UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería Informática. Plan 2010

Desarrollo de un videojuego serio como apoyo al aprendizaje de la biología en el ámbito educativo

Development of a serious video game to support biology learning in education

Realizado por
Joaquín Martín Villa

Tutorizado por
Antonio José Fernández Leiva

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, septiembre 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA. PLAN 2010

Desarrollo de un videojuego serio como apoyo al aprendizaje de la biología en el ámbito educativo

Development of a serious video game to support biology learning in education

Realizado por
Joaquín Martín Villa

Tutorizado por
Antonio José Fernández Leiva

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2025

Fecha defensa: septiembre de 2025

Agradecimientos

Este trabajo está dedicado a todas las personas que me han apoyado y acompañado durante mis años de carrera, ayudándome a no salirme del camino y a conseguir los objetivos que me marqué.

En especial, quiero agradecer a mis padres y abuelos, por no dudar de mí en ningún momento y darme todo el amor posible del mundo.

A mis amigos, con los que he podido desconectarme de la presión que tenía siempre encima. Nos hemos reído, divertido y compartido momentos inolvidables.

A mi pareja, María: te quiero. Gracias por tanto, por tener esa paciencia conmigo cuando este trabajo me superaba y también por todo el esfuerzo que has hecho al diseñar la mayor parte del apartado artístico de este proyecto.

Y, por último, a mi abuela. Siento no haber podido terminar a tiempo para que vieras todo lo que he conseguido. Todo esto va por ti; espero que estés orgullosa de mí, estés donde estés.

Gracias a todos de corazón.

Resumen

El presente Trabajo de Fin de Grado desarrolla un videojuego serio titulado *Brain Battle*, concebido como herramienta educativa para dinamizar el aprendizaje en el aula. El proyecto se ha realizado con el motor *Unity 2022*, empleando C# y el framework *Photon Engine* para la conexión multijugador. El juego combina los géneros *Quiz* y *Party Game*, estructurándose en una aplicación principal para ordenador, que actúa como host y muestra el desarrollo de la partida, y una aplicación secundaria para dispositivos móviles que funciona como mando para los jugadores. Su dinámica se basa en preguntas de opción múltiple con cuatro respuestas posibles, complementadas con minijuegos y habilidades estratégicas que aportan variedad y profundidad, todo esto basado en una temática central como la biología. Además, los usuarios pueden crear y añadir sus propias preguntas, lo que favorece la participación activa y la personalización de las partidas. A continuación, en el presente documento se detallará de manera estructurada todo el proceso de creación de *Brain Battle*, desde la concepción de la idea inicial, pasando por las fases de diseño, desarrollo y pruebas, hasta alcanzar la creación del producto completo.

Palabras clave: Videojuego, Unity, Photon Fusion 2, Biología, Party game.

Abstract

This Final Degree Project presents a serious video game entitled *Brain Battle*, designed as an educational tool to enhance learning in the classroom. The project was developed using the *Unity 2022* engine, programmed in C#, and integrated with the *Photon Engine* framework for multiplayer connectivity. The game blends the *Quiz* and *Party Game* genres, structured into a main computer application that acts as the host and displays the progress of the match, and a secondary mobile application that serves as a controller for the players. Its dynamic revolves around multiple-choice questions with four possible answers, complemented by mini-games and strategic abilities that provide variety and depth, all framed within a central theme of biology. In addition, users can create and add their own questions, fostering active participation and customization of the matches. The following document details the entire process of creating *Brain Battle* in a structured manner, from the conception of the initial idea, through the design, development, and testing phases, up to the completion of the final product.

Keywords: Video game, Unity, Photon Fusion 2, Biology, Party game.

Índice

1 Introducción

En el siguiente documento se presenta todo el trabajo realizado para la creación del videojuego *Brain Battle*, un party game enfocado en preguntas y minijuegos. A lo largo de la memoria se detallan los aspectos más importantes y relevantes de este proyecto, comenzando por la creación de la idea inicial hasta la implementación técnica final y las pruebas correspondientes. En esta primera sección se describe la estructura del documento, la motivación detrás de este trabajo, los objetivos fijados para su desarrollo y, por último, las tecnologías empleadas a lo largo del proceso.

1.1. Estructura del documento

En este documento se presentan las siguientes secciones principales:

- **Sección 1. Introducción**. Se presenta la motivación que impulsó la idea de este proyecto, los objetivos marcados a realizar durante el desarrollo y las tecnologías usadas para la implementación del videojuego.
- **Sección 2. Contexto**. Se explica tanto la idea principal de *Brain Battle*, como el género Party Game. Además, se comentan y analizan referencias a juegos similares o inspiraciones para este mismo.
- **Sección 3. Análisis y Diseño de la Solución**. Se detalla la metodología empleada en el desarrollo, bocetos de diagramas de flujo iniciales, requisitos funcionales y no funcionales. También, se presentan bocetos iniciales del diseño de interfaces.
- **Sección 4. Implementación y Desarrollo**. Se describe la implementación técnica del juego en Unity, comentando la arquitectura general, el diseño del sistema en línea con arquitectura cliente-servidor, la gestión de base de datos, e interfaces de usuario, animación y audio. También se presentan algunos cambios realizados respecto a la idea inicial.

- **Sección 5. Pruebas**. Se presentan las pruebas realizadas, comprobando así el funcionamiento de la aplicación, tanto del flujo de partida como del sistema multijugador, explicando la metodología utilizada, así como sus resultados.
- **Sección 6. Conclusiones y Líneas Futuras**. Se resumirán los objetivos alcanzados y las conclusiones extraídas de todo el trabajo general; además, se propondrán líneas de mejora y formas de continuar este mismo.

1.2. Motivación

El mundo de los videojuegos cobra cada día mayor importancia en nuestra sociedad, ya no solo representa un simple pasatiempo de entretenimiento, sino que también se utiliza como una herramienta pedagógica y de apoyo que ayuda en ámbitos como el aprendizaje o en terapias de rehabilitación motora, cognitiva y de socialización. Desde siempre han sido una forma de disfrute para el autor de este TFG, tanto en solitario como acompañado de amigos, y esa afición fue la motivación que llevó a querer diseñar este proyecto. La pasión por afrontar el reto de crear algo propio, y que no solo sirviera para jugar en un entorno social, sino que también prestase una ayuda al ámbito educativo, impulsó el desarrollo de una propuesta que ofreciera una herramienta diferente y divertida para los alumnos.

El resultado de esta idea se materializa en un juego de preguntas con temática de biología, pensado en el objetivo de reforzar los conocimientos adquiridos en clase de una forma amena y participativa, evitando dinámicas monótonas y repetitivas. Para ello, busqué un funcionamiento sencillo y fácil de aplicar en el aula. Ante el problema del uso de los teléfonos móviles en clase, diseñé la aplicación de forma que los estudiantes no los perciban únicamente como una distracción, sino que comprendan que también puede ser una herramienta útil para el aprendizaje. En ese sentido, la aplicación se basa en un modelo multidispositivo, donde los jugadores se conectan desde sus teléfonos a un host en PC, que actúa como pantalla principal y muestra el desarrollo de la partida.

Todo esto supone un desafío técnico para evaluar las capacidades adquiridas durante el grado, gracias a la implementación necesaria de componentes como la comunicación de red entre cliente y servidor, el diseño de interfaces interactivas e intuitivas de fácil acceso para el usuario, y la adaptación de mecánicas de juegos clásicos de preguntas a un motor de videojuegos.

En resumen, la motivación combina mi pasión por los videojuegos con el reto que supone el diseño de uno desde cero, integrando tanto el componente lúdico como el pedagógico en una misma propuesta.

1.3. Objetivos

Los objetivos definidos para este proyecto fueron los siguientes:

- **Implementar los siguientes sistemas**:
  - **Preguntas**. El eje central de *Brain Battle* son la resolución de preguntas, el jugador podrá definir cuántas cuestiones quiere durante su sesión de partida. Las preguntas estarán formadas por cuatro respuestas, donde solo una será la verdadera.
  - **Puntuación**. Se implementará un sistema de puntuación basado tanto en el número de respuestas correctas obtenidas como en la rapidez de cada jugador al responder, de modo que se premie no solo el conocimiento, sino también la agilidad en la participación.
  - **Minijuegos y Habilidades**. Para aportar dinamismo a las partidas, se diseñarán diferentes minijuegos que interrumpan la dinámica principal de preguntas y ofrezcan retos alternativos, como por ejemplo pruebas de memoria, rapidez o asociación. Las habilidades, por su parte, agregarán un componente de diversión a los jugadores más competitivos, proporcionando efectos tanto negativos como positivos, para ayudar o perjudicar la resolución de las preguntas. Los jugadores podrán decidir si desean jugar con estas modificaciones o no.
  - **Agregación de preguntas**. Se introducirá un apartado para poder agregar preguntas personalizadas a la partida; esto proporciona la capacidad de introducir la aplicación a otros ámbitos y no limitarlo solo al de la biología. Además, se podrá indicar si se desea jugar solo con las preguntas personalizadas o no.
- **Conectividad entre teléfono móvil y PC.** El proyecto se basa en la creación de dos aplicaciones: una app para el dispositivo móvil, a través de la cual los jugadores/as o alumnos/as interactuarán directamente con el juego, y otra aplicación para ordenador, que actuará como pantalla principal durante la partida y que además contendrá las op-

ciones de configuración. Por este motivo, es fundamental asegurar una correcta integración y comunicación en red entre ambos dispositivos.

- **Desarrollo del multijugador.** El proyecto debe soportar partidas multijugador, en las que los jugadores podrán conectarse desde cualquier red mediante un código generado en el host principal. Será necesario garantizar una conexión estable, así como un flujo de partida fluido y adaptable tanto para un único jugador como para varios participantes.
- **Diseño de interfaces de usuario**. Incluir pantallas para el menú inicial, la creación de preguntas, configuración del jugador y la partida, además de las pantallas principales del transcurso de la partida. Se busca que todas ellas sean claras, intuitivas y de fácil comprensión, con el objetivo de facilitar la interacción y la accesibilidad a los jugadores.

1.4. Tecnologías usadas

La creación de *Brain Battle* se realizó principalmente con el motor de videojuegos Unity, utilizado más en concreto Unity 5 con su versión 2022. Este motor es una herramienta muy extendida en el ámbito del desarrollo de videojuegos, tanto a nivel profesional como casual; gracias a esto, Unity tiene una amplia comunidad y una gran oferta de documentación (Unity Technologies, 2022) accesible, que facilita y ayuda al proceso de aprendizaje de su uso. La elección de este motor se basó en diversas razones, como por ejemplo su capacidad para desarrollar aplicaciones en múltiples sistemas operativos (Windows y Android, usados en este proyecto); además, proporciona un entorno de trabajo con herramientas para el diseño de interfaces gráficas, animaciones, gestión de escenas, etc. Toda la programación necesaria para el proyecto se ha llevado a cabo en C#, que es el lenguaje principal admitido por Unity para la implementación de scripts y lógica interna.

A continuación, se listan otras tecnologías usadas para el desarrollo de la aplicación:

- **Editor de código:** Visual Studio Code. Se emplea para escribir y depurar código en C#. Asimismo, VS Code permite añadir extensiones que ayudan al desarrollador, por ejemplo, extensiones específicas para Unity que proporcionan autocompletado de métodos, propiedades y clases predefinidas, de manera que no es necesario memorizar toda la API del motor. (Microsoft Corporation, 2022)
- **Control de versiones:** En este trabajo se han utilizado dos herramientas para este propósito.

Por un lado, *WinMerge*, un programa de software libre utilizado para comparar y combinar archivos. Dado que el trabajo incluye dos aplicaciones distintas (Android y PC) que en ocasiones comparten scripts, *WinMerge* se utiliza para verificar que ambas aplicaciones cuenten con la versión correcta de cada archivo, facilitando la detección de diferencias y asegurando coherencia entre los proyectos. (WinMerge Development Team, 2025)

Por otro lado, también se ha usado Unity Version Control, una herramienta integrada que proporciona Unity en su entorno. Con ello se ha podido llevar el control de las versiones de cada aplicación.

- **Gestión Ágil:** Como metodología de trabajo se escogió utilizar Scrum. Para la planificación y el seguimiento de tareas se ha utilizado Trello, una herramienta basada en tableros y tarjetas que facilita la organización visual del proyecto. (Atlassian, 2025)
- **Assets y recursos:** La mayoría de recursos utilizados en el trabajo han sido creados por la artista del proyecto, María Aron, elaborados con el programa Aseprite. Se trata de un software especializado en la edición y animación de gráficos en pixel art, que ofrece una gran variedad de funciones para el diseño de sprites, tilesets y sus animaciones. (Aseprite, 2023)

  Algunos assets adicionales han sido obtenidos en páginas de recursos gratuitos como itch.io (itch.io, 2023) y Unity Asset Store (Unity Technologies, 2023). En el caso de los audios, se han empleado pistas libres de derechos de autor, obtenidas desde YouTube.
- **Framework de red:** Para implementar la funcionalidad online se ha usado la plataforma Photon Engine, que proporciona varios frameworks para dicha función. En este trabajo se ha utilizado Photon Fusion 2, el cual ofrece una infraestructura optimizada para la sincronización en tiempo real de partidas multijugador, reduciendo la complejidad de implementar un servicio propio desde cero. (Exit Games, 2025a)
- **Editor de texto:** Para la elaboración de esta memoria se ha usado Overleaf, una herramienta online dedicada a la edición de documentos con el lenguaje LaTeX. (Overleaf, 2021)

2
Contexto

En esta sección se van a comentar puntos importantes que contextualizan el juego *Brain Battle*. Primero se explicará en qué consiste el concepto de videojuego serio y se destacarán conceptos clave del desarrollo de videojuegos usados en este trabajo. A continuación, se clasificará este proyecto dentro de un género específico (Party Game, Quiz) y, finalmente, se analizarán juegos que sirven como referencia e inspiración para su desarrollo.

