejecución que muestra los resultados de la partida (caso de uso "Mostrar

resultados"). Cuando se recibe el mensaje del servidor de que una nueva partida ha empezado, se oculta la ventana de espera.

- RoundTime y NextRoundTime. Estos mensajes que provienen del servidor indican el tiempo que queda para que termine una partida o para que empiece la siguiente respectivamente. La recepción de estos mensajes activa la ejecución que muestra un cronómetro en la partida indicando este tiempo (caso de uso "Mostrar cronómetro").
- DeadPlayer. Es el mensaje proveniente del servidor que indica que un jugador de la partida ha muerto. Si el que ha muerto es el propio receptor, muestra la interfaz ResurrectInterface, indicándole que ha muerto.
- Board, Balls, NewBall, DeletedBall, NewPlayerImgString, PlayersImgString, Players, NewPlayer y DeletedPlayer. Todos estos mensajes son notificaciones que el servidor envía al cliente anunciando de la adición y eliminación de elementos del juego: adición/eliminación de bolas, jugadores, avatares de jugadores, etc. , y que activan la ejecución de la adición/eliminación de estos elementos.
- Audio. Es el mensaje que contiene el audio de voz enviado por algún jugador del juego desde el servidor. Cuando se recibe, activa la ejecución que reproduce el audio en el dispositivo del jugador.
- PlayersPlayed. Este mensaje proveniente del servidor anuncia que el juego ha ejecutado una jugada, es decir, ha ejecutado el caso de uso "Hacer jugada". Al recibirse en el cliente, activa la ejecución que muestra gráficamente la nueva situación de los elementos del juego (caso de uso "Mostrar partida") y actualiza la lista de jugadores ordenada por puntos (caso de uso "Mostrar lista de jugadores").

3.2.2 Implementación de la parte cliente en el PC

En el siguiente diagrama se puede ver la implementación en un PC de la parte cliente del videojuego diseñada en el apartado anterior y su relación con otros elementos del sistema:

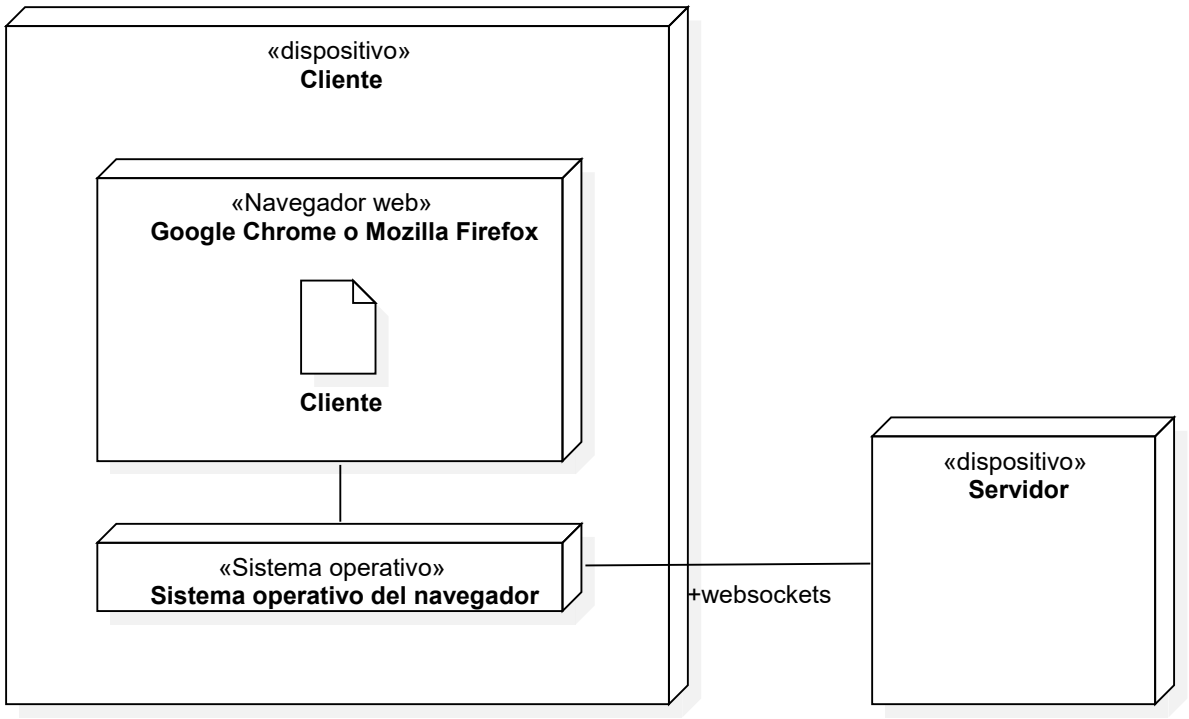


Figura 3.21 Diagrama de despliegue del cliente PC

La parte cliente del videojuego es el artefacto "Cliente" que se puede observar en el diagrama. Como se vio en el apartado de diseño, la parte cliente del videojuego se ha diseñado siguiendo el patrón de diseño modelo-vista-controlador. Tanto el modelo como el controlador se han implementado usando el lenguaje JavaScript, ya explicado en la implementación de la parte servidor del videojuego. Además, en JavaScript, se usa la API de Canvas 2D para poder dibujar gráficos en la interfaz gráfica, la API de audio que permite grabar y reproducir audio y el paquete de websockets que se usó para la parte servidor del videojuego, y el cual permite comunicarse con el servidor.

La vista, es decir la interfaz gráfica, está implementada usando los lenguajes HTML y CSS. Toda esta parte cliente del videojuego se ejecuta en el entorno de ejecución que proporciona un navegador web. El navegador web interpreta los lenguajes HTML, CSS, y JavaScript. En este

proyecto se ha probado a implementar la parte cliente del videojuego sobre los navegadores Google Chrome y Mozilla Firefox. En Internet Explorer, que es otro de los navegadores web más usados, no se puede ejecutar el videojuego porque muchas de las funcionalidades de la API de audio y vídeo aún no están implementadas a pesar de estar ya estandarizadas.

El artefacto que contiene la implementación de la parte cliente del videojuego se trata de una carpeta que contiene de forma separada las implementaciones del modelo, de la vista y del controlador. Las interfaces gráficas o vistas están definidas en el archivo index.html en lenguaje HTML, y su diseño gráfico está definido en los archivos contenidos dentro de la carpeta "css" que están escritos en lenguaje CSS. Cada archivo tiene el nombre de la interfaz de la cual define su diseño gráfico. Por ejemplo, "rankingInterface.css", "playerDataInterface.css", etc. Los controladores están definidos en los archivos contenidos dentro de la carpeta "js/gameInterface". Al igual que en el caso de las vistas, los nombres de los archivos son los nombres de las interfaces del cual implementan su controlador. Por ejemplo: "AccelerateInterface.js", "AudioInterface.js", etc. Y las clases del modelo están definidas en los archivos contenidos dentro de la carpeta "js/gameLogic", con sus mismos nombres: "Ball.js", "Player.js", etc.

A continuación se va a explicar cómo es, por un lado, la implementación del modelo y el controlador y las tecnologías utilizadas. Y por otro lado, la implementación de la vista y las tecnologías utilizadas.

3.2.2.1 Implementación del modelo y el controlador

Tanto el modelo como el controlador de la parte cliente del videojuego se han implementado usando JavaScript. JavaScript y su modelo de ejecución ya se explicaron en el capítulo de diseño e implementación de la parte servidor.

En este caso, el entorno de ejecución para JavaScript no es Node JS, sino los navegadores web. Los navegadores web funcionan como intérpretes del lenguaje JavaScript. En este proyecto se ha implementado para ser usado en los navegadores Google Chrome y Mozilla Firefox. En Internet Explorer no, porque las implementaciones de JavaScript para audio y vídeo no están totalmente implementadas en este navegador, y por ejemplo no están implementadas las funciones relativas a la

grabación de audio, funciones que ya están estandarizadas.

Para explicar la implementación del diseño de los controladores en JavaScript, se muestra cuál podría ser una situación en la que podría encontrarse el modelo de ejecución de JavaScript cuando está ejecutando los controladores:

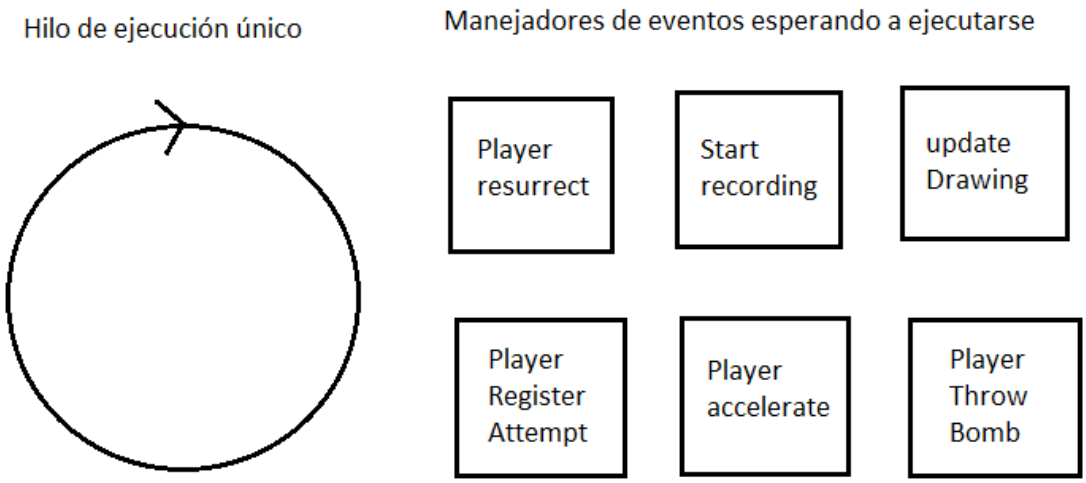


Figura 3.22 Modelo de ejecución de la parte cliente

Como se puede observar, al igual que en el caso de la ejecución de la parte servidor del videojuego, existe un único hilo de ejecución. A la derecha se puede ver un conjunto de acciones de controlador que pueden estar esperando a ejecutarse en la cola de manejadores de eventos en un momento dado. Si una acción está esperando a ejecutarse, es porque ha ocurrido el evento que activa su manejador y ejecuta la acción.