2.1. Juego Serio

Los videojuegos tradicionalmente se han visto solo como una forma de ocio, destinada a proporcionar distracción del mundo real y orientada principalmente a un público joven. Lejos de esta concepción, el papel de los videojuegos ha evolucionado y hoy en día su uso va más allá del simple entretenimiento. El término juego serio fue acuñado por Clark Abt en su libro Serious Games (Abt, 1970), donde definió este concepto para referirse a juegos diseñados con el propósito de enseñar o entrenar, más allá del mero entretenimiento.

En un juego serio, el objetivo principal es que el jugador aprenda algún concepto, la diversión queda en un segundo plano. Esto convierte a los videojuegos serios en herramientas muy útiles en ámbitos como la educación, la salud o en empresas privadas, ya que ofrecen experiencias interactivas que favorecen la motivación por aprender y mejoran la atención sobre los contenidos tratados.

En el contexto de este proyecto, *Brain Battle* se sitúa dentro de esta categoría, al utilizar la dinámica de un videojuego basado en preguntas para reforzar conocimientos en su tema central, la biología. Su objetivo no es únicamente entretener, sino también facilitar el aprendizaje de manera divertida y participativa.

2.2. Conceptos principales de Unity

Unity es uno de los motores de desarrollo de videojuegos más utilizados de la industria (Unity Technologies, 2022). Su uso está muy extendido entre desarrolladores profesionales como amateurs, gracias a que ofrece un entorno de creación completo que permite crear juegos y experiencias tanto en 2D como en 3D. Unity proporciona herramientas integradas para diseñar diversas facetas del juego, como escenas, interfaces, físicas, animaciones y audio. Todo ello con un alto grado de personalización gracias a la programación en C#, un lenguaje orientado a objetos que permite estructurar el código de manera modular, reutilizable y fácil de mantener.

Unity no solo destaca por su flexibilidad y potencia, sino también por su comunidad de usuarios activa, que ofrece documentación, tutoriales y recursos variados que facilitan el aprendizaje y la resolución de problemas. Asimismo, este motor permite desarrollar proyectos para múltiples sistemas operativos y dispositivos.

A continuación, se explicarán algunos de los conceptos más importantes para el desarrollo en el entorno de Unity.

Las **escenas** son contenedores donde se colocan todos los elementos del juego, como personajes, objetos, cámaras, etc. El motor incluye un gestor de escenas que permite crear, organizar y cambiar entre distintas escenas de manera eficiente, facilitando la estructuración del juego en niveles o fases. Estas escenas ayudan a gestionar la carga y descarga de los recursos del juego, optimizando el rendimiento.

Los objetos del juego son definidos como **GameObjects**, que son la unidad básica dentro de Unity. Representan desde un personaje hasta un simple decorado o la propia cámara del juego. Las funcionalidades de los GameObjects se definen mediante componentes, que son módulos que añaden propiedades o comportamientos específicos, como físicas, iluminación, colisiones o scripts personalizados. Son los componentes los que realmente definen a un GameObject. También, un GameObject puede contener otros GameObjects en su interior, estableciendo relaciones de padre e hijo.

En cuanto a la **animación** y el **audio**, Unity ofrece herramientas integradas para controlar el movimiento de los objetos mediante el Animator, así como para gestionar el sonido a través del Audio Mixer, que permite combinar, ajustar y aplicar efectos a las pistas de audio de manera profesional.

Los **assets** son los recursos utilizados en el juego, como imágenes, texturas, sonido y animaciones. El sistema que proporciona Unity permite importar, organizar y reutilizar estos recursos de manera eficiente. Para este proyecto se ha utilizado principalmente sprites 2D, son imágenes bidimensionales, siendo el recurso más utilizado en el estilo gráfico en dos dimensiones, como es el caso de este trabajo.

La codificación en Unity es el proceso mediante el cual se añaden comportamientos y lógica al juego. En este motor se utiliza principalmente C#, permitiendo controlar interacciones, mecánicas del juego, respuestas a eventos y comunicación entre distintos componentes. Normalmente, los scripts heredan de **MonoBehaviour**, la clase base de Unity, de él se utilizan dos métodos muy importantes que son Start(), método que se ejecuta al comenzar una escena, y el Update(), este método es llamado en cada frame por el motor. Además, existen otros métodos que se irán comentando según sea necesario, como otras clases heredadas del sistema base de Unity.

Para finalizar, explicaremos qué es el **Canvas**. El Canvas es un GameObject que actúa como contenedor principal de todos los elementos de interfaz de usuario (Objetos UI). Todos los objetos UI deben estar dentro del Canvas para que se rendericen correctamente en pantalla. El Canvas gestionará cómo se muestran dichos elementos en función de la resolución y el tamaño de la pantalla, permitiendo que la interfaz se adapte a distintos dispositivos.

2.3. Conceptos principales de Photon Fusion 2

Photon Engine es un framework de red para videojuegos que facilita la implementación de funcionalidades multijugador en tiempo real y se integra fácilmente con Unity. Proporciona medios para manejar la comunicación entre clientes y servidores, sincronizar estados de juego, gestionar salas de partidas y transmitir datos de forma eficiente, evitando la necesidad de programar todo el sistema de red desde cero (Exit Games, 2025a).

Dentro de esta plataforma, se encuentra *Photon Fusion 2*, una solución diseñada específicamente para ofrecer redes de alta fidelidad y baja latencia en juegos de este estilo. Fusion ofrece cuatro tipos de topologías diferentes según las necesidades de la aplicación. En este proyecto se ha optado por la topología *Shared-Mode*, en la que todos los jugadores son considerados clientes activos que participan en la simulación, pero existe un jugador que actúa como host principal o servidor, encargado de la autoridad sobre ciertos estados críticos del juego. Los

demás clientes se sincronizan con este host para mantener la coherencia de la partida (Exit Games, 2025b).

Fusion incluye varios conceptos clave que son importantes para entender su funcionamiento:

Network Objects. Son objetos del juego que se registran en la red para que su estado se sincronice automáticamente entre todos los clientes. Cada objeto tiene un propietario que controla su entrada, denominado **InputAuthority**, mientras que el servidor posee autoridad sobre el estado del objeto, llamada **StateAuthority**.

NetworkBehaviour. Normalmente, los scripts en Unity heredan de MonoBehaviour, pero cuando se necesita que un script sea gestionado por la red (por ejemplo, para sincronizar variables como la vida de un personaje o ejecutar métodos compartidos entre clientes), este debe heredar de NetworkBehaviour. Dentro de estos scripts se pueden definir **variables de tipo Network**, que se sincronizan constantemente en la red y solo pueden ser modificadas por el StateAuthority.

Remote Procedure Calls (RPCs). Son métodos que permiten la comunicación entre clientes y servidor. Para que funcionen correctamente, es importante incluir las siglas **Rpc** en el nombre del método. Al definir un RPC, se establece quién puede llamarlo (cualquier cliente, su dueño o el servidor) y quién puede recibirlo.

FusionController. Para implementar Fusion en un juego, se añade el componente FusionController a un GameObject. Este componente incluye varios **callbacks o eventos de red**, que se ejecutan automáticamente cuando ocurren determinados eventos, como la conexión o desconexión de un jugador, o la creación de un objeto en la red.

2.4. Género: Party Game y Quiz

Brain Battle se enmarca dentro de un híbrido entre los géneros **Party Game** y **Quiz**, combinando la dinámica social y competitiva de los juegos para múltiples jugadores con el formato de preguntas y respuestas. Al igual que otros party games, los jugadores interactúan en sesiones cortas y divertidas, fomentando la participación grupal y la competitividad amistosa. Por su parte, la vertiente de quiz introduce un componente educativo y de conocimientos, donde cada jugador debe responder preguntas correctamente para puntuar. Este enfoque permite que el juego sea tanto entretenido como formativo.

2.5. Juegos de referencia

Durante la fase inicial del proyecto se tomaron varios juegos como referencia para el diseño de *Brain Battle*. El nacimiento de este proyecto viene dado gracias, en esencia, a los juegos que comentaremos a continuación.

- **Trivial Pursuit**. El juego de preguntas por excelencia que ha marcado a toda una generación. Publicado originalmente en 1981, se convirtió rápidamente en un referente de los juegos de mesa de tipo quiz. Su popularidad fue tal que no solo se consolidó como un clásico de las reuniones sociales y familiares, sino que también influyó en el desarrollo posterior de numerosos videojuegos de preguntas. (Hasbro, 1981)

Figura 1: Trivial Pursuit(1981)

Trivial Pursuit es un ejemplo de cómo mezclar el reto intelectual con una dinámica accesible y competitiva.

- **¿Quién quiere ser Millonario?**. Este programa de televisión, emitido por primera vez en 1998 en el Reino Unido y posteriormente adaptado en numerosos países, se convirtió en un referente dentro del género de concursos de preguntas y respuestas.

Figura 2: ¿Quién quiere ser Millonario?

El programa popularizó el uso de comodines, que ofrecían a los participantes distintas formas de apoyo, como eliminar respuestas incorrectas o consultar al público. (Eidos Interactive, 2000)

- **Buzz! El Gran Reto**. Lanzado en 2006 para PlayStation 2, este título inauguró una de las sagas de videojuegos de preguntas y respuestas más populares en consolas. El juego recrea un concurso ficticio presentado por el carismático personaje Buzz, ofreciendo partidas dinámicas y accesibles para todo tipo de jugadores. (Relentless Software, 2006)

Figura 3: Buzz! El gran reto

Su principal innovación fue la inclusión de un periférico específico: un mando con cuatro botones de colores destinados a seleccionar respuestas y un botón rojo central utilizado en determinadas pruebas y minijuegos. Este accesorio, diseñado para facilitar la partici-

pación y aumentar la inmersión, supuso un gran atractivo y contribuyó a la popularidad de la saga.

- **Saber es Poder**. Publicado en 2017 para PlayStation 4. Se trata de un concurso de preguntas y respuestas donde los jugadores compiten en distintas rondas por contestar de la forma correcta y más rápida posible. (Wish Studios, 2017)

Figura 4: Saber Es Poder

Este título formó parte de la iniciativa de *PlayLink*, que buscaba fomentar el juego social utilizando dispositivos móviles como mandos. Ha sido la principal inspiración de este proyecto, ya que incluye muchas de las mecánicas que se incorporan, como minijuegos y habilidades.

3

Análisis y Diseño de la Solución

El siguiente capítulo abordará la planificación y el diseño de la aplicación *Brain Battle*. En primer lugar, se describirá el concepto inicial del proyecto. Seguidamente, se presentará la metodología escogida y el modo en que se ha llevado a cabo su desarrollo. También se mostrará la estructura de trabajo definida, incluyendo requisitos funcionales y no funcionales, así como diagramas diseñados. Para finalizar, se presentarán bocetos iniciales sobre las interfaces de usuario, que sirvieron como base para las pantallas finales del juego.

3.1. Primer concepto del juego

La idea inicial del proyecto consistió en el desarrollo de un videojuego de preguntas orientado a su uso en el aula como herramienta de apoyo para los docentes. El objetivo principal era fomentar la participación activa del alumnado mediante una dinámica lúdica, accesible y motivadora.

Desde el inicio se planteó que la aplicación no debía limitarse al uso en ordenadores portátiles, ya que no todos los estudiantes disponen de uno. En su lugar, se decidió implementar un sistema en el que la pantalla principal actuara como host, mientras que los jugadores participarían a través de sus teléfonos móviles, utilizados como controladores. De este modo, se garantiza la accesibilidad y la inclusión de todos los estudiantes, siempre que cuenten con un dispositivo móvil. Esta decisión implicó que el desarrollo debía contemplar dos entornos diferenciados: uno para la pantalla principal y otro para los dispositivos móviles de los participantes.

Con el fin de dotar al proyecto de mayor originalidad frente a otros juegos de preguntas ya existentes, se estableció un sistema de puntuación dual. Este no solo se basaría en el número

de respuestas correctas, sino que también valoraría la rapidez de respuesta, premiando así la agilidad mental de los jugadores.

Asimismo, para aportar variedad a las partidas y mantener la motivación del alumnado, se incluyeron **minijuegos** que actúan como variaciones dentro de la dinámica principal, ofreciendo distintos tipos de interacción y fomentando el pensamiento divergente. De forma complementaria, se diseñó un sistema de **habilidades** que incrementa la competitividad entre los jugadores, permitiendo tanto beneficiarse mutuamente como entorpecer el progreso de otros, siempre dentro de un marco de competitividad sana.

Finalmente, en el ámbito artístico, se decidió que el videojuego contase con un personaje ficticio que desempeña el papel de presentador. Dotando de mayor personalidad a *Brain Battle*.

3.2. Metodología de desarrollo

Existen numerosas metodologías para el desarrollo de software. En este trabajo se optó por utilizar un enfoque ágil. Este tipo de metodologías se caracteriza por su flexibilidad y capacidad de adaptación, lo que permite ajustar la forma de trabajo a las condiciones específicas del proyecto y de su entorno. Resultan especialmente adecuadas en sectores donde la demanda y la competencia evolucionan rápidamente.

El desarrollo ágil se organiza en iteraciones cortas y entregas continuas, lo que facilita la evaluación constante del proyecto e involucra al cliente de manera más activa en su evolución. Estas metodologías se fundamentan en los cuatro valores del Manifiesto Ágil (Beck et al., 2001), de los cuales se derivan posteriormente doce principios. Dichos valores son:

- Individuos e interacciones por encima de procesos y herramientas
- Software funcionando por encima de documentación exhaustiva
- Colaboración con el cliente por encima de negociación contractual
- Respuesta ante el cambio por encima de seguir el plan

Finalmente, dada la naturaleza de este trabajo, se optó por emplear la metodología ágil Scrum, ya que se adapta de manera adecuada al desarrollo iterativo e incremental requerido por el proyecto.

3.2.1. Scrum

Scrum es, probablemente, el marco de trabajo más conocido dentro de las metodologías ágiles, ya que se adapta adecuadamente al desarrollo iterativo e incremental. Se basa en dividir el proyecto en *sprints*, períodos cortos de tiempo, normalmente de dos a cuatro semanas, en los que se deben alcanzar objetivos específicos. Al final de cada sprint, el equipo se reúne para evaluar el progreso y ajustar los planes para el siguiente ciclo.

Este enfoque flexible facilita la entrega continua de resultados, permite observar de forma temprana los requisitos implementados y recibir comentarios del cliente, favoreciendo así una evolución más ajustada a sus necesidades.

Con esto en mente, se decidió elaborar *sprints* con una duración de 2 semanas. Al inicio de cada ciclo se realizaba una planificación de qué tareas, organizadas en el *product backlog*, se iban a desarrollar. El *backlog* hace referencia a una lista priorizada de tareas y funcionalidades pendientes, que sirve como guía del trabajo a realizar. Durante el *sprint*, dado que el proyecto ha sido desarrollado por una persona, se llevaba a cabo un breve seguimiento diario (*daily*), destinado a evaluar el progreso y reajustar la organización personal en caso necesario.

Al finalizar cada *sprint*, se realizaba una revisión en la que se comprobaban las tareas o funcionalidades completadas. Generalmente, esto se materializaba en una nueva versión demo de *Brain Battle*, sobre la cual se recopilaba retroalimentación para orientar el siguiente ciclo de trabajo.

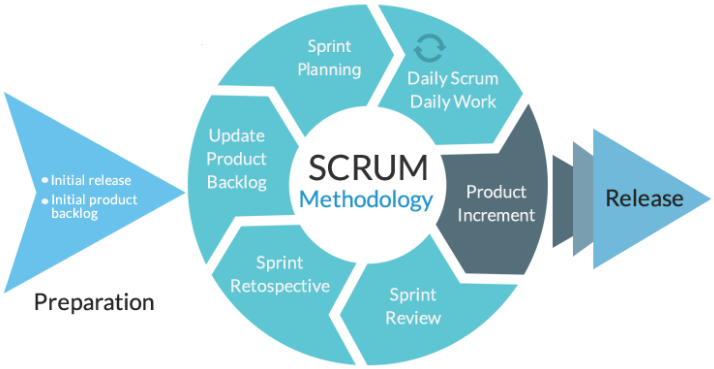


Figura 5: Representación funcionamiento Scrum

En el inicio del desarrollo, se elaboró el *Product Backlog*, que recopila todas las tareas inicialmente detectadas. Sin embargo, este listado no fue algo estático, sino que se mantuvo dinámico a lo largo del proyecto: conforme se avanzaba y aparecían nuevos requisitos o se ajustaban ideas, se incorporaban nuevas historias de usuario y se reordenaban las prioridades. En la tabla siguiente se presentan las historias de usuario definidas.

Historias de usuario	Tareas
HU01: Como sistema, quiero que la conexión entre móvil y PC sea estable e ininterrumpida.	▪ Implementar la arquitectura de comunicación de red. ▪ Programar el manejo de las conexiones de jugadores. ▪ Desarrollar la comunicación entre dispositivo móvil y PC.
HU02: Como sistema, quiero gestionar el flujo de la partida y la sincronización entre la pantalla principal y los móviles, para asegurar que todos los jugadores vean lo mismo.	▪ Crear la lógica de flujo de fases. ▪ Sincronizar los estados entre dispositivos. ▪ Implementar una clase que guarde referencias a los jugadores conectados.
HU03: Como jugador, quiero conectarme a la partida desde mi móvil introduciendo un código, para unirme fácilmente a la sesión en el PC.	▪ Diseñar la interfaz de conexión para el teléfono. ▪ Generar y validar los códigos de sesión. ▪ Conectar el móvil con la partida en el PC.

Historias de usuario	Tareas
HU04: Como jugador, quiero poder responder a las preguntas desde mi móvil.	▪ Diseñar la interfaz de preguntas y respuestas. ▪ Mostrar las preguntas y sus opciones en la pantalla del móvil. ▪ Implementar la selección de respuesta y el envío al host. ▪ Confirmar la recepción de la respuesta. ▪ Asegurar que solo se pueda escoger una única respuesta.
HU05: Como jugador, quiero ver el progreso de la partida en la pantalla principal.	▪ Crear la interfaz gráfica de la partida. ▪ Diseñar la tabla de marcadores para las puntuaciones. ▪ Actualizar la interfaz según la fase (Preguntas, Puntuaciones, Habilidades, Minijuegos). ▪ Indicar al jugador cuál es la respuesta correcta. ▪ Guardar las puntuaciones de los jugadores.

Historias de usuario	Tareas
HU06: Como jugador, quiero poder configurar la partida antes de una sesión.	▪ Implementar la interfaz de configuración de la partida. ▪ Guardar los ajustes elegidos. ▪ Validar las opciones antes de iniciar la partida. ▪ Crear un sistema de dificultad de preguntas. ▪ Habilitar opciones para jugar con habilidades o minijuegos.
HU07: Como jugador, quiero poder personalizar mi avatar o nombre, para diferenciarme del resto de los jugadores.	▪ Diseñar la interfaz de configuración del jugador. ▪ Guardar los datos en el host. ▪ Diseñar el avatar del jugador. ▪ Crear un gestor de skins para el avatar. ▪ Gestionar colores para el avatar. ▪ Validar los datos antes de aceptar la conexión del jugador.

Historias de usuario	Tareas
HU08: Como jugador, quiero jugar a minijuegos durante la partida.	▪ Implementar distintos tipos de minijuegos. ▪ Programar un gestor de minijuegos encargado de seleccionar uno para cada partida.
HU09: Como jugador, quiero tener habilidades que influyan en el transcurso de la partida.	▪ Definir la lista de habilidades. ▪ Implementar interfaz para usar las habilidades y seleccionar su objetivo. ▪ Programar los efectos de las habilidades. ▪ Crear un gestor de habilidades que asegure la consistencia durante la partida.
HU10: Como jugador, quiero ver las habilidades recibidas por otros jugadores.	▪ Mostrar en la interfaz el nombre del jugador y la habilidad que ha recibido. ▪ Verificar que la habilidad se aplique correctamente al jugador afectado.

Historias de usuario	Tareas
HU11: Como jugador, deseo poder añadir mis propias preguntas a la sesión y jugar con ellas.	▪ Implementar la interfaz para la creación de preguntas. ▪ Validar que las preguntas y respuestas sean correctas y coherentes. ▪ Guardar las preguntas en la base de datos. ▪ Integrar las preguntas personalizadas en la partida. ▪ Permitir configurar la partida para jugar únicamente con preguntas personalizadas.
HU12: Como jugador, quiero poder navegar fácilmente e intuitivamente entre los menús del juego.	▪ Desarrollar un sistema de gestión de interfaces. ▪ Implementar un flujo lógico de interacción.
HU13: Como jugador, quiero escuchar música y efectos de sonido durante las sesiones para mejorar la experiencia de juego.	▪ Integrar música de fondo y efectos de sonido en las acciones. ▪ Programar control de volumen y efectos de sonido. ▪ Diseñar un menú para la gestión del audio.

Historias de usuario	Tareas
HU14: Como jugador, quiero que haya un presentador guiando el flujo de la partida.	▪ Diseñar personaje y animaciones. ▪ Crear guion del presentador. ▪ Implementar sistema de diálogos. ▪ Integrar al presentador con las interfaces diseñadas para la partida.
HU15: Como jugador, quiero tener tutoriales que expliquen de manera clara y sencilla el funcionamiento del juego.	▪ Crear cinemáticas de aprendizaje. ▪ Implementar sistema de reproducción de tutoriales. ▪ Integrar opción de usar o no tutoriales.

Cuadro 1: Historias de usuario y sus tareas desglosadas

3.2.2. Sprints y cronograma

En este apartado se comentará el cronograma seguido, así como los hitos principales de cada *sprint*. Cada ciclo ha tenido una duración de aproximadamente dos semanas. Destacar también que, antes de iniciar con el cronograma, ha habido una fase de aprendizaje personal, en la que se investigaron soluciones y diseños para implementar las funcionalidades principales de los objetivos.

A continuación, se presenta el desglose de los distintos sprints llevados a cabo durante el desarrollo del proyecto, indicando los objetivos de cada sprint, las tareas principales realizadas y los resultados obtenidos.

Sprint	Objetivo	Tareas	Resultado
Sprint 0. Planificación inicial y configuración del entorno de trabajo	Definir la estructura inicial del proyecto	▪ Escribir documentos necesarios para el desarrollo, tales como **High Concept** o **Game Design Document**. ▪ Investigar y escoger el motor de juego. ▪ Escoger framework de red. ▪ Configurar el entorno de desarrollo. ▪ Implementar control de versiones. ▪ Crear bocetos de interfaces. ▪ Diseñar la arquitectura principal del proyecto.	Documentación inicial elaborada y entorno de desarrollo configurado con las tecnologías seleccionadas

Sprint	Objetivo	Tareas	Resultado
Sprint 1. Desarrollo del modelo inicial de conexión y flujo básico de juego	Implementar sistema de red que permita la conexión entre cliente (móvil) y host (PC), y crear un flujo básico de preguntas dentro del juego.	▪ Integrar Photon Fusion 2 para la gestión de la red. ▪ Diseñar una clase que controle las fases del juego. ▪ Crear un gestor de preguntas y su estructura de datos. ▪ Programar el flujo de preguntas. ▪ Elaborar la interfaz mínima para mostrar cuestiones en el móvil y en el host. ▪ Probar la conexión y sincronización entre dispositivos. ▪ Programar el sistema de selección de respuestas para el cliente.	Primer prototipo del juego funcional con conexión de red y flujo básico de preguntas implementado.

Sprint	Objetivo	Tareas	Resultado
Sprint 2. Implementación de puntuaciones, minijuegos y flujo final de partida.	Desarrollar el sistema de puntuaciones y su correspondiente tabla de posiciones, desarrollar la lógica de final de partida y definir e integrar los minijuegos dentro del flujo del juego.	▪ Elaborar la interfaz para la tabla de puntuaciones. ▪ Diseñar el sistema de puntuaciones. ▪ Integrar el sistema de puntuaciones con el flujo general de la partida. ▪ Implementar un flujo para el final de la partida. ▪ Diseñar interfaces para los minijuegos. ▪ Programar los minijuegos y su estructura de datos. ▪ Crear un gestor de minijuegos. ▪ Integrar los minijuegos con el flujo general de la partida.	Sistema de puntuación y minijuegos integrado, flujo final de la partida definido.

Sprint	Objetivo	Tareas	Resultado
Sprint 3. Implementación de habilidades y preguntas locales.	Desarrollar e integrar el sistema de habilidades junto con la agregación de preguntas personalizadas por el jugador.	▪ Diseñar las interfaces necesarias para las habilidades. ▪ Desarrollar el gestor de habilidades encargado de proporcionar habilidades al jugador y recibirlas. ▪ Integrar el gestor de habilidades con el flujo de la partida. ▪ Comprobar la recepción de habilidades. ▪ Programar las habilidades. ▪ Diseñar la interfaz del sistema de agregación de preguntas locales. ▪ Programar la lógica de creación y eliminación de preguntas personalizadas. ▪ Programar la lógica de gestión de preguntas personalizadas.	Sistema de habilidades funcional y agregación de preguntas personalizadas incluidas en el producto.

Sprint	Objetivo	Tareas	Resultado
Sprint 4. Mejora de puesta en escena visual y auditiva.	Integrar la presentadora, avatares para el jugador, tutoriales y sistema de audio.	▪ Integrar el diseño de la presentadora con las interfaces. ▪ Programar sistema de diálogo. ▪ Animar a la presentadora. ▪ Implementar a la presentadora en el flujo de la partida. ▪ Crear cinemáticas explicativas del juego (tutoriales). ▪ Implementar tutoriales en el flujo del juego. ▪ Desarrollar el sistema de audio. ▪ Diseñar el sistema de elección de avatares.	Versión completa del juego, con presentadora, tutoriales y mejoras de audio implementadas.

Sprint	Objetivo	Tareas	Resultado
Sprint 5. Desarrollo de memoria, pruebas y pulido.	Pulir el producto final y elaborar la memoria de desarrollo del proyecto.	▪ Mejorar detalles menores de la interfaz de usuario. ▪ Probar la fluidez de los menús con usuarios reales. ▪ Probar la escalabilidad del proyecto. ▪ Corrección de errores surgidos. ▪ Elaborar la memoria de desarrollo. ▪ Preparar versiones de las aplicaciones para la entrega final.	Producto final completo y documentación de entrega completada.

Cuadro 2: Sprints de desarrollo

Es importante señalar que, aunque se estableció una planificación inicial, ésta se mantuvo lo suficientemente flexible como para ajustarse a cambios e imprevistos durante el desarrollo. Dichos ajustes forman parte natural del enfoque iterativo propio de las metodologías ágiles. En este caso, la aplicación de Scrum resultó efectiva, ya que permitió disponer, en una fase temprana, de un prototipo jugable básico (conexión cliente-host y flujo de preguntas), sobre el cual se fueron incorporando progresivamente otras funcionalidades clave, como minijuegos, habilidades y presentadora.

3.2.3. Diagrama de Gantt

El siguiente diagrama de Gantt presenta la planificación temporal del desarrollo de *Brain Battle*, estructurada en tres bloques principales: **Diseño**, **Desarrollo** y **Pruebas**. Cada bloque

se compone de un conjunto de tareas distribuidas a lo largo de los *sprints* anteriormente definidos, especificando tanto la duración de las actividades como los entregables obtenidos en cada una de ellas.

La planificación refleja la aplicación de una metodología ágil, en la que las fases no se desarrollan de forma secuencial, sino que se superponen e iteran de manera continua. Este enfoque permite incorporar retroalimentación en cada sprint y realizar mejoras incrementales en el producto. Además, se han establecido una serie de hitos clave que señalan los entregables más relevantes y marcan la evolución del proyecto en cada etapa.