Estos manejadores los ejecuta el único hilo de ejecución de forma asíncrona. Y esto implica que solo se puede ejecutar un manejador a la vez. Existe por tanto el mismo problema que en el servidor: si un manejador tiene mucho tiempo de ejecución, bloquearía al sistema entero. Esto nunca ocurre, porque las acciones que podrían tener mucho tiempo de ejecución, como son el tratamiento del audio y del vídeo, se ejecutan por el navegador y el sistema operativo fuera del modelo de ejecución de JavaScript. Es decir, JavaScript solo da la orden de que se reproduzcan o se capturen audios y que se dibujen gráficos, y espera de forma asíncrona a su ejecución por el navegador, no bloqueando al resto de manejadores.

A continuación se va a explicar cómo los controladores dibujan los gráficos del videojuego mediante la API de Canvas 2D en JavaScript, y cómo captura y reproduce audio mediante la API de audio.

3.2.2.1.1 Dibujo de gráficos con Canvas 2D

A continuación se procede a explicar la tecnología con la que se dibujan los gráficos del videojuego: Canvas 2D. Canvas es una etiqueta nueva en HTML 5, en el que mediante JavaScript se puede acceder a los métodos de una API con la que se pueden generar gráficos que se visualizarán dentro de esa etiqueta. Con Canvas se pueden generar tanto gráficos en dos dimensiones como en tres dimensiones. Para dibujar en un canvas, lo primero que se hace es acceder a su contexto con el método "getContext". Si en este método se introduce el parámetro "2d" se podrá dibujar gráficos solo en dos dimensiones, y si se introduce el parámetro "3d" se podrá dibujar tanto gráficos en dos dimensiones como en tres dimensiones usando la API WebGL, que está basada en la API OpenGL y está estandarizada.

Se ha usado el contexto de dos dimensiones de Canvas. Una vez definido este contexto, se puede proceder a dibujar. El espacio de dibujo se divide en un grid de coordenadas X e Y. Se pueden dibujar dos tipos de elementos primitivos: las líneas (rectas o curvas) y los rectángulos. A partir de las líneas, se pueden dibujar elementos más complejos llamados "caminos" o "paths", que son la unión de una o varias líneas. Para crear un camino, se usa el método "beginPath" y para terminar de crearlo y cerrarlo, se usa el método "closePath". Para dibujar el camino, se usa el método "stroke" y para dibujar su relleno se usa el método "fill". Para cambiar el estilo de línea se usa el atributo strokeStyle, y para cambiar el estilo del relleno, se usa el atributo fillStyle.

Un ejemplo de todo esto se puede ver en el siguiente fragmento de código, donde dentro de la interfaz DrawingInterface, se procede a dibujar un jugador:

```
this._drawing.save();
this._drawing.beginPath();
this._drawing.arc(playerPosXDrawing,playerPosYDrawing,playerRadiusDrawing,0,2*Math.PI);
this._drawing.strokeStyle = "black";
this._drawing.lineWidth = 5;
this._drawing.stroke();
this._drawing.fillStyle = player.getColor();
this._drawing.fill();
this._drawing.closePath();
this._drawing.restore();
```

Figura 3.23 Ejemplo de Canvas 2D

Como se puede observar, primero se procede a guardar el contexto de dibujo usando el método "save", porque durante el dibujo se pueden hacer cambios en él que luego no se desean en otros dibujos. Cuando se termina de dibujar, se usa el método "restore" para obtener el contexto original sin cambios. Se dibuja un círculo dibujando un camino que está compuesto por un tipo de línea curva llamado "arco" (método "arc"). Luego se definen los estilos tanto de la línea como de su relleno y se procede a dibujarlos usando los métodos descritos anteriormente.

Canvas 2D posee muchos más métodos para dibujar diferentes tipos de líneas, además de poder dibujar también imágenes dentro de formas con el método "drawImage". Para simular el movimiento de los elementos, lo que se hace es borrar y volver a dibujar los elementos en las nuevas posiciones.

3.2.2.1.2 Captura y reproducción de audio con la API de audio

En el controlador de la interfaz "AudioInterface" se implementa cómo se hace la captura de audio. JavaScript posee una API para poder acceder a elementos multimedia como el audio y el vídeo. El método que permite hacer eso se llama "getMedia" del objeto global "navigator". Y en él se puede definir si se quiere capturar audio, vídeo o los dos. Esta orden se manda al navegador, que en el caso de pedir captura de audio, pregunta al usuario los permisos para usar el micrófono, por ejemplo. JavaScript también permite grabar audio mediante la clase/prototipo "MediaRecorder". Para reproducir el audio, existe una nueva etiqueta en HTML 5 llamada "audio", en la cual mediante JavaScript y el método "play" puede hacer reproducir un audio.

3.2.2.2 Implementación de la vista

La vista se corresponde con la interfaz gráfica del cliente. Como el cliente se ejecuta en un navegador web, la interfaz gráfica se define mediante los lenguajes HTML y CSS. HTML es un lenguaje en el que, mediante etiquetas, se definen los elementos que componen la interfaz gráfica. En nuestro caso, estos elementos se definen en el archivo "index.html". Luego, mediante el lenguaje CSS, se define para cada etiqueta o elemento cual es su diseño gráfico.

En el diseño explicado en los diagramas de clases, se describió cómo se estructuraba en clases las interfaces de la parte cliente del videojuego. Cada interfaz disponía de un controlador, y algunas podían tener una interfaz gráfica. De esas interfaces, las que tienen interfaz gráfica son: AudioInterface, PlayerDataInterface, NotificationInterface, RankingInterface, ResultsInterface, ResurrectInterface y WaitingInterface. Los elementos de estas interfaces gráficas están definidos en el archivo index.html y su diseño en los archivos de extensión ".css" con el mismo nombre de la interfaz.

Un ejemplo es la definición de la interfaz gráfica de AudioInterface. Sus elementos se definen en el archivo index.html y luego el diseño de esos elementos se define en el siguiente archivo audioInterface.css:

```css
#audioPanel
{
  border:none;
  width: 100%;
  position:absolute;
  z-index:1;
  top: 91%;
  left: 0%;
  min-width: 300px;
}

#audioText
{
    display: block;
    background-color: rgba(255,255,255,0);
    color: black;
    opacity: 0;
    text-align:center;
    width: 100%;
}


#audioButton
{
    display: block;
    color: black;
    background-color: rgba(80,80,80, 0.4);
    font-size: 150%;
    text-align: center;
    width: 100%;
    height: 50px;
}

#audioElement
{
    display: none;
}
```

Figura 3.24 Ejemplo de CSS

Como se puede observar, en el archivo viene definido el diseño de cada elemento de la interfaz gráfica de AudioInterface. Por ejemplo “audioPanel” representa el panel de audio al completo donde el usuario puede ver un botón para pulsarlo cuando quiere grabar su voz para chatear, y puede ver el nick del jugador que está hablando en ese momento. En él se definen una serie de

atributos como son por ejemplo su ancho de tamaño (atributo “width”), o su posición (atributos “position”, “top” y “left”). Luego también se define el diseño del botón de audio “audioButton” y el texto donde se muestra el nick del jugador que habla “audioText”. Se puede observar que hay atributos para definir su color (atributo “color”), su color de fondo (atributo “background-color”), además de muchos más atributos.

3.2.3 Implementación de la parte cliente en dispositivo móvil

A continuación se expone el diagrama de despliegue en UML de la parte cliente del videojuego en un dispositivo móvil como puede ser un smartphone o una tablet:

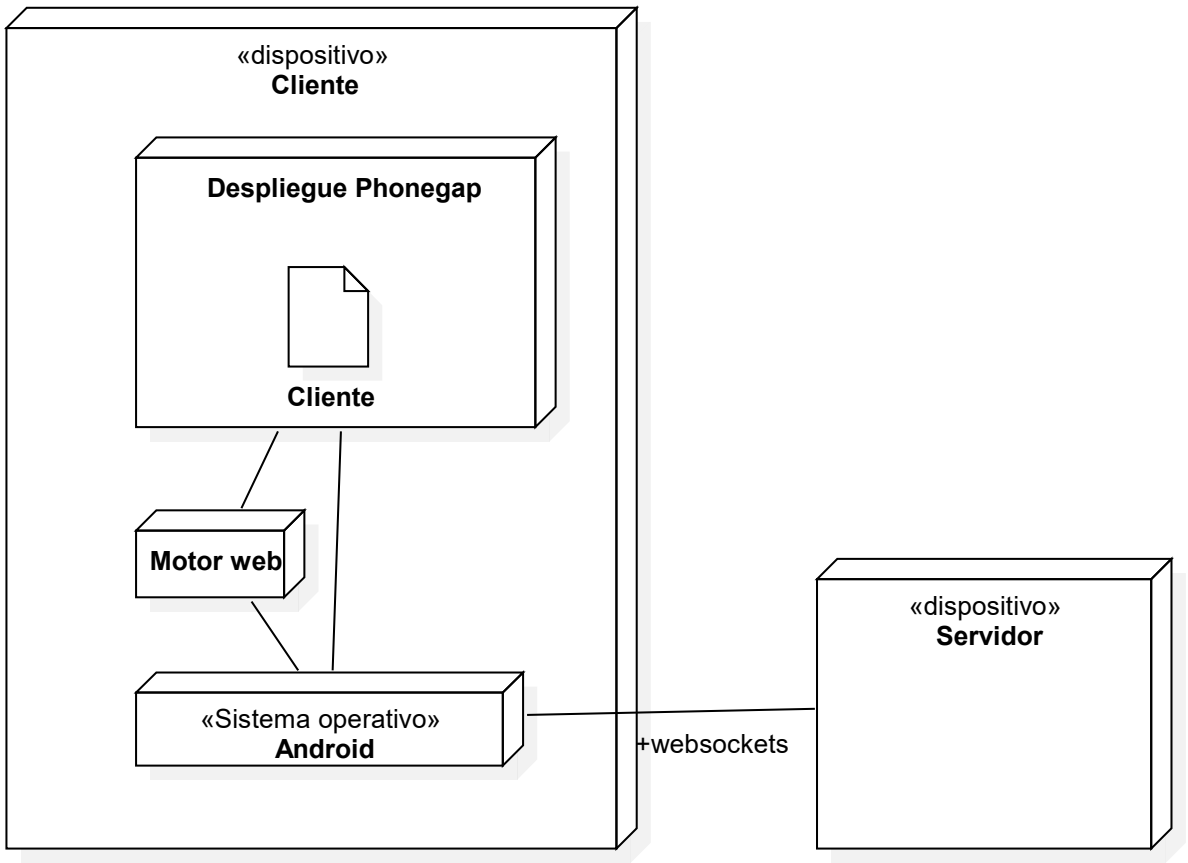


Figura 3.25 Diagrama de despliegue del cliente móvil

Como se puede observar en el diagrama, existe un artefacto llamado “Cliente” que es exactamente el mismo que el artefacto descrito en la implementación de la parte cliente del videojuego en el PC. Se puede usar el mismo código que en el cliente PC porque este código se despliega en el dispositivo móvil usando un framework llamado “Phonegap”.