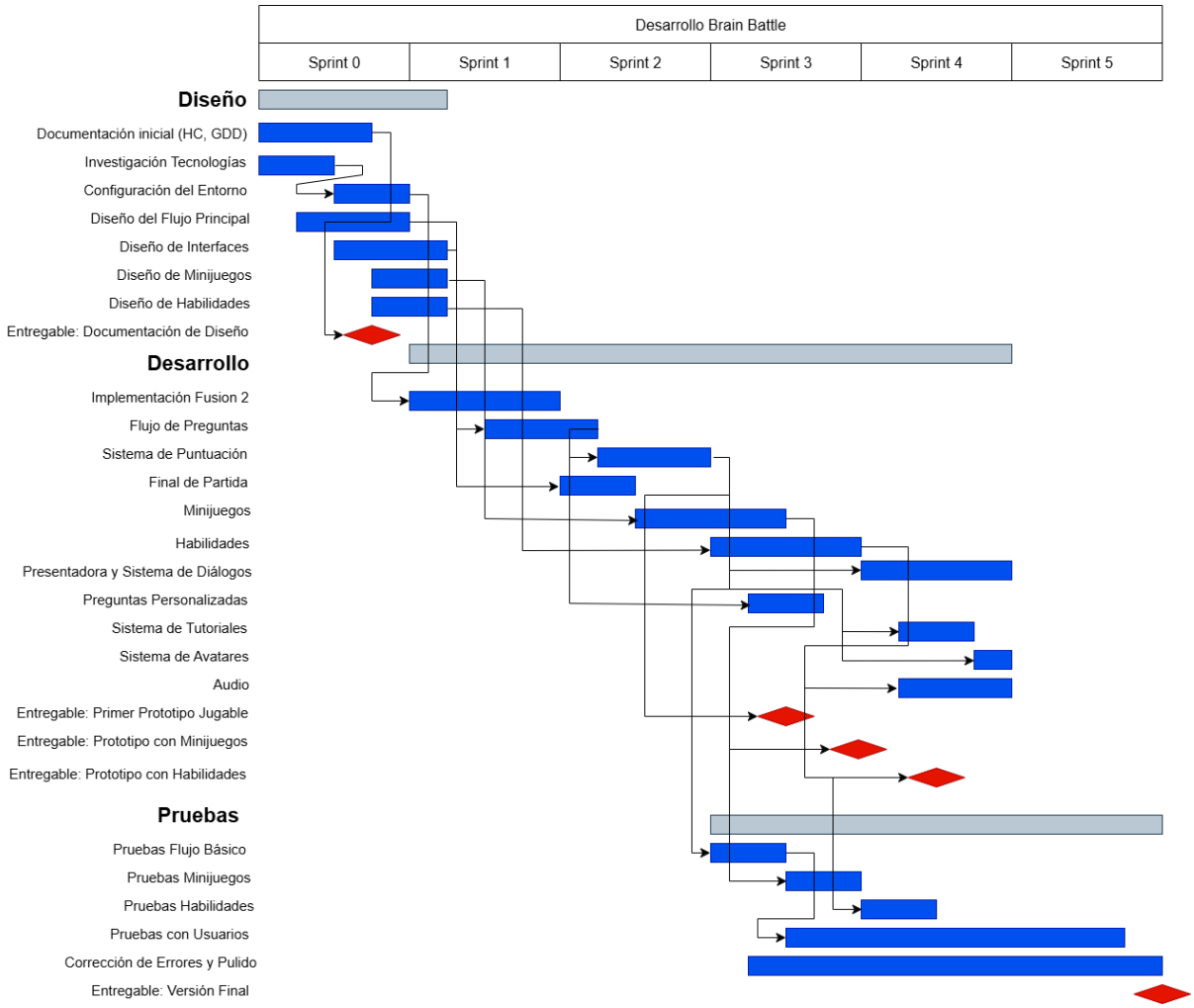


Figura 6: Diagrama de Gantt

3.3. Requisitos

Los requisitos de un software son las especificaciones que definen qué debe hacer un sistema y bajo qué condiciones debe operar. Se dividen en dos tipos principales: los **requisitos funcionales**, que describen las funciones y comportamientos que el software debe cumplir, como procesos, entradas y salidas; y los **requisitos no funcionales**, que establecen restricciones o características de calidad del sistema, como rendimiento, seguridad, usabilidad o compatibilidad.

A continuación, se van a desglosar los requisitos definidos para este proyecto.

3.4. Requisitos Funcionales

Se presentan primero los requisitos funcionales, que describen las funciones y comportamientos que el sistema debe ejecutar durante la partida, estableciendo las bases para el correcto desarrollo y gestión del juego.

- **RF1.1: Creación y gestión de partidas online.** El sistema host debe ser el único con la capacidad de crear y gestionar las conexiones de red de la partida.
- **RF1.2: Flujo de la partida.** El sistema host debe controlar el desarrollo de la partida, determinando en qué momento se muestran las preguntas, se resuelven y cuándo se introducen los minijuegos o las habilidades.
- **RF1.3: Tablas de puntuación.** El sistema debe mostrar la puntuación obtenida por los jugadores en cada pregunta o minijuego, así como la puntuación total acumulada de cada jugador al finalizar cada ronda.
- **RF1.4: Pregunta y respuestas.** El sistema debe mostrar la pregunta correspondiente junto con sus cuatro opciones de respuesta de forma simultánea, tanto en el dispositivo host como en los dispositivos móviles de los jugadores.
- **RF1.5: Respuesta correcta.** El sistema debe indicar y mostrar, una vez finalizado el tiempo de respuesta, cuál era la opción válida.
- **RF1.6: Guardar respuestas de los jugadores.** El sistema debe ser capaz de guardar todas las respuestas escogidas por los jugadores y calcular el tiempo tardado en ello.

- **RF1.7: Tiempo de resolución de cada ronda.** El sistema debe mostrar en todo momento el tiempo restante para responder la pregunta o completar el minijuego correspondiente. Además, debe avisar al jugador, de forma sonora, cuando quede poco tiempo para finalizar.

- **RF1.8: Opciones al final de la partida.** El sistema debe ofrecer la posibilidad de repetir la partida con la misma configuración y jugadores, o finalizarla por completo desconectando a los usuarios.

- **RF1.9: Pérdida de conexión.** El sistema debe notificar al jugador cuando pierda la conexión con el host o sea desconectado de la partida.

- **RF1.10: Manejo de audio.** El sistema debe ofrecer la posibilidad de controlar el volumen tanto de la música de fondo como de los efectos sonoros.

- **RF1.11: Explicación de tutoriales.** El sistema debe ofrecer una serie de tutoriales que expliquen de manera concisa y clara su funcionamiento.

Seguidamente, se muestran los requisitos funcionales que describen las acciones que el jugador debe poder realizar durante la partida, definiendo cómo interactúa con el sistema y asegurando una experiencia de juego coherente y controlada.

- **RF2.1: Conexión mediante código**. Los jugadores deben poder unirse a la partida mediante un código proporcionado por el sistema host.

- **RF2.2: Elección de respuesta**. El jugador debe poder seleccionar únicamente una de las cuatro opciones ofrecidas, y el sistema debe mostrar de forma clara cuál ha sido la respuesta elegida.

- **RF2.3: Menú e interacción**. El jugador debe poder navegar entre los menús definidos de una manera clara y fluida.

- **RF2.4: Nombre y avatar personalizados**. El jugador debe poder escoger un nombre propio que le represente durante la partida, al igual que un avatar personalizado, pudiendo escoger el color y la skin de este.

- **RF2.5: Dificultad de las preguntas**. El jugador debe poder escoger la dificultad de las preguntas antes del comienzo de la partida.

- **RF2.6: Probabilidad de aparición de preguntas de dificultad inferior**. El jugador debe poder escoger si quiere que aparezcan preguntas de un nivel inferior al escogido; si es así, también debe tener la posibilidad de definir el porcentaje de aparición.

- **RF2.7: Número de preguntas**. El jugador debe poder escoger el número de preguntas que desee tener en su sesión de juego antes del comienzo de la partida.

- **RF2.8: Minijuego y habilidades durante la sesión de juego**. El jugador debe poder escoger si desea jugar con minijuegos o habilidades durante la partida.

- **RF2.9: Tutoriales durante la sesión de juego**. El jugador debe tener la elección de usar o no los tutoriales durante la partida.

- **RF2.10: Preguntas personalizadas**. El jugador debe tener la posibilidad de agregar sus propias preguntas a la base de datos y elegir si desea jugar únicamente con ellas.

- **RF2.11: Eliminar preguntas personalizadas**. El jugador debe tener la opción de eliminar preguntas personalizadas agregadas a la base de datos.

Por último, se mostrarán los requisitos definidos para dos bloques importantes dentro de la sesión de juego, minijuegos y habilidades.

- **RF3.1: Elegir tipo de minijuego**. El sistema debe ser capaz de escoger y cargar un tipo de minijuego, sin repetir el mismo más de dos veces por partida. Solo se repetirá si todos los minijuegos ya han aparecido en la sesión.

- **RF3.2: Mover ficha del minijuego**. El jugador debe poder mover la ficha usada en algunos minijuegos al panel correspondiente, y debe tener una referencia visual de si ha acertado o ha fallado.

- **RF3.3: Repartir habilidades**. El sistema debe repartir habilidades aleatorias para cada jugador.

- **RF3.4: Jugar una habilidad**. El jugador debe poder ver sus habilidades en el inventario y escoger una de ellas solo pulsando en la pantalla.
- **RF3.5: Elección de objetivo**. El jugador debe poder escoger el objetivo al que lanzar la habilidad simplemente pulsando la pantalla de su dispositivo.

3.5. Requisitos No Funcionales

- **RNF1: Rendimiento general**. El juego debe ejecutarse de forma estable y fluida en dispositivos de gama media, manteniendo aproximadamente 60 FPS, y debe ser capaz de funcionar de manera continua durante sesiones de hasta una hora sin interrupciones ni cierres inesperados.
- **RNF2: Desconexión de jugadores**. El sistema debe manejar las desconexiones de los jugadores de manera que no se interrumpa la sesión de juego para los demás usuarios.
- **RNF3: Seguridad**. Las conexiones entre host y clientes deben ser seguras para evitar cualquier tipo de ataque o intrusión.
- **RNF4: Fiabilidad**. El juego debe ser estable, sin presentar fallos críticos ni interrupciones del sistema. Además, debe garantizar que todos los clientes reciban y sincronicen correctamente la misma información proporcionada por el host.
- **RNF5: Usabilidad**. La interfaz debe ser intuitiva y comprensible, de forma que un nuevo jugador pueda entender cómo jugar en menos de 5 minutos.
- **RNF6: Integración de elementos de interfaz**. Los elementos de la interfaz de usuario deben tener un tamaño y disposición adecuados, de manera que el jugador pueda visualizarlos claramente y utilizarlos de forma cómoda.
- **RNF7: Retroalimentación de la interfaz**. El sistema debe proporcionar un feedback visual y/o sonoro inmediato cuando el jugador interactúe con la interfaz, ya sea al pulsar botones, seleccionar opciones o arrastrar elementos en los minijuegos, garantizando que las acciones realizadas queden claramente reflejadas.
- **RNF8: Minijuegos modulares**. El sistema debe integrar los minijuegos de forma modular, de manera que sea posible agregar nuevos minijuegos al proyecto en el futuro.

- **RNF9: Habilidades modulares**. De manera similar, el sistema debe integrar las habilidades de forma modular, permitiendo la incorporación de nuevas habilidades al proyecto cuando sea necesario.

3.6. Diagramas de flujo

Los diagramas de flujo son representaciones gráficas que muestran, de manera secuencial, las etapas, decisiones y procesos que conforman un sistema o procedimiento. Su propósito principal es visualizar de forma clara y ordenada el flujo de información o de acciones, facilitando la comprensión del funcionamiento del sistema, la identificación de posibles errores y la planificación de su desarrollo o mejora.

En el ámbito del desarrollo de software, los diagramas de flujo permiten a diseñadores y programadores entender la lógica de los procesos antes de implementar el código.

A continuación, se presentan los diagramas de flujo diseñados inicialmente para *Brain Battle*, los cuales ilustran de manera gráfica el funcionamiento de los distintos menús previstos y el flujo principal de la partida.

3.6.1. Flujo de una partida

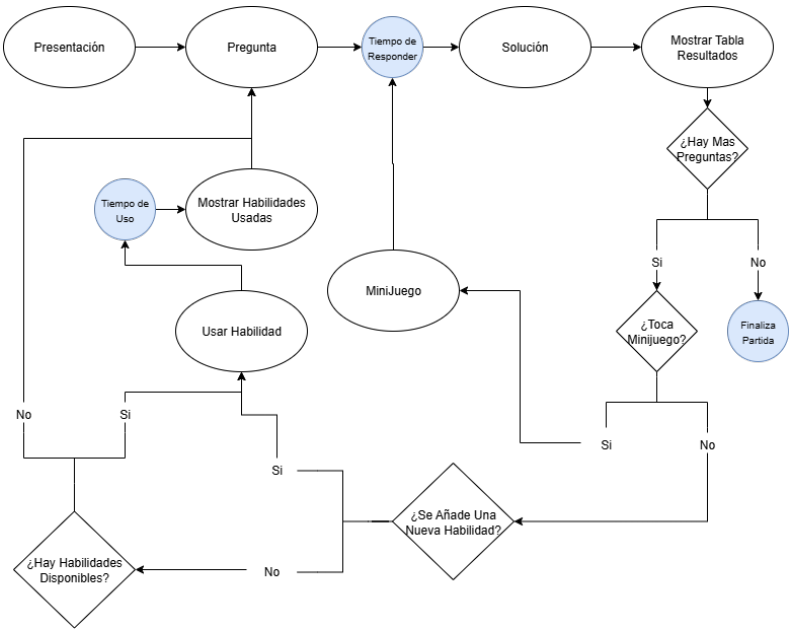


Figura 7: Flujo principal de una partida

3.6.2. Lógica de las interfaces del host

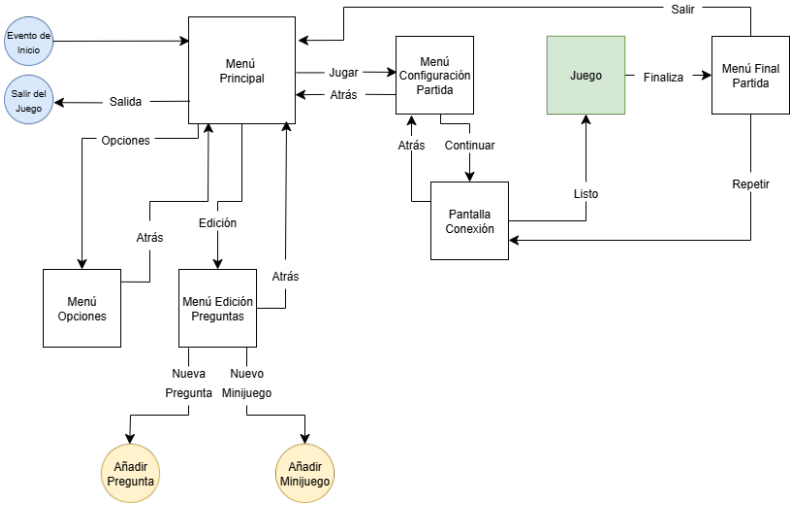


Figura 8: Flujo de interfaces del host

3.6.3. Lógica de las interfaces del dispositivo móvil

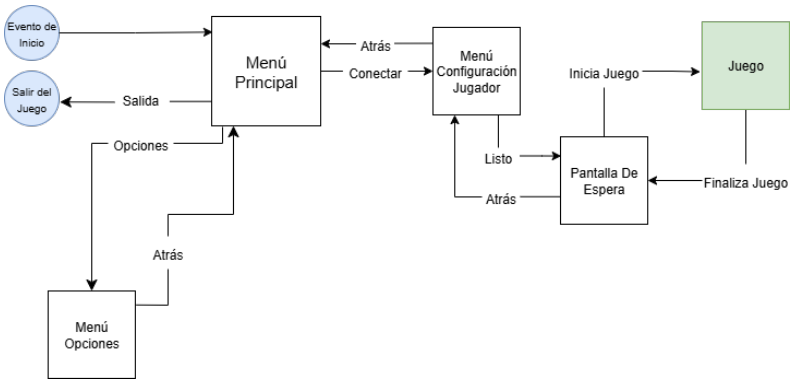


Figura 9: Flujo de interfaces del dispostivo móvil

3.7. Bocetos iniciales de las interfaces

En esta fase de diseño se realizaron dos tipos de bocetos. En primer lugar, se elaboraron dibujos en papel con el objetivo de explorar de forma rápida distintas disposiciones y elementos clave de la interfaz. Posteriormente, estos bocetos sirvieron para ser trasladados a Unity, donde se desarrolló un prototipo inicial para confirmar la disposición deseada. Este proceso permitió no solo visualizar las pantallas del juego, sino también asegurar una buena usabilidad y detectar posibles problemas de uso.

3.7.1. Bocetos en papel

.

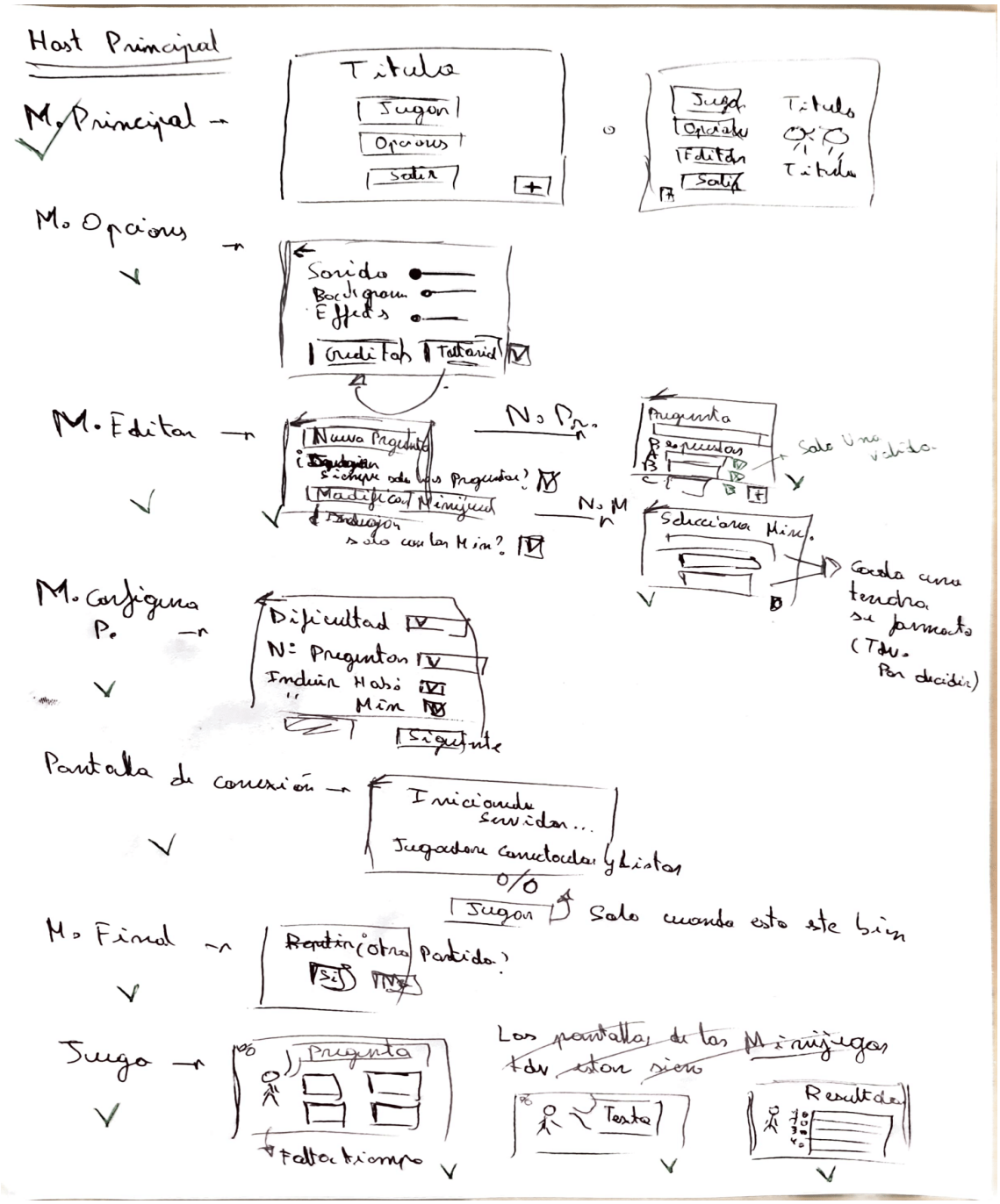


Figura 10: Bocetos de interfaces para el host

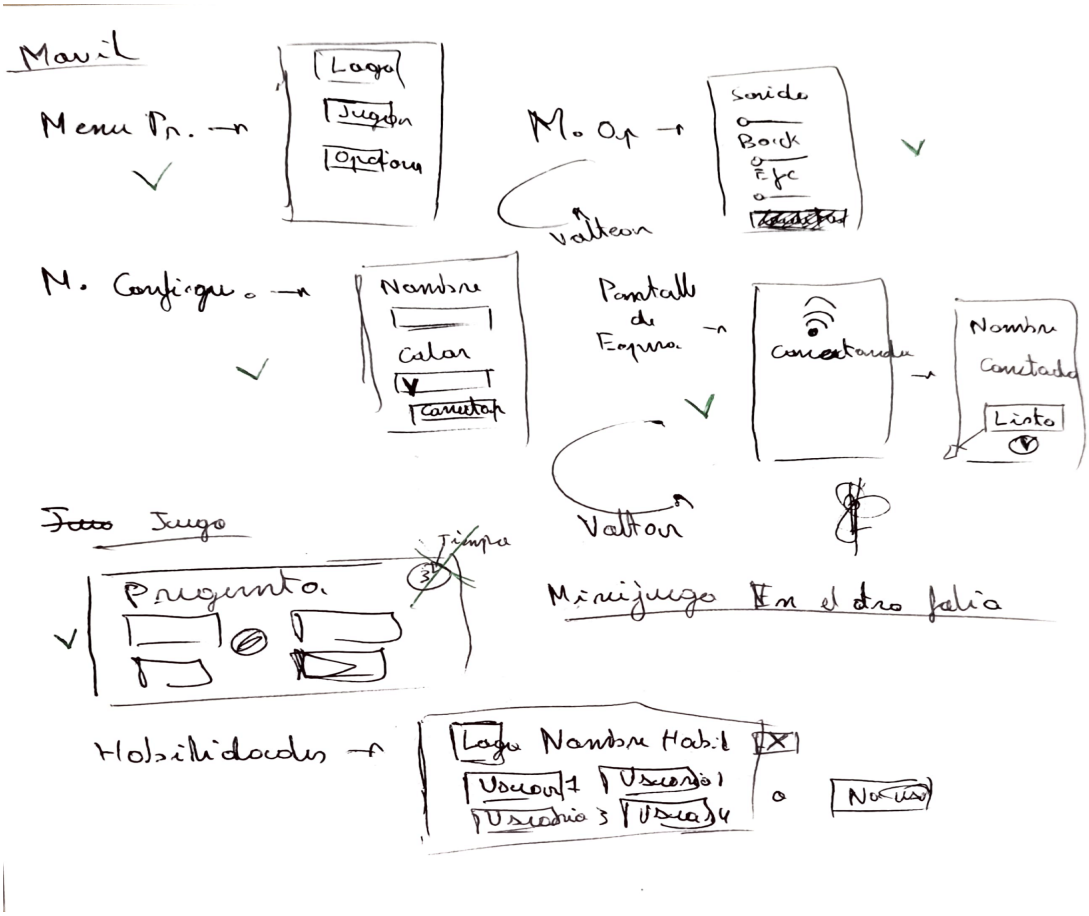


Figura 11: Bocetos de interfaces para el móvil

3.7.2. Menú Inicial

Para el menú inicial del host se ha diseñado una disposición sencilla e intuitiva. El logotipo original de *Brain Battle* se ubicará en la parte izquierda de la pantalla, mientras que a su derecha se presentarán las opciones principales mediante botones de gran tamaño, con el fin de facilitar la navegación del jugador. Las opciones disponibles serán las siguientes:

- **Jugar:** inicia la partida y establece la conexión en red necesaria.
- **Opciones:** abre el menú de configuración del juego.
- **Salir:** cierra la aplicación.
- **+ :** botón de menor tamaño que permite acceder a la interfaz de edición de la base de datos personalizada de preguntas.

Figura 12: Menú principal del host

El menú inicial para los dispositivos móviles sigue un patrón similar al del host, pero en este caso solo muestra dos opciones de botones:

- **Jugar:** inicia la partida y establece la conexión en red necesaria.
- **Opciones:** abre el menú de configuración del juego.

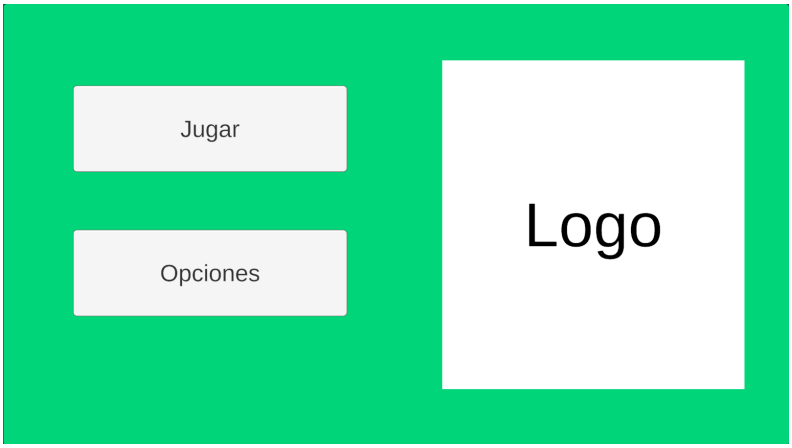


Figura 13: Menú principal del telefono

3.7.3. Menú configuración de partida y jugador

Antes de crear la sala del host, donde los jugadores podrán unirse al servidor, se mostrará un menú de configuración de la partida. En este menú se podrán ajustar las principales opciones del juego, como la **dificultad**, el **número de preguntas**, y la posibilidad de habilitar los **minijuegos** y/o las **habilidades**, permitiendo así una experiencia más personalizada para los jugadores.

Figura 14: Menú configuración partida

A su vez, los jugadores, en sus dispositivos móviles, dispondrán de un menú de configuración personal, en el que deberán introducir su **nombre** así como escoger un **color** personal.

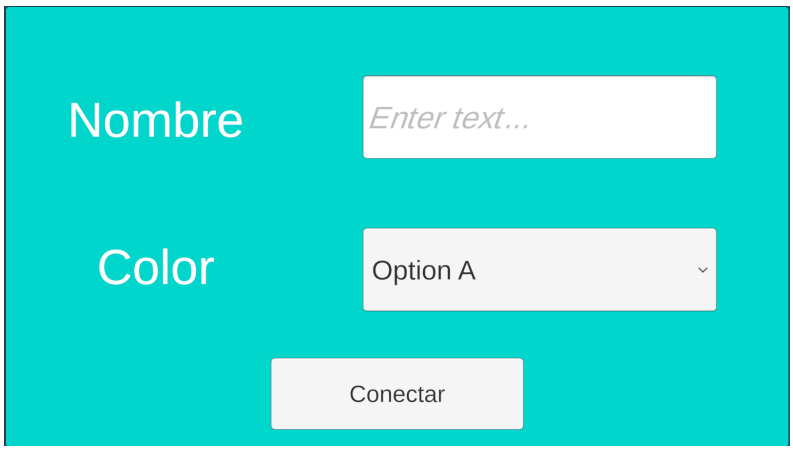


Figura 15: Menú configuración jugador

Más adelante, en el diseño, se decidió incluir otras opciones en estas interfaces, como la posibilidad de que el jugador escoja un **avatar** y la opción de permitir la aparición de preguntas de un nivel inferior durante la partida, pudiendo seleccionar el porcentaje de estas en el menú de configuración de la partida.