PhoneGap es propiedad de la empresa Adobe, pero se distribuye bajo código abierto por la fundación Apache con el nombre de “Cordova”. Este framework permite usar la implementación hecha anteriormente mediante HTML, CSS y JavaScript y transformarla en una aplicación híbrida. Una aplicación híbrida es en parte una aplicación web y en parte una aplicación nativa del dispositivo móvil. Es una aplicación web porque para las interfaces gráficas y sus controladores escritos en lenguaje HTML, CSS y JavaScript se usa el motor web instalado en el dispositivo móvil, que es el mismo motor web que usan las aplicaciones de navegador en el dispositivo móvil. En el diagrama se puede ver cómo la aplicación Phonegap se comunica con el motor web del sistema del dispositivo. Y también es una aplicación nativa porque para algunas funciones del controlador que requieren acceder a funciones nativas del dispositivo como por ejemplo a los sensores y a los botones, utiliza directamente la API del sistema operativo (por ejemplo Android). Para poder hacer esto, Phonegap proporciona una serie de plugins que permite acceder, mediante funciones de JavaScript, a funciones del sistema operativo del dispositivo móvil.

Se necesitan entonces eventos en JavaScript que se lancen cuando por ejemplo el usuario toca un botón y manejadores que sepan tratar esos eventos. El evento fundamental de JavaScript que se puede manejar en Phonegap se llama “deviceready”. Sin él, no se puede acceder a ningún evento o función de JavaScript que permite el acceso a funciones nativas. En la siguiente imagen se puede ver cómo se trata este evento:

```
document.addEventListener("deviceready", onDeviceReady, false);
function onDeviceReady() {
    var options = { frequency: ACCELEROMETER_TIME};
    var watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError, options);

    document.addEventListener("backbutton", onBackButton, false);
    navigator.app.overrideButton("menubutton", true);
    document.addEventListener("menubutton", onMenuButton, false);
}
```

Figura 3.26 Fragmento de código Phonegap

Como se puede ver, dentro del manejador del evento deviceready llamado "onDeviceReady" se pueden tratar otros eventos nativos como por ejemplo "backbutton" y "menubutton". "backbutton" es el evento que se activa cuando se pulsa el botón de ir hacia atrás que tienen los dispositivos móviles, y que en este caso lo que hace es activar el manejador "onBackButton" que hace salir de la partida al jugador. Y "menubutton" es el evento que se activa cuando se pulsa el botón de menú que tienen los dispositivos móviles y que en este caso lo que hace es activar el manejador "onMenuButton" que hace que el jugador lance una bomba.

Para poder mover el jugador en el dispositivo móvil no se dispone del ratón que se tiene en el PC, así que se va a usar el acelerómetro del dispositivo. El acelerómetro es un dispositivo que mide la aceleración en cada una de las tres coordenadas espaciales X, Y y Z. Por ejemplo si un smartphone está en posición vertical, la aceleración en Y valdrá 9,8 que es la aceleración de la gravedad. Con Phonegap, para poder acceder a estos valores del acelerómetro, se debe usar un plugin que se llama "device motion". Este plugin proporciona una función en JavaScript que se puede ver en la imagen anterior y que se llama "watchAcceleration". En la función "onSuccess" se puede acceder a un objeto llamado "acceleration", y que tiene como atributos las variables X, Y y Z del acelerómetro. Según estos valores, se puede saber en qué dirección y con qué velocidad quiere moverse el jugador.

Otro elemento importante de un proyecto hecho con Phonegap es el fichero "config.xml". En él se puede configurar mediante el lenguaje XML diversas opciones de configuración para la aplicación Phonegap. Por ejemplo, se puede establecer el nombre con el que se verá la aplicación en el dispositivo, su icono, si se quiere que aparezca por defecto en pantalla completa y en horizontal, etc.

Por defecto la aplicación móvil resultante tendrá todos los permisos para acceder a las funciones nativas del dispositivo, pero si se prefiere que haya funciones que no estén disponibles, se debe modificar el archivo AndroidManifest.xml del código resultante al generar el código en Android.

CAPÍTULO 4. PRUEBAS

Una parte muy importante del desarrollo de software es la realización de pruebas. Las pruebas permiten averiguar si el software cumple con los requisitos. En este proyecto se han realizado pruebas automáticas y pruebas manuales. En los dos siguientes apartados se hablará de cada una de estas pruebas.

4.1 Pruebas automáticas

Las pruebas automáticas son pruebas realizadas mediante software, que de forma automática verifica que el software funciona de acuerdo a los requisitos. Estas pruebas se dividen en las siguientes:

- Pruebas unitarias.
- Pruebas de integración.

Al tratarse de un software que está dividido en dos partes: cliente y servidor, las pruebas anteriores se dividen a su vez en las siguientes:

- Pruebas unitarias de la parte servidor.
- Pruebas de integración de la parte servidor.
- Pruebas de integración de la parte cliente.

Para realizar estas pruebas se ha usado una herramienta de prueba del software escrito en lenguaje JavaScript para Node Js que se llama Mocha. A continuación se procede a explicar esta herramienta y cada uno de los tipos de pruebas que se han realizado.

4.1.1 Mocha

Mocha es un software con el que, de forma automática, se puede probar software escrito en JavaScript. Funciona tanto para Node Js como para los navegadores web. Para poder usarlo, se descarga e instala como un paquete de Node Js utilizando para ello la herramienta NPM que viene junto con el software de Node. Si se escribe el siguiente comando en una consola, situándonos en el directorio donde se encuentra el software de la parte servidor, se instala el paquete que contiene Mocha como un módulo de Node Js:

npm install mocha –save

Para explicar cómo se utiliza esta herramienta, se expone un trozo de código de ejemplo:

```
var assert = require('assert');
describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present', function() {
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Figura 4.1 Ejemplo de código de Mocha

Como se puede observar, las pruebas se agrupan en conjuntos de pruebas que tienen un nombre. Esto se hace mediante la función de JavaScript “describe”. Dentro de cada conjunto de pruebas pueden haber más conjuntos de pruebas. Las pruebas se declaran con la función “it”, que toma como argumentos el nombre de la prueba y la función en la que se ejecuta. Para comprobar si el resultado de la prueba es válido o no, se puede hacer uso del módulo de Node Js llamado “assert”. En el ejemplo de la imagen anterior, existe un conjunto de pruebas llamado “Array”, que a su vez contiene un conjunto de pruebas llamado “#indexOf()”. Este último conjunto de pruebas contiene una prueba en el que se comprueba si un número pertenece a un array numérico.

Para ejecutar las pruebas escritas de esta forma, dentro del archivo package.json del paquete Node Js que queremos testear, se escribe lo siguiente:

```
"scripts": {
    "test": "mocha"
  }
```

Figura 4.2 Código para ejecutar Mocha

Entonces si usando una consola nos situamos en el directorio donde se encuentra el paquete de Node Js que queremos testear y escribimos el siguiente comando se ejecutan las pruebas:

npm test

Dando como resultado lo siguiente:

```
Array
  #indexOf()
    ✓ should return -1 when the value is not present


1 passing (9ms)
```

Figura 4.3 Ejemplo de resultados de Mocha

Cuando una prueba es válida, aparece el símbolo de verificación junto con el nombre de la prueba y cuánto ha tardado en ejecutarse en milisegundos. Al final de todas las pruebas aparece la cantidad de pruebas que han resultado válidas (donde pone “passing”).

Si se quiere testear software asíncrono, lo cual es muy habitual en JavaScript, se puede hacer lo siguiente:

```
describe('User', function() {
  describe('#save()', function() {
    it('should save without error', function(done) {
      var user = new User('Luna');
      user.save(function(err) {
        if (err) done(err);
        else done();
      });
    });
  });
});
```

Figura 4.4 Ejemplo de Mocha con código asíncrono

En este ejemplo, user.save es una función asíncrona, es decir, no se ejecuta inmediatamente, sino que se ejecuta cuando se recibe un evento que ejecuta la función manejadora "function(err)". Cuando se ejecuta la función manejadora, si la variable "err" que representa que ha ocurrido un error, es verdadera, ejecuta la función "done" con un argumento de entrada. Cuando la función "done" se ejecuta con un argumento de entrada, sea el que sea, la prueba resultará no válida, y si se ejecuta sin argumentos, la prueba resultará válida.

4.1.2 Pruebas unitarias de la parte servidor

Para realizar las pruebas unitarias de la parte servidor, se han realizado pruebas para cada una de las clases/prototipos de JavaScript que existen en el código de la parte servidor del videojuego. Así, dentro de la carpeta "test" del código de servidor, se han incluido los siguientes archivos: testBall.js, testPlayer.js, testBot.js, testBomb.js, y testDatabaseServer.js. Cada archivo hace pruebas unitarias de la clase/prototipo correspondiente. Existen clases/prototipos de los que no se han hecho tests unitarios porque la prueba de dichos prototipos es trivial al solo contener métodos get/set para obtener y establecer atributos. Además, no se ha hecho prueba unitaria de la clase Game porque no se puede probar de forma separada de las demás clases, así que ya se prueba en las pruebas de integración.

Un ejemplo de un trozo de los resultados de las pruebas de testPlayer.js (pruebas de la clase/prototipo Player) es el siguiente:

```
#attackPlayer(player)
  √ debería no atacar al jugador si el jugador ya está muerto
  √ debería no atacar al jugador si es el mismo
  √ debería no atacar al jugador si no está en la distancia adecuada
  √ debería no atacar al jugador si tiene la distancia adecuada pero menor radio
  √ debería matar al jugador atacado y el atacante crecer en area, puntos, etc adecuadamente
```

Figura 4.5 Ejemplo de resultados de test de Player.js

Esta prueba se corresponde con las pruebas del método “attackPlayer” de la clase/prototipo Player. No es posible realizar todas las pruebas posibles, porque son infinitas, pero sí las más relevantes y las que podrían dar más problemas.

4.1.3 Pruebas de integración de la parte servidor

En las pruebas de integración, se hacen pruebas en las que interviene más de una clase/prototipo. En el caso de este proyecto, las pruebas de integración de la parte servidor se han hecho sobre la clase Game, que es la clase que integra a las demás clases y que no se puede probar de forma unitaria sin las demás. Las pruebas se han realizado de forma que se pudiera verificar que se cumple el comportamiento especificado en los casos de uso en la parte de servidor, y están contenidas dentro del archivo “testUseCases.js”.

Por ejemplo, se hacen pruebas para comprobar que en la parte servidor se ejecuta correctamente los casos de uso “Registro de jugador” y “Login de jugador”:

```
Tests de casos de uso
  Registro de jugador
    √ Si un jugador intenta registrarse con un nick y una contraseña que no existen, permite el registro (602ms)
    √ Si un jugador intenta registrarse con un avatar, éste se crea en el directorio correspondiente (334ms)
    √ Si un jugador se intenta registrar con un nick que ya existe y con su misma contraseña, cambia el avatar po
uevo (372ms)
    √ Si un jugador se intenta registrar con un nick que ya existe pero no coincide la contraseña, no deja que se
:re (389ms)
  Login de jugador
    √ Si un jugador intenta loguearse con un nick y contraseña y ya está registrado con estos datos, le deja logu
(410ms)
    √ Si un jugador intenta loguearse con un nick y contraseña y el nick está registrado pero la contraseña es in
:a, el juego no le deja loguearse (374ms)
    √ Si un jugador intenta loguearse con un nick y contraseña y el nick no está registrado, el juego no le deja
'se (334ms)
    √ No deja loguearse en el juego a más jugadores de los permitidos (7411ms)
```

Figura 4.6 Ejemplo de resultados de testeo de dos casos de uso

Para simular las conexiones de varios clientes, se hace uso de los websockets de Socket.io por la parte cliente, usando el paquete “socket.io-client”, descargado e instalado mediante NPM.

En la siguiente imagen se puede observar un trozo del código de prueba del caso de uso “Login de jugador”, en la secuencia en la que se tratan de loguear más usuarios de los que pueden estar en la sala:

```
it("No deja loguearse en el juego a más jugadores de los permitidos", function(done){
    this.timeout(20000);
    var datas = [];
    var playersNotLoggedIn = 0;
    var numPlayersTest = 20;
    for(var i = 1 ; i <= numPlayersTest; i++)
    {
        sockets[i] = io.connect("https://localhost:3000", { secure: true, reconnect: tru
        datas[i] = {nick: "jugador" + i, password: "123"};
        sockets[i].emit("readyPlayer", {windowWidth: 1024, windowHeight: 768});
        sockets[i].emit("playerLoginAttempt", datas[i]);
        sockets[i].on("playerNotLoggedIn", function(){
            playersNotLoggedIn++;
            if(playersNotLoggedIn === numPlayersTest + global.game.getConfiguration().ge
            {
                done();
            }
        });
    }
});
```

Figura 4.7 Prueba del logueo de más jugadores de los permitidos

Como se puede ver, en la variable “numPlayersTest” se puede especificar el número de jugadores o clientes que se quiere simular que se conectan al servidor para loguearse en el juego. Esta prueba automática permite hacer algo que no se puede realizar en una prueba manual, como es la conexión de cientos de jugadores, y comprobar si funciona correctamente. Cada variable “sockets[i]” representa un websocket cliente de un jugador, y mediante la función asíncrona “on” se puede comprobar qué clientes/jugadores no han podido loguearse si han recibido el mensaje de websocket “playerNotLoggedIn”. Si finalmente la prueba es correcta, se ejecutará la función “done” sin argumentos, lo cual hará que la prueba termine y muestre por consola que la prueba es correcta.

Si se quiere ejecutar un test de los especificados en los dos apartados anteriores, solo hay que irse al archivo package.json y observar el siguiente código:

```
"scripts": {
  "start": "node app.js",
  "test": "mocha test/testUseCases.js"
},
```

Figura 4.8 Código para ejecutar un test concreto de Mocha

Dentro de “test” se pone lo que se ejecuta cuando se escribe el comando “npm test” para hacer tests del paquete a probar. En este caso, al ejecutar ese comando, se harían los tests de integración contenidos en “testUseCases.js”. Si por ejemplo se quiere realizar la prueba unitaria de la clase Player, habría que cambiar esa línea de código para poner el archivo “testPlayer.js” en lugar del anterior.

4.1.4 Pruebas de integración de la parte cliente

Para probar la parte cliente del proyecto, se han realizado pruebas de integración. No se pueden realizar pruebas unitarias porque cada controlador del patrón de diseño modelo-vista-controlador no se puede ejecutar de forma independiente de los demás. Además, la parte de modelo carece de funcionalidad más allá de los métodos get/set para obtener sus atributos.

Estas pruebas de integración se han realizado simulando el servidor, creando una instancia de la clase Server. Para probar que la parte cliente funciona correctamente, se tienen que observar dos cosas:

1) Que el servidor recibe los mensajes de websocket adecuados por parte del cliente.
2) Que en el cliente se puede visualizar gráficamente de forma correcta el resultado de la recepción de los mensajes de websocket por parte del servidor.

Para hacer todo esto, se han codificado las pruebas en un archivo llamado "testClient.js". En él se comprueban si los casos de uso se ejecutan correctamente en la parte cliente. Por ejemplo, con el caso de uso "Registro de jugador" se ejecuta la siguiente prueba, que es una prueba del tipo 1) explicado anteriormente:

```
describe("Registro de jugador", function(){
    it("Se reciben los datos del jugador a registrar", function(done){
        this.timeout(300000);
        console.log("Intente registrar un jugador introduciendo los datos en el formulario");
        socket.on("playerRegisterAttempt", function(data){
            if(data.hasOwnProperty("nick") && data.hasOwnProperty("password")
                    && data.hasOwnProperty("imgData"))
            {
                setTimeout(function(){
                    socket.emit("playerRegistered");
                    socket.removeAllListeners();
                    done();
                }, 3000);
            }else{
                socket.removeAllListeners();
                done("Error, no se han recibido correctamente los datos de registro");
            }
        });
    });
```

Figura 4.9 Ejemplo de prueba en Mocha del caso de uso Registro de jugador

Cuando se ejecutan las pruebas, de forma guiada va diciendo al usuario qué es lo que debe hacer en la interfaz. Por ejemplo en la prueba de registro de jugador anterior, le pide que intente registrar un jugador introduciendo los datos en el formulario. Al introducirlos, si la prueba es correcta, se recibirá en el servidor un mensaje de websocket llamado "playerRegisterAttempt" en el que se

envían el nick, la contraseña y el avatar. Si es así, se ejecutará la función “done” sin argumentos, lo cual escribirá por consola que la prueba es correcta, y si no, se ejecutará la función “done” con argumentos, lo cual escribirá por consola que es incorrecta.

Una prueba de tipo 2) sería por ejemplo la comprobación de que cuando una bola es destruida por un jugador que se la come o por una bomba, debe desaparecer del gráfico del juego si está en el área visible del jugador. El código de esta prueba es el siguiente:

```
it("Cuando una bola es comida por un jugador o le afecta la explosión de una b
    this.timeout(300000);
    console.log("Compruebe que desaparece la bola blanca y la bola verde");
    socket.emit("deletedBall", ball5.getId());
    socket.emit("deletedBall", ball2.getId());

    setTimeout(function(){
        socket.emit("playersPlayed");
        setTimeout(function(){
            done();
        }, 3000);
    }, 5000);
});
```

Figura 4.10 Prueba de que una bola desaparece gráficamente cuando es destruida

Como se puede observar en el código, se envían mensajes de websocket “deletedBall” que ocurren cuando una bola es destruida, y de forma guiada se pide al usuario que compruebe si desaparecen del gráfico las dos bolas borradas. Para hacer las pruebas en el cliente móvil se hace lo mismo pero ejecutando la parte cliente en el dispositivo móvil Android.

4.2 Pruebas manuales

Además de las pruebas automáticas, se han realizado pruebas manuales. Para hacer esto, se ha jugado al videojuego como un usuario normal comprobando que no ocurría ningún comportamiento inesperado. No se pueden comprobar todas las posibilidades de uso, pero sí las más relevantes. No se puede comprobar de forma manual la situación de varios jugadores jugando a la vez, por falta de personas, pero se debe recordar que estas pruebas se han realizado ya de forma automática simulando las conexiones de los jugadores mediante software. Se han hecho pruebas en dos navegadores web distintos: Google Chrome y Mozilla Firefox. En Internet Explorer no, porque el videojuego no está hecho para ejecutarse en Internet Explorer al no disponer todavía este navegador de algunas funcionalidades ya estandarizadas. También se ha probado en dos dispositivos móviles (smartphones) Android distintos.

Capítulo 5. Conclusiones y trabajo futuro

5.1 Conclusiones

Las conclusiones que se obtienen del desarrollo de este software son las siguientes:

1) Tener un único lenguaje de programación para todo el proyecto (JavaScript) resulta muy beneficioso. Ahorraría costes si se tratase de un proyecto llevado a la realidad, ya que no habría que contratar a gente que supiera desarrollar con experiencia en otros lenguajes de programación, ni formar a los trabajadores existentes en nuevos lenguajes, lo cual llevaría tiempo y dinero.
2) El modelo de programación asíncrono de JavaScript es difícil de aprender y comprender al principio para los que están acostumbrados a programar en tecnologías síncronas. Pero una vez aprendido, resulta en un código fácil de escribir y entender.
3) Node Js tiene sus ventajas y sus desventajas. Como ventajas se puede citar que no hay que lidiar con la complejidad de una tecnología basada en hebras, en la que habría que tener cuidado con la sincronización entre ellas. Node Js, al tener una única hebra, no tiene que guardar espacio de memoria ni de procesador para ejecutar cada hebra. En el caso de una tecnología multi hebra, se ocupan recursos cuando por ejemplo una hebra está esperando a operaciones de entrada/salida, y esto es especialmente ineficiente. En cambio, con Node Js y su modelo de eventos, solo se ejecutaría código y se ocuparía la memoria correspondiente cuando se activa el manejador del evento. Si por ejemplo se quiere hacer una petición al gestor de base de datos, se hace la petición y el hilo de ejecución sigue ejecutando otras cosas hasta que recibe el evento del gestor de base de datos de que ya ha terminado de realizar la tarea. Node JS tiene como desventajas que si existe una tarea de mucho tiempo de ejecución que no se corresponde con una tarea de entrada/salida, no puede aprovechar el modelo de hebras del sistema operativo para dividir dicha tarea en varias que se ejecuten de forma concurrente y así disminuir su tiempo de ejecución. En el proyecto solo existe una tarea que puede tener mucho tiempo de ejecución: la de realizar una jugada. Pero es poco divisible en varias tareas concurrentes, se tratan de tareas síncronas. Por ejemplo, un jugador no se puede mover hasta que se han movido los otros, no se pueden ejecutar de forma concurrente.

4) MongoDB también tiene sus ventajas y sus inconvenientes. Como ventaja es de destacar que utiliza el modelo de datos de JavaScript (JSON), lo cual es beneficioso porque no hay que aprender distintos lenguajes. También es ventajoso que sea capaz de leer bases de datos muy grandes, eliminando el coste de unir varias tablas que ocurre en una base de datos puramente relacional. Es útil usar este gestor de base de datos para el proyecto porque pueden haber muchos usuarios, partidas y datos de partidas, que al estar todos estos datos juntos, se pueden leer mucho más rápido. La gran desventaja que tiene MongoDB es que es un gestor de base de datos diseñado para un propósito específico, no sirve para todos los tipos de bases de datos. El principal inconveniente es que las bases de datos en MongoDB pueden no estar normalizadas, lo cual da muchos problemas en bases de datos que requieran de muchas escrituras, porque al no estar normalizadas, ocurre la redundancia de datos y puede que ocurra que haya que actualizar los mismos datos en distintos sitios. Otro problema que existe es que MongoDB no soporta transacciones (ni debería soportarlas, porque MongoDB está hecho para bases de datos donde haya que leer mucho y escribir poco). Y al no soportar transacciones, puede ocurrir que la base de datos se encuentre en estados inconsistentes, es decir, con escrituras que no estén terminadas de escribir del todo. Además hay que añadir que el lenguaje de consultas que utiliza no es estándar y es único para MongoDB.
5) Los plugins de navegador para desarrollar software de vídeo interactivo en la parte cliente están en proceso de extinción, y es una buena noticia. Aunque en mi opinión las nuevas tecnologías de HTML5 todavía tienen que madurar. Muchas están recién estandarizadas y algunos navegadores todavía no implementan todas las funcionalidades deseadas. Es el caso de grabar audio en el navegador Internet Explorer, que está recién estandarizado pero no implementado en dicho navegador. En Canvas 2D se echa de menos que los elementos del dibujo se puedan guardar de alguna forma como objetos en el software para luego manipularlos fácilmente. Si por ejemplo dibujas una bola, luego para borrarla no existe un método directo para hacerlo, sino que debes borrarla pintando un rectángulo blanco encima.
6) El uso de Phonegap para poder usar el videojuego en un dispositivo móvil ahorra mucho coste de aprendizaje y programación en el lenguaje de Android o de cualquier otro sistema operativo como iOS en el que también se pueden desplegar las aplicaciones desarrolladas para Phonegap. El problema que existe es que un cliente desarrollado con tecnologías web para desplegarlo con Phonegap en un dispositivo móvil no es una aplicación nativa del dispositivo móvil. Es más bien

un cliente web incrustado dentro de una aplicación nativa (lo que se llama una aplicación híbrida), en la que solo se hace uso de las funcionalidades nativas del sistema operativo móvil para acceder al hardware del dispositivo, como por ejemplo al acelerómetro. Una aplicación cliente nativa en Android por ejemplo, podría ser mucho más vistosa gráficamente al poder usar las interfaces gráficas que proporciona Android y no las que proporciona HTML y CSS.

5.2 Trabajo futuro

Si se quisiera en un futuro ampliar las funcionalidades del videojuego, se podrían realizar las siguientes ampliaciones:

1) Desarrollar las funcionalidades que permiten que en el videojuego se pueda loguear un administrador que pueda realizar diversas acciones como por ejemplo: cambiar la configuración del juego (cambiar el tamaño del tablero, de las bolas y jugadores, el tiempo de las partidas, etc), poder ver en cualquier momento qué jugadores están jugando y poder modificar sus atributos o hacer tareas de moderación para poder echar a un jugador por mal comportamiento, poder ver los resultados de cada jugador en cada partida y diversas estadísticas en cualquier momento, etc.
2) Implementar medidas de seguridad, como por ejemplo para que cuando un usuario se registra o se loguea no pueda introducir código o algún elemento extraño con el que pueda obtener información que no debería estar disponible para él o provocar que el juego no funcionase.
3) Si se dispone de un hardware de servidor con varios núcleos y mucha potencia, se podría intentar desarrollar la funcionalidad en la que se pudieran crear salas de juego, y así aumentar el número de jugadores que pueden jugar a la vez. Para aprovechar un hardware de varios núcleos, se podrían crear procesos de Node JS distintos que se comunicaran entre sí, cada uno ejecutándose en un núcleo. El problema es que quizás sería mejor implementar el software con alguna tecnología multihilo, porque los procesos ocupan mucha más memoria y tiempo de ejecución y de intercambio en el procesador que los hilos.
4) Se podría probar a implementar una aplicación nativa de Android u otro sistema operativo móvil, que permitiría usar las interfaces gráficas de ese sistema operativo.
5) Introducir más tipos de jugadores o de enemigos. Como por ejemplo bots que se movieran

de forma más inteligente, o que los jugadores pudieran lanzar veneno que hiciese que los jugadores envenenados fueran perdiendo tamaño.

APÉNDICE A. Contenido del CD y Manual de instalación

A.1 Contenido del CD

El CD que acompaña a la presente memoria contiene 3 carpetas y la memoria del proyecto.
Las tres carpetas son las siguientes:

- Cliente. Contiene el código fuente de la parte cliente del proyecto. Contiene a su vez el archivo "config.xml" que es el archivo de configuración de Phonegap con las opciones de configuración particulares para el proyecto, y una carpeta llamada "www" que contiene el código fuente. El código fuente contiene el archivo index.html de inicio y una carpeta para el código en JavaScript (carpeta "js"), otra para el código en CSS (carpeta "css") y una carpeta que contiene el icono para la aplicación de Phonegap (carpeta "img").
- Servidor. Contiene el código fuente de la parte servidor del proyecto en los archivos con extensión ".js", y además contiene la carpeta "img" con una colección de imágenes para usar como avatares de los bots, la carpeta "sslcert" con el certificado de clave pública autofirmado y la clave privada, la carpeta "test" con el código del software de prueba, el archivo "package.json" con la definición del paquete Node JS, el archivo "Install Node packets" que sirve para instalar los paquetes de Node JS que se necesitan en la parte servidor, el archivo "databaseServerAuthInit.bat" que inicia el gestor de bases de datos MongoDB y el archivo "gameInit.bat" que sirve para ejecutar el servidor web y el juego en el servidor.
- Instalacion. Contiene todo lo referente a la instalación e inicialización de todo lo necesario para poder jugar al juego. En el manual de instalación se cuenta para qué sirve cada archivo y cómo se debe utilizar.

A.2 Manual de instalación

A.2.1 Manual de instalación de la parte servidor

Para instalar e inicializar la parte servidor del videojuego, hay que seguir los siguientes pasos:

1) Descargar e instalar Node JS en la máquina que se vaya a utilizar como servidor o el hosting de Internet. Actualmente se puede descargar del siguiente enlace: https://nodejs.org/es/download/ . Pulsando sobre "Windows Installer" se descarga el instalador para Windows y solo hay que ejecutarlo y seguir las instrucciones de instalación. Con esto se instala tanto Node JS como el descargador e instalador de paquetes de Node JS, llamado "npm".
2) Copiar en el servidor la carpeta "Servidor" que viene en el CD, y descargar e instalar los paquetes de Node JS necesarios abriendo una consola, situándonos dentro de la carpeta Servidor y escribiendo "npm install". O bien haciendo doble click sobre el archivo "Install Node packets.bat" que viene dentro de la carpeta Servidor y que ejecuta lo mismo. Al hacer esto, comprobar que dentro de la carpeta Servidor se ha creado una nueva carpeta llamada "node_modules". Comprobar que dentro de dicha carpeta existen las carpetas: "express", "mocha", "mongodb", "socket.io", "socket.io-client" y "uniqid" donde cada una corresponde con un paquete de Node JS.
3) Descargar e instalar el gestor de bases de datos MongoDB. En la actualidad se puede descargar de la siguiente dirección: https://www.mongodb.com/download-center#community . Seleccionar la versión para Windows y pulsar sobre "Download" para descargar el instalador. Una vez descargado, ejecutarlo y seguir los pasos indicados.
4) Crear una carpeta de nombre "data" dentro de la carpeta "Servidor". Ir a la carpeta "Instalacion" y editar el archivo de nombre "databaseServerInit.bat". Se puede ver que en ese archivo existe el siguiente comando para ejecutar el gestor de bases de datos tomando la carpeta "data" como almacén de los datos:

   "C:/Program Files/MongoDB/Server/3.4/bin/mongod.exe"
   --dbpath=C:/Users/Antonio/Documents/Proyecto/Servidor/data

   Hay que editar la dirección del gestor de bases de datos "mongod.exe" si es otra distinta a la establecida aquí, y hay que cambiar la dirección de la carpeta "data" (--dbpath) por la

dirección adecuada de la carpeta “data” recién creada. Una vez hecho esto, guardar los cambios y hacer doble click sobre el archivo para ejecutar el comando.

5) El comando anterior lo que hace es ejecutar el gestor de bases de datos sin autenticación de usuarios, de forma que se pueda crear un nuevo usuario del gestor de bases de datos. En este paso lo que se va a hacer es crear un usuario nuevo en el gestor. Para ello hay que irse a la carpeta “Instalacion” y editar el archivo “databaseUserCreate.bat”. Este archivo contiene la siguiente línea de comandos:

   "C:/Program Files/MongoDB/Server/3.4/bin/mongo.exe" < DatabaseUserCreate.js

   mongo.exe es un cliente de consola para acceder al gestor de bases de datos MongoDB y se le pasa como entrada el archivo “DatabaseUserCreate.js” que contiene lo siguiente:

   ```
   use webgamedb
   db.createUser({user: "antonio",pwd: "password",roles: [ {role: "dbOwner", db:
   "webgamedb"}]})
   ```

   Estas instrucciones en lenguaje de MongoDB lo que hacen es crear una base de datos de nombre “webgamedb” dentro de la carpeta “data” anterior, y crear el usuario “antonio” de contraseña “password” como administrador de esa base de datos. Se debe editar el contenido del archivo databaseUserCreate.bat y cambiar la dirección de “mongo.exe” por la dirección correcta si no es la misma que la de aquí. Si se prefiere otro nombre de usuario para la base de datos y/o otra contraseña, se debe editar el archivo “DatabaseUserCreate.js” y poner el usuario y la contraseña deseados. Además, se debe ir al archivo “DatabaseServer.js” del código fuente y buscar las siguientes líneas de código:

   ```
   this._user = "antonio";
   this._password = "password";
   ```

   Esas líneas de código se deben cambiar por el usuario y la contraseña deseados.
   Una vez hecho los cambios, se debe hacer doble click sobre el archivo databaseUserCreate.bat para ejecutar el comando y crear el nuevo usuario en el gestor.

6) Apagar el gestor de bases de datos ejecutado anteriormente, yéndonos a la ventana de la

consola, y pulsando “Ctrl + C”.

7) Volver a ejecutar el gestor de bases de datos, pero esta vez con la opción de que los usuarios deban autenticarse para poder usarlo. Para ello, primero hay que editar el contenido del archivo “databaseServerAuthInit.bat”, que se encuentra dentro de la carpeta “Servidor” y que contiene el siguiente comando:

   "C:/Program Files/MongoDB/Server/3.4/bin/mongod.exe"
   --dbpath=C:/Users/Antonio/Documents/Proyecto/Servidor/data –auth

   Hay que cambiar la dirección de “mongod.exe” por la dirección adecuada donde está ese ejecutable, y cambiar la dirección de la carpeta “data” por la dirección adecuada. Una vez hecho esto, guardar los cambios y ejecutar el archivo haciendo doble click en él. Se ejecutará el gestor de bases de datos.

8) En este paso se inicializa el contenido de la base de datos. Para ello, hay que editar el archivo “databaseReset.bat” que se encuentra dentro de la carpeta “Instalacion” el cual contiene el siguiente comando:

   "C:/Program Files/MongoDB/Server/3.4/bin/mongo.exe" -u "antonio" -p "password"
   --authenticationDatabase "webgamedb" < DatabaseReset.js

   Este comando ejecuta los comandos contenidos dentro del archivo DatabaseReset.js en el gestor de bases de datos MongoDB mediante el cliente de MongoDB llamado “mongo.exe”. Hay que cambiar la dirección de mongo.exe por la dirección adecuada, cambiar el nombre y la contraseña del usuario si es distinto y guardar los cambios. Una vez hecho esto, ejecutar el archivo haciendo doble click en él. De esta forma, se ejecutarán los comandos de MongoDB contenidos en DatabaseReset.js, que lo que hacen es borrar la base de datos si contenía algún dato, crear una nueva base de datos vacía con una colección llamada “players” (jugadores) e inicializarla con los datos de los bots que vienen por defecto y algunos jugadores normales (no bots) de prueba. Una vez hecho todo esto, ya se ha instalado la parte servidor del proyecto, y solo hay que ejecutar la parte servidor del videojuego luego tal y como se indica en el manual de usuario.

A.2.2 Manual de instalación de la parte cliente

Para instalar la parte cliente en el PC no hay que hacer prácticamente nada, solo copiar la carpeta "Cliente" que viene en el CD en el mismo directorio en el que se encuentra la carpeta "Servidor" copiada anteriormente.

Para instalar la parte cliente del videojuego en el dispositivo móvil, hay que seguir los siguientes pasos:

1) Descargar e instalar Phonegap en el PC. Para ello, primero hay que descargar e instalar Node JS tal y como se explicó en el apartado anterior. Luego hay que descargar e instalar Phonegap como un paquete de Node JS, mediante el siguiente comando de consola:

   npm install -g cordova

2) Abrir una consola de comandos e ir al directorio donde se copió la carpeta "Servidor". Una vez situado en ese directorio, ejecutar el siguiente comando:

   cordova create Cliente

   Este comando crea un directorio de nombre "Cliente" con un proyecto de Phonegap en su interior.
3) Comprobar que dentro de la carpeta "Cliente" existe una carpeta llamada "www" y un archivo llamado "config.xml". Sustituir ese archivo y esa carpeta por los del mismo nombre que están dentro de la carpeta "Cliente" del CD.
4) Añadir la plataforma Android al proyecto Phonegap para que después se pueda crear el ejecutable para Android. Esto se hace situándose primero dentro de la carpeta "Cliente" en una consola de comandos y ejecutar el comando siguiente:

   cordova platform add android

   Comprobar que después de ejecutar este comando, dentro de la carpeta "platforms" que está

dentro de la carpeta “Cliente” se ha creado una carpeta llamada “android”.

5) Descargar e instalar el SDK de Android y el JDK de Java.
6) Dentro de la carpeta “www” comprobar que existe el archivo index.html y editarlo para cambiar esta línea de código:

   <script src="/socket.io/socket.io.js"></script>

   Por la siguiente:

   <script src="https://direccion:3000/socket.io/socket.io.js"></script>

   Donde “direccion” es la dirección IP o el nombre de dominio del servidor web del proyecto. También hay que comprobar que dentro de “www/js/gameInterface” existe el archivo “SocketInterface.js”. Hay que editar este archivo, y sustituir la siguiente línea de código:

   this._socket = io.connect();

   Por la siguiente:

   this._socket = io.connect("https://direccion:3000");

   Donde “direccion” es la dirección IP o el nombre de dominio del servidor web del proyecto.

7) Ejecutar el comando siguiente:

   cordova build android

   Esto crea el archivo “android-debug.apk” dentro de la carpeta platforms/android/build/outputs/apk , que es la parte cliente ya construida para el dispositivo móvil.
8) Copiar el archivo anterior al dispositivo móvil Android, y hacer click sobre él para que se instale, y ya tenemos instalado el cliente para el dispositivo móvil.

APÉNDICE B. Manual de usuario

B.1 Manual de usuario del servidor

Para poder jugar al videojuego, hay que seguir estos pasos en el servidor:

1) Iniciar el gestor de bases de datos. Para ello, se hace doble click sobre el archivo "databaseServerAuthInit.bat", que está situado dentro de la carpeta "Servidor". Esto inicia el gestor y se conecta con el usuario especificado en la instalación.
2) Iniciar el servidor web y el videojuego. Para iniciar conjuntamente el servidor web y el videojuego, se tiene que hacer doble click sobre el archivo "gameInit.bat", el cual está situado dentro de la carpeta "Servidor".

Si en algún momento se quiere resetear la base de datos a su estado original, hay que hacer doble click sobre el archivo "databaseReset.bat" que se encuentra dentro de la carpeta "Instalacion".

B.2 Manual de usuario para el PC

Si se quiere jugar al videojuego con un PC, hay que seguir los siguientes pasos:

1) Instalar un navegador web que tiene que ser o "Google Chrome" o "Mozilla Firefox"
2) Con el navegador web, entrar en la dirección web establecida en la instalación del servidor, accediendo mediante "https". Si se está usando el propio PC como servidor con un certificado digital autofirmado, debe ignorarse la advertencia de que la conexión no es segura y continuar igualmente con la conexión.

Al realizar los dos pasos anteriores, debe salir una pantalla como la siguiente:

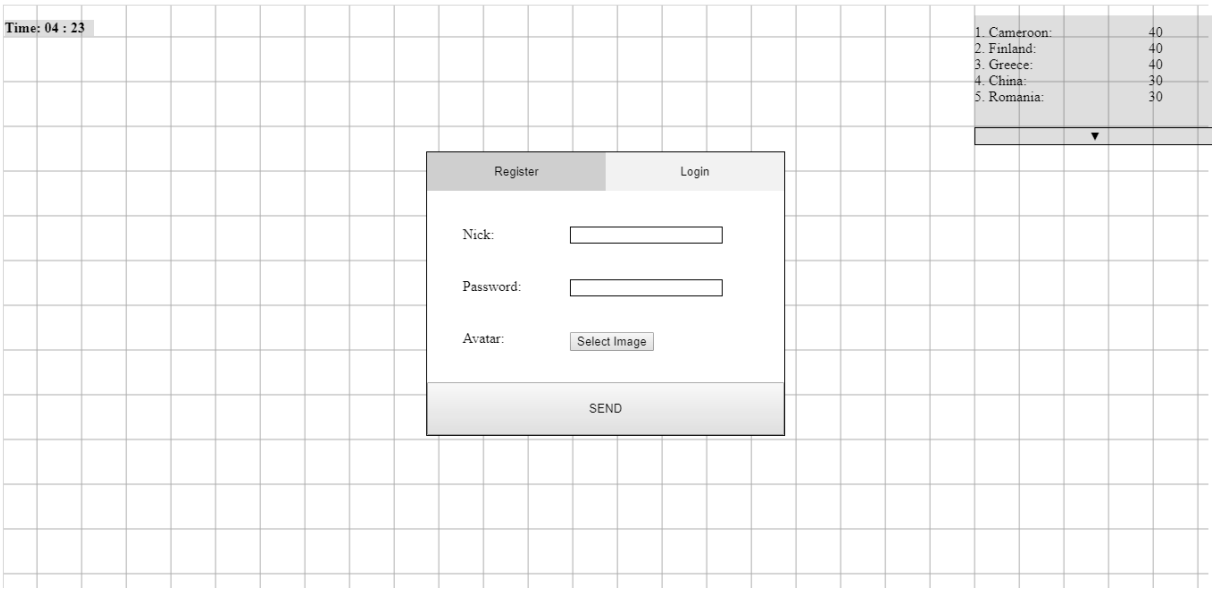


Figura B.1 Pantalla inicial del videojuego

Esta es la pantalla inicial del videojuego. Si aún no se tiene registrado ningún jugador en el videojuego, hay que irse al formulario que se encuentra en el centro, en la que hay dos pestañas que se llaman “Register” y “Login”. Para registrar un jugador, hay que rellenar el formulario de la pestaña “Register” con los datos que se piden: nick, password y avatar. Una vez rellenados los datos, pulsar sobre el botón “Send” que envía los datos al servidor para su registro. Si todo ha salido bien, se recibirá un mensaje en el que se muestra que el jugador se ha registrado correctamente. Si no, saldrá un mensaje de error.

Una vez registrado el jugador, se puede loguear en el juego entrando en la pestaña “Login” del formulario, en el que se pide el nick y la contraseña del usuario registrado, y pulsar sobre el botón “Send”. Si el jugador se ha logueado correctamente, el jugador entrará en la partida que esté en marcha y se verá algo como lo siguiente:

Figura B.2 Jugador logueado sin avatar