3.7.4. Interfaz preguntas personalizadas

Al pulsar el botón + en el menú inicial, se abrirá la interfaz destinada a la gestión de las preguntas personalizadas por el jugador. Inicialmente, también se propuso la idea de agregar

información local a los minijuegos; sin embargo, por falta de tiempo, esta funcionalidad no se implementó.

En este menú se pueden encontrar las siguientes opciones: **agregar una nueva pregunta**, **mostrar todas las preguntas locales agregadas** y, finalmente, la opción de jugar únicamente con las preguntas locales durante la partida.

Figura 16: Menú edición preguntas

La siguiente interfaz muestra el proceso para agregar una nueva pregunta a la base de datos.

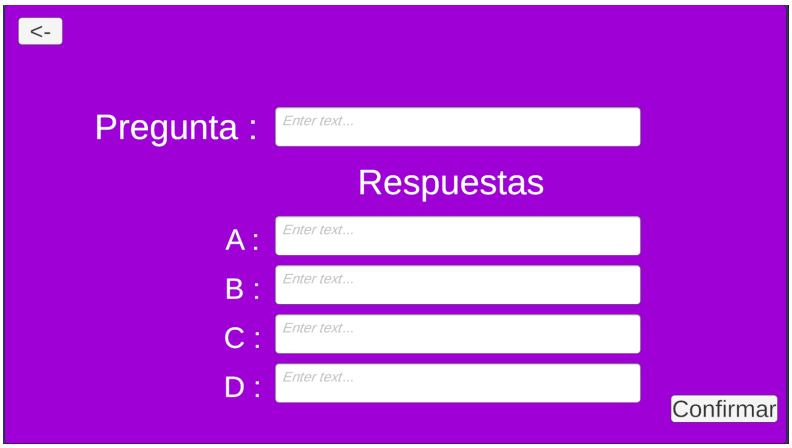


Figura 17: Interfaz agregar pregunta

3.7.5. Interfaz principal del juego

La interfaz principal para el host se ha diseñado reservando un espacio en la parte izquierda de la pantalla para la aparición de la presentadora. Además, se mantiene un área libre

destinada a la colocación de los distintos elementos que pueden aparecer durante la partida, como las preguntas y sus opciones, la tabla de resultados o la explicación de algún concepto. A continuación, se presentan varios bocetos que ilustran este diseño.

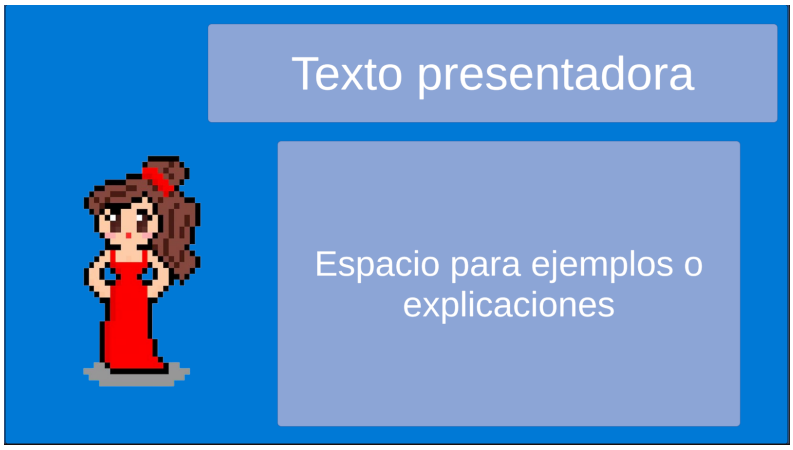


Figura 18: Interfaz principal de una partida

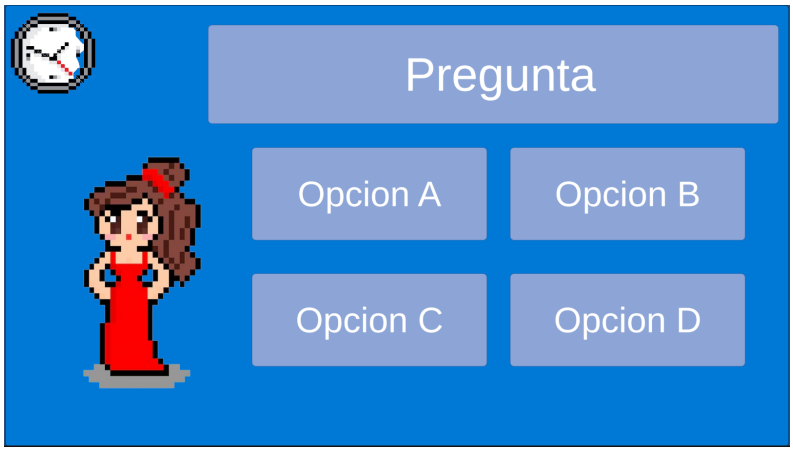


Figura 19: Interfaz principal de una partida con preguntas

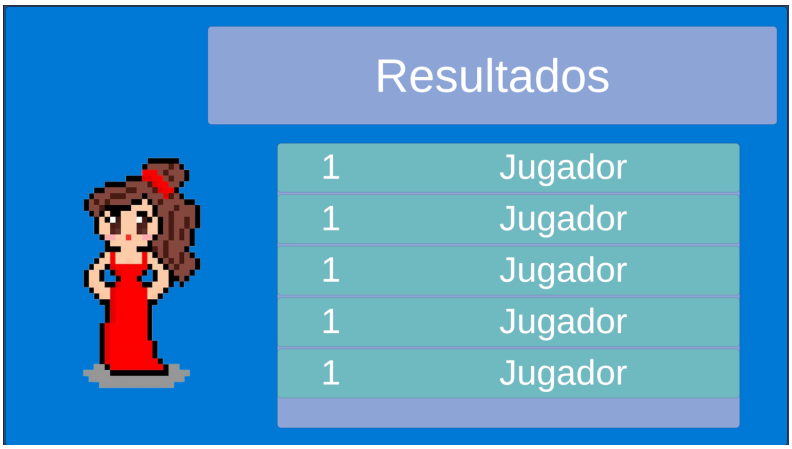


Figura 20: Interfaz principal de una partida con resultados

En cuanto a la interfaz principal para dispositivos móviles, se optó por un diseño simplificado, centrado en facilitar el flujo de preguntas de manera clara y accesible.

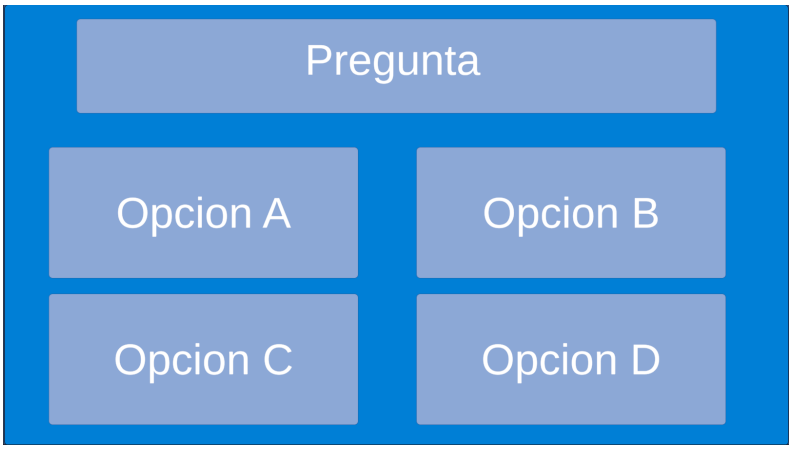


Figura 21: Interfaz principal de una partida para el jugador

3.7.6. Minijuegos

.

Aunque se diseñaron varios minijuegos con sus respectivas interfaces, en esta sección se presentan únicamente aquellos que fueron finalmente implementados en el proyecto. En concreto, se muestran dos minijuegos:

- **Verdadero o Falso**: el jugador arrastra las fichas hacia los paneles de *Verdadero* o *Falso*, según corresponda al enunciado de la pregunta.

- **Tema Correcto**: el jugador arrastra cada ficha al panel que considere correspondiente al tema al que pertenece.

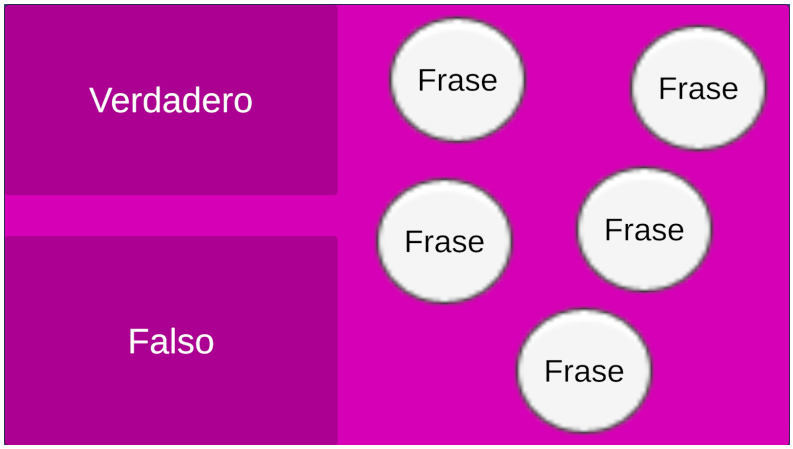


Figura 22: Verdadero o Falso

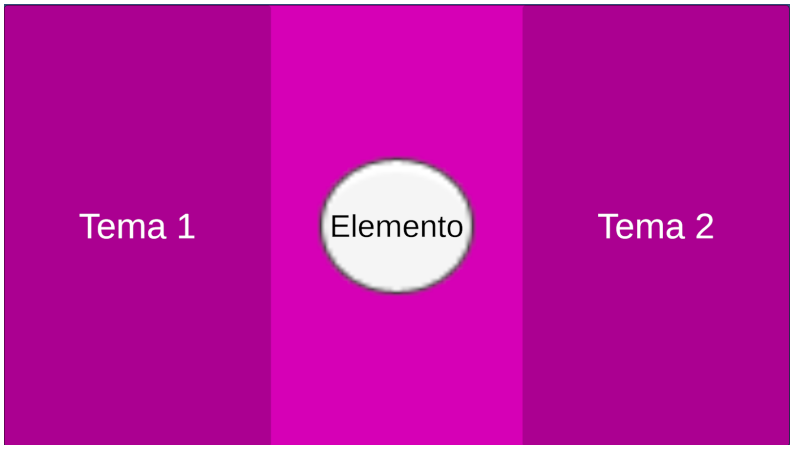


Figura 23: Tema Correcto

Además de las interfaces presentadas, se diseñaron otras pantallas, como la de espera de conexión de jugadores, el menú de opciones y la interfaz de habilidades, entre otras. Sin embargo, para no alargar el desarrollo de la memoria, estas se incluyen de manera detallada en el Apéndice B, correspondiente al *Game Design Document (GDD)*.

3.8. Minijuegos

.

Se incluyeron minijuegos dentro de *Brain Battle* con el objetivo de enriquecer la experiencia de juego, ofreciendo variedad y dinamismo durante las partidas. Estos minijuegos permiten reforzar los conocimientos del jugador de manera interactiva y lúdica, al mismo tiempo que fomentan la participación activa y aumentan la diversión, haciendo que la experiencia global sea más atractiva y motivadora.

Todos los minijuegos siguen una misma estructura: presentan diversas preguntas de forma dinámica, rápida e individual. Estos minijuegos se reproducen únicamente en los dispositivos móviles, sin que aparezcan en el host principal. Cada jugador dispone de un tiempo específico para contestar correctamente el mayor número posible de preguntas. En caso de error, no se aplica penalización; simplemente la pantalla se mostrará en rojo indicando la equivocación. Por el contrario, si la respuesta es correcta, la pantalla se iluminará en verde y, dependiendo del minijuego, aparecerá la siguiente pregunta. Al agotarse el tiempo, se sumará la puntuación correspondiente al número de respuestas correctas obtenidas por cada jugador.

Estos minijuegos se presentan después de un determinado número de preguntas y, antes de comenzar, se explica su mecánica de juego para que todos los participantes comprendan cómo se desarrollará.

A continuación se explican los diversos minijuegos pensados en la fase de diseño:

- **Tema Correcto -** En cada lado de la pantalla se presentan dos temas diferentes, y en el centro aparece una ficha con un elemento relacionado con uno de estos dos temas. El jugador debe asociar la ficha con el tema que considere correcto, arrastrando el elemento hacia uno de los lados de la pantalla, los cuales están diferenciados con un área de color para identificar cada tema y el lugar donde se puede soltar la ficha. Tanto si acierta como si falla, aparecerá una nueva ficha.
- **Verdadero o Falso -** Diversas fichas con frases que pueden ser verdaderas o falsas se presentan en pantalla. El jugador debe indicar cuál cree que es la respuesta correcta arrastrando la ficha al campo correspondiente. Al finalizar el tiempo, se comprueba cuáles han sido acertadas y cuáles no. Durante el minijuego, las fichas pueden ser cambiadas de bando tantas veces como el jugador considere. El sistema mantiene un equilibrio en-

tre frases verdaderas y falsas para evitar que predominen únicamente un tipo.

- **Conecta Elementos -** La pantalla se divide en dos partes: en un lado se muestran varias fichas que representan distintos elementos, y en el otro lado aparece una única ficha relacionada con solo una de las anteriores. El jugador debe relacionar esta ficha con la correspondiente, tocando una ficha de un lado y arrastrando el dedo para generar una línea que conecte con una ficha del otro lado. Si ambas fichas están relacionadas, desaparecen y son reemplazadas por otras; en caso de error, la línea se borra y el jugador debe intentarlo nuevamente.
- **Completa la frase -** Como su nombre indica, se presentan frases incompletas en pantalla que el jugador debe completar utilizando las fichas que aparecen en la parte inferior del dispositivo móvil. Para ello, arrastra la ficha que considere correcta hacia la parte superior de la pantalla para completar la frase.

En el producto final presentado con el proyecto, solo se terminaron de incluir los minijuegos **Verdadero o Falso** y **Tema Correcto**, pero se dejaron las bases para poder incluir los demás minijuegos si así se deseara.

3.9. Habilidades

Se incorporaron habilidades dentro de *Brain Battle* con el propósito de enriquecer la estrategia y la dinámica de las partidas. Estas habilidades permiten a los jugadores influir directamente en el desarrollo del juego, favoreciendo la toma de decisiones y fomentando un estilo de juego más activo y personalizado.

Los nombres de las habilidades deberán ser indicativos de su función, de manera que el jugador pueda comprender rápidamente su efecto con una simple lectura. Además, cabe señalar que los efectos de tipo temporal solo podrán aplicarse un número limitado de veces simultáneamente, siendo este límite determinado durante la fase de desarrollo según el balance y nivelado necesario. Antes de cada pregunta, los jugadores dispondrán de un tiempo específico para decidir si desean utilizar alguna de sus habilidades. Una vez transcurrido este período, en las pantallas de los jugadores se mostrarán las habilidades recibidas.

A continuación se muestran las habilidades diseñadas al inicio del proyecto:

- **50/50 -** De las cuatro opciones que normalmente se muestran, esta habilidad elimina

dos incorrectas, dejando al jugador con un 50 % de probabilidades de acertar o fallar.

- **Puntuación Doble** - El jugador que utilice esta acción verá multiplicados por dos los puntos obtenidos en la siguiente pregunta.
- **Más tiempo** - Esta habilidad tiene un efecto general, añadiendo tiempo adicional de respuesta a las preguntas.
- **Congelar** - Congela la pantalla de un jugador seleccionado, quien deberá tocar varias veces la pantalla para poder escoger una de las opciones de respuesta.
- **Ensuciar** - Ensucia la pantalla de un jugador seleccionado, obligándole a limpiarla deslizando el dedo sobre su dispositivo, creando un efecto similar a pasar una servilleta.
- **Encadenar Respuestas** - Las opciones de respuesta presentan uno o varios candados que el jugador seleccionado debe desbloquear tocándolos.
- **Oscurecer Respuestas** - Cada una de las respuestas aparece en negro, debiendo el jugador tocarla para que se muestre.
- **Menos tiempo** - Esta habilidad tiene un efecto general, reduciendo el tiempo de respuesta para todos los jugadores.

Igual que con los minijuegos, en el producto final del proyecto solo se han incluido cuatro habilidades: dos de carácter general, **Más tiempo** y **Menos tiempo**, y otras dos dirigidas a un jugador, **Congelar** y **Ensuciar**. Se ha dejado la estructura preparada para poder integrar más habilidades si así se desea.

3.10. Arquitectura general del proyecto

La aplicación de **Brain Battle** se ha diseñado siguiendo un enfoque modular, dividiendo el juego en distintos componentes, cada uno encargado de una responsabilidad específica. Este diseño permite que las funcionalidades estén menos entrelazadas entre sí, lo que facilita el mantenimiento, la comprensión del código y la incorporación de nuevas funcionalidades. Además, cada módulo mantiene una alta cohesión, centrándose en una única tarea, lo que contribuye a una estructura más ordenada y escalable del proyecto.

El proyecto tiene una naturaleza que lo divide en dos aplicaciones para diferentes dispositivos. Esto hace que, aunque la mayoría de las clases sean compartidas y tengan los mismos métodos, existan otras independientes para cada aplicación.

A continuación, se comentan los diferentes módulos pensados para el desarrollo de las clases:

- **Flujo Principal:** Contiene la lógica principal de la partida y se encarga de gestionar las fases del juego, así como de coordinar las llamadas a otros métodos y módulos principales. Al ser el módulo central, mantiene la mayor cantidad de relaciones con otros componentes y es, por tanto, menos independiente. Se puede considerar como la espina dorsal del sistema.
- **Jugador:** Gestiona la información y los métodos relacionados con los jugadores.
- **Conexión Red:** Se encarga de establecer y mantener la comunicación de red durante la partida.
- **Minijuegos:** Controla todo lo relacionado con los minijuegos en el juego.
- **Habilidades:** Encargado de gestionar todo lo relacionado con las habilidades en la partida.
- **Preguntas Locales:** Aquí se definen las clases encargadas de la agregación y visualización de las preguntas personalizadas en la base de datos.
- **Elementos de Interfaz:** Maneja las interfaces gráficas de los usuarios.
- **Audio:** Encargado de controlar el sistema de sonidos del juego y su volumen.
- **Comunicación Host y Cliente:** Define las clases que son utilizadas únicamente como comunicación entre la aplicación de PC y la de dispositivos móviles, gestionando el intercambio de información y la sincronización de datos entre ambos entornos.

Por último, se presenta un diagrama de clases que se definió al inicio del proyecto, con el objetivo de estructurar las principales clases del código. Este diagrama sirvió como guía durante el desarrollo, facilitando la organización y planificación de los distintos componentes del sistema.

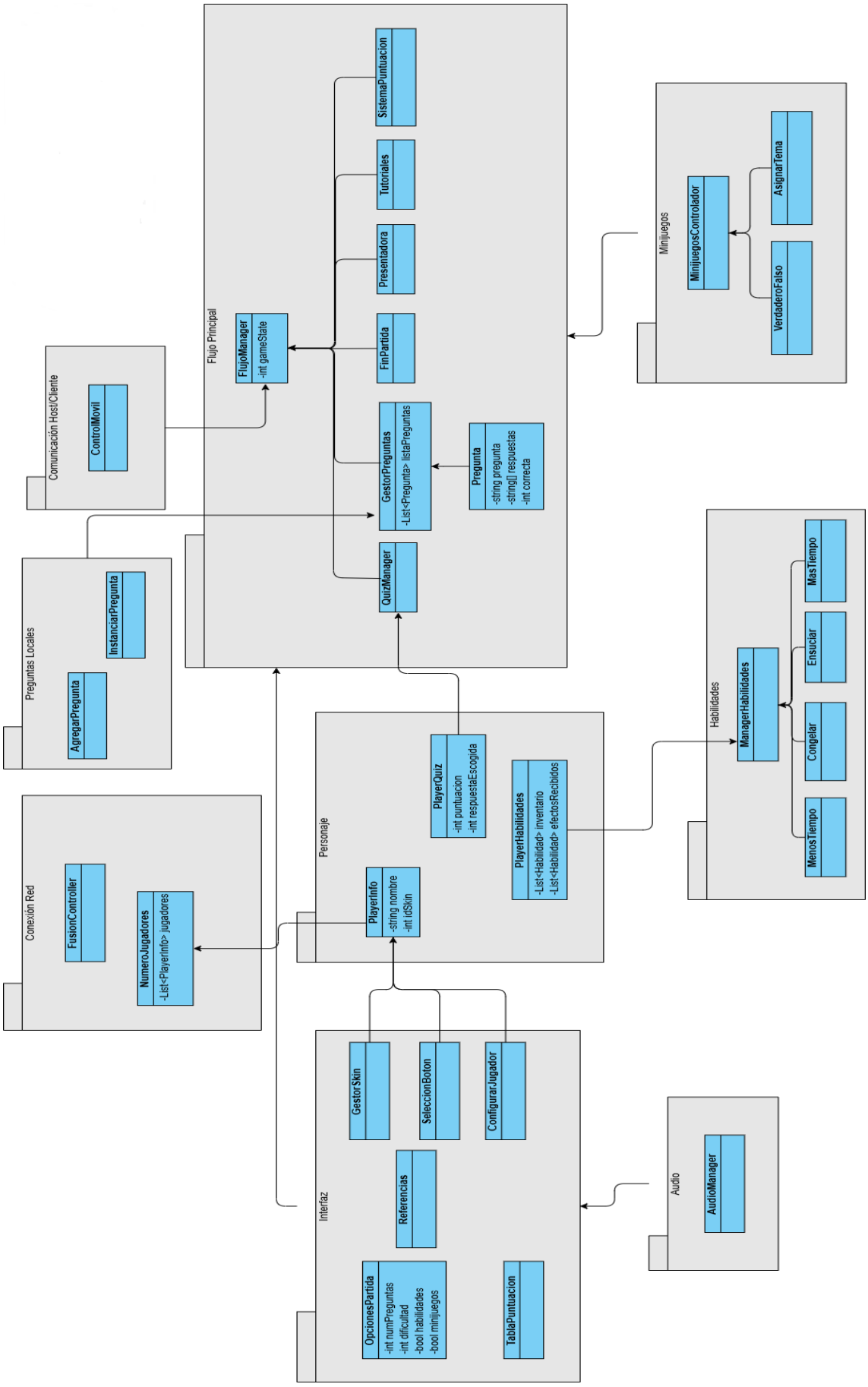


Figura 24: Diagrama de Clase

4

Implementación y Desarrollo

En esta sección se detalla el proceso de implementación de *Brain Battle* en Unity. Se explica la estructura de la base de datos seleccionada, el comportamiento de las clases más relevantes del proyecto y el sistema de comunicación en red utilizado, junto con su funcionamiento. Por último, se incluye una revisión del resultado obtenido, mostrando los menús e interfaces que conforman la versión final del juego.

4.1. Base de datos

Para la base de datos del sistema, se han utilizado varios archivos en formato **JSON** (JavaScript Object Notation). JSON es un formato ligero y ampliamente utilizado para el intercambio de información, basado en una estructura de pares clave-valor y listas que permiten organizar los datos de forma jerárquica y fácilmente legible. En este caso, se optó por JSON debido a su simplicidad, portabilidad y a la facilidad de integración que ofrece dentro de Unity, evitando la necesidad de un gestor de base de datos más complejo. Se ha considerado almacenar tanto las preguntas utilizadas en el juego, como la información usada para los minijuegos.

En el proyecto, los archivos **JSON** se integran en Unity como recursos de datos que el motor puede leer en tiempo de ejecución. Para ello, se almacenan dentro de la carpeta *Resources*. El proceso comienza con la lectura del archivo mediante la clase *TextAsset*, que permite acceder al contenido del archivo en formato de texto. A continuación, este texto se convierte en objetos de C# utilizando el método *JsonUtility.FromJson<T>()*.

A continuación, se especifica un poco más en la estructura de la base de datos:

- **Preguntas:**

  Para representar los datos, se han definido dos clases serializables: *Pregunta* y *Preguntas*. La clase *Pregunta* contiene los atributos *string pregunta*, un array de *string* para las respuestas, y un entero que indica la posición de la respuesta correcta. Por su parte, la clase *Preguntas* contiene varias listas de objetos *Pregunta*, una por cada nivel de dificultad definido en el juego.

  De esta manera, al cargar el JSON, el método mencionado convierte automáticamente los datos en instancias de estas clases, lo que permite acceder a las preguntas de forma estructurada, filtrar por nivel, mezclarlas o seleccionarlas aleatoriamente, integrando toda la información en el flujo de juego sin necesidad de modificar el código.

- **Minijuegos:**

  Para representar los datos de los minijuegos, se han definido clases serializables específicas para cada tipo de juego. En el caso del minijuego *ElegirTema*, se utilizan las clases *Palabra*, *Minijuego* y *ElegirTema*. La clase *Palabra* contiene los atributos *string texto*, que almacena la palabra a mostrar, y un entero *respuesta*, que indica la categoría correspondiente de la palabra. La clase *Minijuego* contiene dos cadenas de texto, *tema1* y *tema2*, que definen las categorías del juego, así como una lista de objetos *Palabra*. Finalmente, la clase *ElegirTema* contiene una lista de *Minijuego*, representando todos los bloques disponibles de este minijuego.

  Por su parte, el minijuego *VerdFal* utiliza las clases *Palabra*, *Minijuego* y *VerdFal*. En este caso, cada *Minijuego* tiene un atributo *tematica* que define el tema general del juego y una lista de *Palabra* con el texto a mostrar y su valor de respuesta (verdadero o falso). La clase *VerdFal* contiene la lista completa de minijuegos de este tipo.

- **Dialogos Presentadora:**

  Sin embargo, para los diálogos de la presentadora, **no se utiliza una base de datos en JSON como en los casos anteriores**, sino que se gestionan mediante un archivo de texto plano almacenado en la carpeta *Resources* de Unity. Este archivo actúa como una base de datos de diálogos y se organiza en bloques identificados por un entero, de

manera que cada bloque contiene una lista de frases correspondientes a un momento específico del juego.

La lectura del archivo se realiza mediante la clase *TextAsset* y la función *Resources.Load*, obteniendo el contenido en formato texto. Posteriormente, se divide en líneas y se construye un diccionario en C#, donde la clave será un número entero precedido de un guion, y los valores serán listas de cadenas de texto que contendrán todas las frases del bloque.

De esta manera, un bloque de diálogo puede estar definido en el archivo de texto de la siguiente forma:

```
-0
Hola, soy la presentadora.
Bienvenida al juego.
-1
Ahora pasemos a las preguntas.
```

y en memoria se almacena como:

```
dialogos[0] = ["Hola, soy la presentadora.", "
Bienvenida al juego."];
dialogos[1] = ["Ahora pasemos a las preguntas."];
```

Además, se han definido varios comandos especiales dentro del texto, que ayudan a mostrar u ocultar el contenedor donde se reproduce el texto (#SHOW, #HIDE) o sustituir dinámicamente el nombre del minijuego (#MINIJUEGO).

Cabe comentar que las preguntas definidas para el proyecto, así como los temas y sus palabras para los minijuegos, han sido sacados de ChatGPT. (OpenAI, 2023)

4.2. Implementación del multijugador

Para dotar a *Brain Battle* de una funcionalidad multijugador se ha utilizado Fusion 2 (Exit Games, 2025b), un motor de red desarrollado para Unity que permite sincronizar el estado del

juego entre varios jugadores de manera eficiente y con baja latencia. Fusion 2 está diseñado para gestionar tanto partidas en modo *hosted* como en modo client-server, ofreciendo herramientas para el manejo de sesiones, replicación de objetos e interpolación de movimientos, entre otras.

En la implementación del proyecto, Fusion 2 se integra creando un *NetworkRunner*, que actúa como el componente central de la sesión multijugador. Este componente se encarga de inicializar la partida en línea, gestionar la conexión de los jugadores y mantener sincronizados los objetos de red. Cada objeto que deba ser compartido entre clientes se instancia como un *NetworkObject*, el cual se registra automáticamente en el *NetworkRunner* y se actualiza de acuerdo con las reglas de sincronización definidas. El *networkRunner* se instancia gracias a una clase prediseñada por Fusion 2, llamada **FusionController**, de la que se hablará más adelante.