Como se puede ver, si el jugador se registró sin un avatar, debe aparecer un círculo de un color aleatorio con el nick del jugador en su interior. Si una vez que el jugador ya está registrado se quiere poner un avatar o cambiar el que ya se tenía, se debe entrar otra vez en la página inicial, pulsar sobre la pestaña "Register", introducir el nick, y la contraseña del jugador que se quiere modificar el avatar, el avatar nuevo y pulsar el botón "Send". Una vez hecho esto, al loguearse el jugador como se ha explicado anteriormente, aparecerá el jugador con el avatar establecido de fondo como se ve en la siguiente imagen:

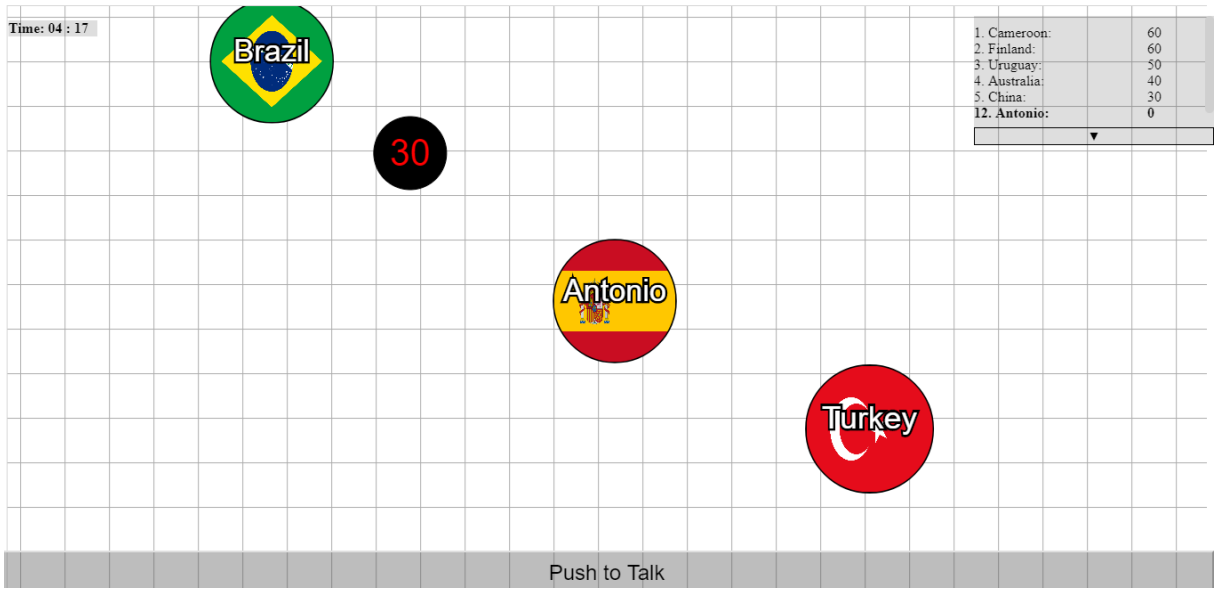


Figura B.3 Jugador logueado con avatar

Como se puede observar en la imagen, los demás jugadores aparecen también como unos círculos con su nick en su interior y su avatar. También, en la parte superior derecha, aparece una lista con los jugadores que están en la partida, y a la derecha se ven los puntos que tienen cada uno ordenados de mayor a menor. El nick y los puntos del jugador propio aparecen en negrita. Si se quieren ver la lista de todos los jugadores y no solo los 5 primeros jugadores y el jugador propio, se tiene que pulsar en la flecha hacia abajo que aparece al final de la lista, para que se abra la lista desplegable con todos los jugadores.

Para lanzar una bomba, en el PC se debe pulsar la barra espaciadora, y entonces aparecerá una bomba como la de la imagen anterior, que es un círculo negro en el cual en su interior tiene el cronómetro con los segundos que queda para que explote. Si explota porque ese cronómetro ha llegado a cero, o porque ha chocado con algún elemento del juego, aparecerá el radio de la explosión tal y como se puede ver aquí:

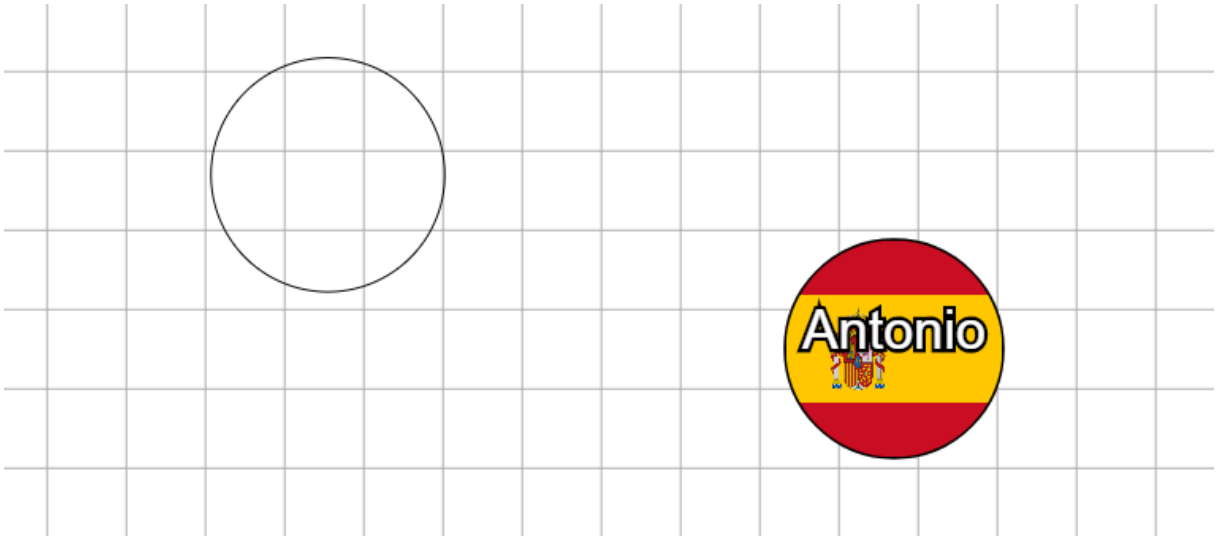


Figura B.4 Explosión de bomba

Una vez que se ha lanzado una bomba, no se puede lanzar otra hasta que ha transcurrido un tiempo.
Si en algún momento el jugador muere porque se lo ha comido otro jugador o porque le ha explotado una bomba, aparecerá lo siguiente:

Figura B.5 Muerte de un jugador

Aparece un mensaje en el que avisa de que has muerto (“You Died!”), y abajo viene un botón que pone “Continue?” que pregunta si quieres volver a entrar a la partida después de haber muerto. Al pulsar sobre ese botón el jugador resucitará en la partida con su tamaño inicial y sus puntos puestos a cero. Como se puede ver en la imagen anterior, el jugador murió porque otro jugador de nombre “Belgium” se lo comió al ponerse encima de él. Observar que este jugador ha crecido de tamaño porque ha aumentado su tamaño en el tamaño del jugador que se ha comido.

Otra funcionalidad que permite el videojuego es el chat de notas de audio. Para chatear, debe pulsarse el botón que pone “Push to talk” en la parte inferior de la pantalla, y al hacerlo aparecerá lo siguiente:

Figura B.6 Grabación de audio

Aparece el mensaje “Recording”, unos puntos suspensivos y a continuación los segundos que quedan para que el videojuego deje de grabar la voz del jugador. Cuando termina la grabación los demás jugadores escucharán el audio de ese jugador, y verán algo como lo siguiente:

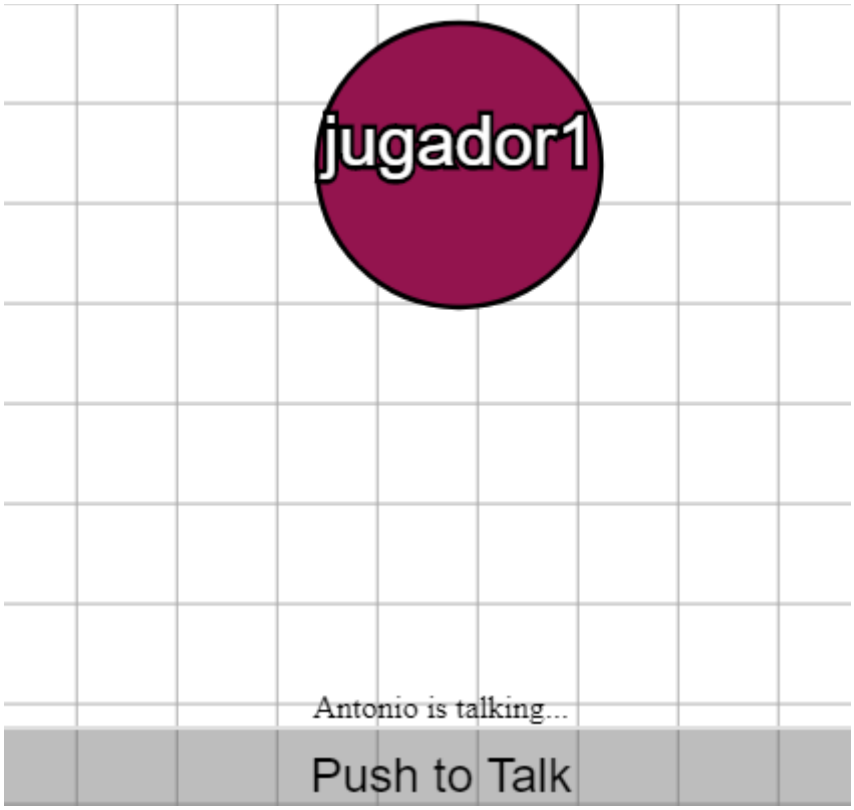


Figura B.7 Recepción de audio por otro jugador

Como se puede ver, aparece el nombre del jugador que está hablando en ese momento, en la imagen aparece “Antonio is talking”porque está hablando el jugador de nick “Antonio”.

En cualquier momento durante la partida, se puede ver en la parte superior izquierda un cronómetro con el tiempo que queda para que termine la partida. Cuando ese cronómetro llega a cero, aparecerán los resultados como se puede ver en la siguiente imagen:

Figura B.8 Pantalla de resultados de la partida

Los resultados que se pueden ver están clasificados en 3 columnas. La primera columna "Player Results" son los resultados del jugador en la partida, clasificada en puntos "Points", bolas que se ha comido "Balls", jugadores que se ha comido "Players", jugadores que ha explotado con bombas "Exploded" y nº de veces que ha muerto "Deaths". La segunda columna llamada "Top Round Results" contiene quién ha conseguido la máxima puntuación durante la partida en cada una de las categorías. Por ejemplo, en la imagen se puede ver que quien consiguió más puntos en la partida fue "Finland" con 300 puntos. La tercera columna llamada "Top Global Results" contiene quién ha conseguido la máxima puntuación en cada una de las categorías sumando los puntos de todas las partidas de cada jugador. Por ejemplo, en la imagen se puede ver que quien se ha comido más bolas durante todas las partidas del juego ha sido el jugador "Uruguay" con 401 bolas comidas.

También se puede ver que en la parte inferior aparece un botón. Pone "Next round in 8 seconds". Significa que quedan 8 segundos para que empiece una nueva partida. Cuando el cronómetro llega a cero, comienza una nueva partida, y se muestra lo siguiente:

Figura B.9 Comienzo de una nueva partida

El botón inferior se transforma en un botón que se puede pulsar. Si se pulsa, el jugador puede entrar en la partida que ya ha comenzado.

B.3 Manual de usuario para el dispositivo móvil.

En un dispositivo móvil, una vez instalada la aplicación en el dispositivo, tendríamos algo como lo siguiente:

Figura B.10 Icono de la aplicación móvil

El icono de la aplicación aparece con el nombre de "Balls Project". Si se hace click sobre la aplicación, se inicia y se usa la aplicación de la misma forma que en el PC, salvo lo siguiente:

- Para mover al jugador se debe mover el dispositivo, y el jugador se mueve con la fuerza de la gravedad y en la dirección en que se mueva el dispositivo.
- Si se quiere lanzar una bomba, se debe pulsar el botón de menú del dispositivo móvil.

- Si se quiere salir de la aplicación, se debe pulsar el botón de retroceso del dispositivo móvil, y aparecerá lo siguiente:

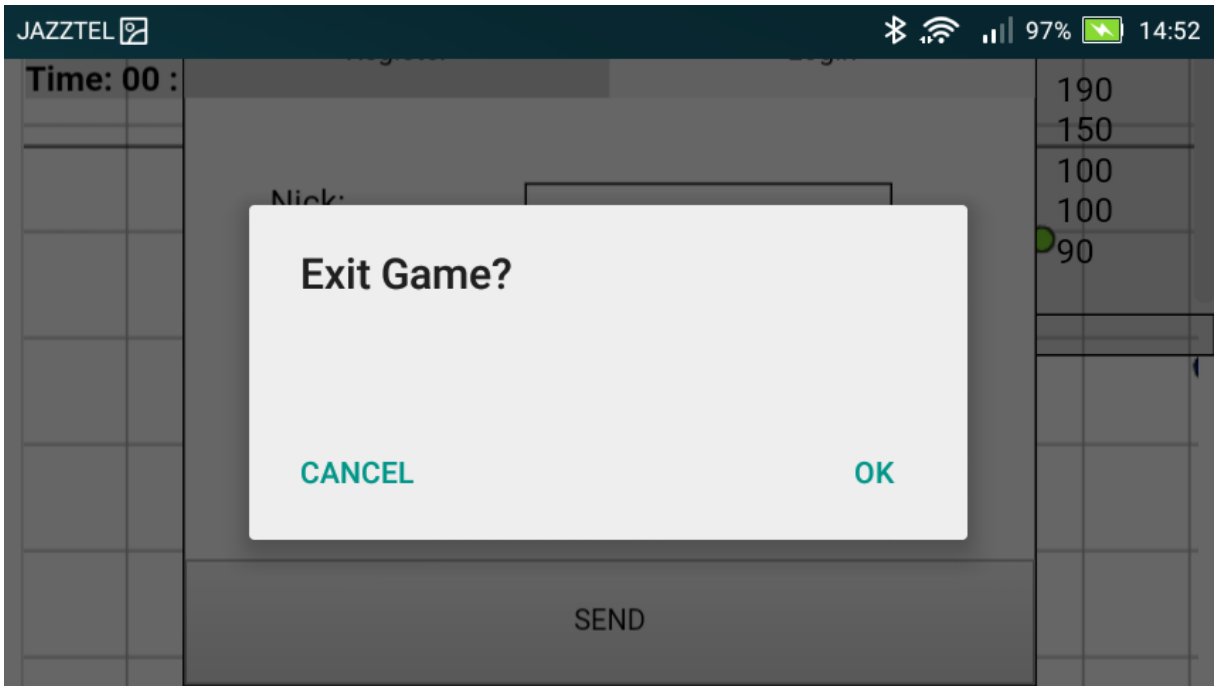


Figura B.11 Salir de la partida en un dispositivo móvil

Si se pulsa OK, se saldrá de la aplicación, y si se pulsa cancel, se continuará en ella.

BIBLIOGRAFÍA

- Tutorial de HTML5 : https://www.w3schools.com/html/
- Tutorial de CSS: https://www.w3schools.com/css/
- Tutorial de JavaScript: https://www.w3schools.com/js/
- Explicación más detallada de JavaScript: https://developer.mozilla.org/es/docs/Web/JavaScript
- Documentación de la API de Canvas: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
- Explicación de cómo capturas audio y vídeo en HTML5: https://www.html5rocks.com/es/tutorials/getusermedia/intro/
- Documentación de Phonegap en su web oficial: http://docs.phonegap.com/
- Documentación de Phonegap en la web de Cordova: https://cordova.apache.org/docs/en/latest/
- Api de Node JS: https://nodejs.org/api/
- Tutorial de Node JS: http://www.tutorialspoint.com/nodejs/
- Explicación de qué son los websockets y su diferencia con la API Rest: https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/
- Explicación de los websockets y sus posibles implementaciones: https://www.html5rocks.com/es/tutorials/websockets/basics/

- Tutorial de Socket.io: https://www.tutorialspoint.com/socket.io/index.htm

- Tutorial de MongoDB: https://www.tutorialspoint.com/mongodb/

- Cómo usar el driver de MongoDB para Node JS: https://docs.mongodb.com/getting-started/node/client/

- Documentación del driver de MongoDB para Node JS: http://mongodb.github.io/node-mongodb-native/2.2/quick-start/quick-start/

- Manual y documentación de MongoDB: https://docs.mongodb.com/manual/

- Web de Mocha con toda su documentación: https://mochajs.org/