Fusion 2 permite definir distintas tipologías de ejecución, como la usada en este trabajo, *Shared*, donde todos los jugadores ven y afectan al mismo estado del juego, junto con *State Authority*, donde un jugador o el servidor tiene autoridad sobre ciertos objetos. Esta flexibilidad ha sido aprovechada para gestionar todo el flujo de la partida, donde se dispone de un cliente principal que actúa como host y *State Authority* en el PC, y el resto de clientes (dispositivos móviles) se conectarán a este host principal.

Además, Fusion 2 incluye sistemas de callbacks y eventos que facilitan la integración con la lógica del juego en Unity. Gracias a esto, se puede mantener la consistencia entre la representación visual en cada cliente y el estado interno del juego, reduciendo la complejidad de tener que implementar manualmente la sincronización de cada acción.

4.2.1. Fusion Controller

FusionController es un script proporcionado por Fusion 2 que implementa eventos de red mediante la interfaz *INetworkRunnerCallbacks*. Este componente se asocia a un objeto principal en la escena de Unity, marcado como *DontDestroyOnLoad* para que permanezca activo y no se destruya al cambiar de escena. Cabe destacar que este script tiene dos versiones diferentes, una para el host y otra para los clientes. A continuación, se explican los métodos redefinidos específicamente para este proyecto, empezando por la clase definida para el host:

- **StartGame()**. Método *asincrono*, lo que significa que se ejecuta de manera no bloqueante, permitiendo iniciar operaciones que pueden tardar en completarse sin detener otras

operaciones del juego. Este método agrega a una variable de tipo *NetworkRunner* el componente del mismo nombre, indicando que este cliente no enviará inputs de jugador (*ProvideInput = false*). Se procede a generar un código único mediante la clase *RoomCodigo*, que devuelve una cadena con el formato de cuatro letras y dos dígitos, utilizada como nombre de la sesión.

Posteriormente, se crea la sala en el servidor con este código, estableciendo el cliente en modo Host mediante la estructura *StartGameArgs*. Finalmente, si la sala es creada correctamente, se activará el panel correspondiente a la espera de jugadores en la escena y se instanciará un objeto prefab encargado de guardar el número de jugadores que se conectan y sus referencias.

Cabe destacar que este método es público y se llama desde el menú de configuración de partida.

- **OnPlayerJoined()**. Este método recibe como parámetros el controlador de red *NetworkRunner* y un objeto de tipo *PlayerRef*, que representa una referencia al cliente que ha generado el evento.

  El evento se ejecuta cada vez que un nuevo cliente se conecta al servidor. Su lógica principal solo se desarrolla si el método es ejecutado por el servidor y la referencia del jugador que se pasa como parámetro no corresponde al propio servidor.

  Cuando se cumplen estas condiciones, se instancia en todos los clientes (incluido el host mismo) un prefab predefinido para representar al jugador y se añade a una lista interna de la clase, asociando la referencia de red del jugador con el objeto instanciado.
- **OnPlayerLeft()**. De la misma manera que el método anterior, este recibe los mismos parámetros de entrada. Sin embargo, este evento se ejecuta cuando un cliente se desconecta de la red. Entonces se intenta buscar su referencia dentro de la lista interna de la clase, y si es encontrado elimina su instancia de todas las escenas de los clientes, y además borra su referencia de la lista.
- **OnShutdown()**. Este evento se llama cuando el servidor se cierra. Recibe como parámetros el controlador de red y una variable de tipo *ShutdownReason*, que indica la razón por la que se ha cerrado el servidor. Si la razón es *Ok*, se carga automáticamente la escena del menú inicial, asegurando que el cliente principal vuelva a la interfaz inicial del juego.

- **OnSceneLoadDone()**. Este método solo recibe como parámetro el controlador de red y es llamado cuando se carga una nueva escena en el servidor.
  Solo se ejecuta si el controlador de red es el servidor; si es así, llamará al método interno *CargarQuizManage*, explicado más adelante.

En la siguiente lista se explicarán los métodos propios definidos para esta clase.

- **CargarJuego()**. Método encargado de iniciar el juego, cargando la escena principal de este.
- **CargarQuizManage()**. Este método se encarga de instanciar un prefab que contiene las clases principales para la gestión de la partida. Este objeto se instancia únicamente una vez por el servidor, garantizando que solo exista una instancia por partida en cada cliente. El prefab incluye los siguientes componentes clave: *QuizManage*, *Minijuegos-Controlador* y *ManagerHabilidad*, los cuales centralizan la lógica de preguntas, minijuegos y habilidades dentro del juego. Por otra parte, busca el objeto con el componente *FlujoManager* y lo inicializa pasando las referencias de los componentes del prefab comentados anteriormente.
- **DesconectarJugadores()**. Este método se utiliza para cerrar la sesión del servidor de manera controlada. Llama al método *Shutdown(true, ShutdownReason.Ok)* de la clase *NetworkRunner*, iniciando el cierre del servidor y desconectando a todos los jugadores conectados. Previamente, se incluye un retraso de 3 segundos mediante *Task.Delay*, para permitir que se completen operaciones pendientes antes de cerrar la sesión.

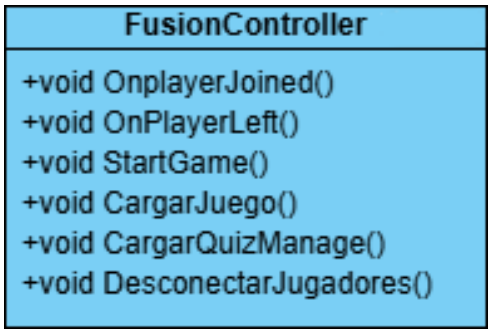


Figura 25: Diagrama de Clase FusionController

Ahora se va a proceder a explicar las principales diferencias en los métodos redefinidos en la clase *FusionController* de la versión móvil. Los métodos no mencionados significa que no tienen ninguna lógica en esta clase.

- **StartGame()**. La principal diferencia de este método respecto a su homólogo es que el código de la sala no es generado automáticamente por ninguna clase, sino que debe ser introducido manualmente por el jugador. Además, este método inicia la conexión en modo *Client* y no como host. En caso de que, al intentar unirse a la sala, se produzca algún error, el jugador es redirigido a la escena inicial y se llama al método *ResetearFusionController()*.
- **OnShutdown()**. Este evento, dependiendo de si la razón es *Ok* o *DisconnectedByPluginLogic*, redirige al jugador a la escena del menú principal o a una escena de error de conexión, respectivamente.
- **OnConnectedToServer()**. Activa el panel de configuración del jugador.

Por último, los métodos propios definidos para esta clase son los siguientes:

- **ResetearFusionController()**. Encargado de buscar su propia referencia en el juego y llamar al método *InicializarFusionController()*.
- **InicializarFusionController()**. Este método se creó para reiniciar todas las variables necesarias del *FusionController*. Se implementó debido a un problema que ocurría al intentar conectarse a una sala inexistente, lo que provocaba la pérdida de todas las variables configuradas en el inspector de Unity. Por ello, se desarrollaron este y el método *ResetearFusionController()*.
- **FinalizarConexion()**. Simplemente finaliza la conexión con el servidor.

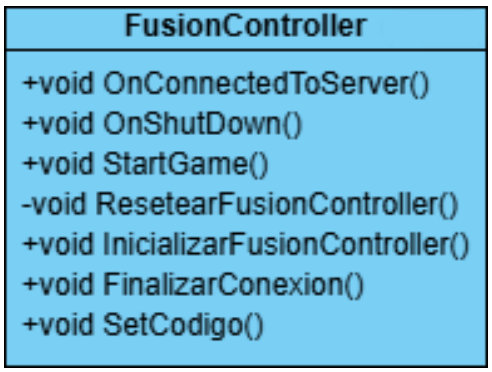


Figura 26: Diagrama de Clase FusionController

Una vez establecida la conexión con el servidor y creada la sala de la partida, el host queda a la espera de que los jugadores estén listos. Esto se controla mediante el atributo booleano *listo* de la clase *PlayerInfo*. La clase *ControladorPreparados* se encarga de verificar continuamente, en

cada *frame*, si todos los jugadores tienen esta variable en *true*. Cuando esto ocurre, se habilita el botón que permite iniciar la partida.

4.3. Sistema principal del juego

En este apartado se va a explicar el funcionamiento del flujo principal de una partida de *Brain Battle*. Una vez creado el servidor y confirmada la preparación de todos los jugadores, se carga en ambas aplicaciones la escena principal donde se desarrolla la partida.

Dicha escena se compone de un *Canvas* principal que contiene varios paneles, los cuales se activan o desactivan dinámicamente según el momento del transcurso del juego. De esta manera, se gestionan las diferentes fases de la partida de forma visual y organizada. Además de los *GameObjects* mencionados, la escena incorpora otros objetos cuya finalidad es exclusivamente el control y gestión de los distintos sistemas del juego.

Figura 27: Objetos escena juego host

Figura 28: Objetos escena juego cliente

Existe una clase dedicada específicamente a centralizar todas las referencias necesarias

del juego en ambas aplicaciones. En ella se almacenan los elementos principales de la escena, facilitando así su acceso desde otros componentes.

Diversas clases consultan estas referencias mediante el uso del método heredado de *MonoBehaviour*, *FindFirstObjectByType<Class>()*, el cual devuelve el primer componente encontrado que coincida con la clase indicada.

Gracias a este enfoque, es posible activar o desactivar paneles, botones u otros elementos de la interfaz de manera más fluida y estructurada, evitando redundancias en el código y mejorando la legibilidad de la implementación.

Una vez cargada la escena principal del juego, el primer método relevante en ejecutarse es *Start()* de la clase *FlujoManager*. Esta clase tiene la responsabilidad de controlar y gestionar los diferentes estados de la partida, actuando como un coordinador central que determina el flujo entre fases y la activación de los sistemas y/o elementos necesarios en cada momento.

A continuación, se procederá con la explicación y descripción detallada de esta clase.

4.3.1. Flujo Manager

Una de las clases más importantes de todo el proyecto es **FlujoManager**, ya que se encarga de coordinar cuándo deben ejecutarse las demás clases y sus métodos. Para ello, define inicialmente una estructura enumerada denominada *State*, la cual representa los diferentes estados posibles de la partida. Esta enumeración facilita el control del flujo del juego, ya que cada valor corresponde a una fase concreta del desarrollo, como se muestra en la imagen.

```
//Estados del flujo
public enum State : int
{
    Intro = 0,
    Habilidades = 1,
    Pregunta = 2,
    Respuesta = 3,
    MostrarTabla = 4,

    PresentadoraPregunta = 5,
}
```

Figura 29: Estados definidos para la partida

En esta clase se almacenan atributos esenciales para el desarrollo de la partida, como los tiempos definidos para cada fase, los clips de audio que se reproducirán en momentos concretos y otras variables configurables por el jugador, por ejemplo, la dificultad, el número de preguntas o la activación de tutoriales, habilidades y minijuegos.

El método **Start()** es el primero en ejecutarse y tiene como objetivo inicializar las referencias necesarias para el funcionamiento de la clase. Estas referencias provienen de dos fuentes principales: por un lado, de otras clases localizadas en la escena mediante *FindFirstObjectByType<Class>()* y, por otro, de los componentes del *prefab QuizManager*, los cuales son asignados a través de métodos *Setter()* proporcionados por la clase *FusionController*. Además, en este método se inicializan variables internas propias del flujo de la partida.

También se consultan los valores definidos en la clase **OpcionesPartida**, como la dificultad, el número de preguntas o la activación de tutoriales, habilidades y minijuegos, para almacenarlos y utilizarlos durante el desarrollo del juego. Finalmente, se establece la variable *gameState* con el valor *Intro*, indicando la fase inicial de la partida, y se lanza una corrutina que ejecuta el método *FaseIntro()*.

Una **corrutina** en Unity es un tipo especial de método que permite ejecutar código de manera secuencial a lo largo del tiempo sin bloquear el flujo principal del juego. A diferencia de los métodos normales, que se ejecutan de principio a fin en un solo *frame*, una corrutina puede pausar su ejecución en determinados puntos utilizando la instrucción *yield* y reanudarla posteriormente en *frames* futuros. Esto resulta especialmente útil para tareas como esperar un tiempo determinado, realizar animaciones graduales o gestionar secuencias de eventos que dependen del tiempo.

A continuación, se van a explicar los métodos más importantes definidos para esta clase y el flujo principal de preguntas:

- **NextPhase() y EjecutarFase()**. Son los métodos encargados de calcular, según un orden establecido en un *Array* de *State*, cuál es la siguiente fase que procede. Y mediante un *switch*, llamar al método correspondiente con la lógica de esa fase.
- **Update()**. El *Update()* actualiza en cada *frame* la variable *time*, restándole el tiempo pasado en el sistema real. Esta es inicializada por los métodos de las fases correspondientes, y cuando su valor es menor que cero, procede a llamar al método **NextPhase()**. De esta

manera se controla el flujo de las fases con los tiempos definidos.

- **FaseIntro()**. Método encargado de activar una pequeña presentación narrada por la presentadora, así como de llamar a la clase **Tutorial** para reproducir, si es necesario, los primeros tutoriales de la partida.
- **FaseHabilidades()**. Método encargado de gestionar las llamadas a los métodos correspondientes, principalmente de la clase **ManagerHabilidades**.
- **FasePreguntas()**. Previo a este método, existe otro llamado *FasePresentadoraPreguntas()*, el cual está encargado de obtener una pregunta y sus respuestas consultando la clase **GestorPreguntas**, además de presentar la pregunta con la presentadora. El método **FasePreguntas()** es realmente el encargado de proporcionar la pregunta obtenida anteriormente a la clase **QuizManager**, la cual se encargará de transmitir dicha cuestión al resto de clientes y activará los paneles de las interfaces correspondientes.
- **FaseRespuesta()**. Método encargado de llamar a los métodos de la clase **QuizManager**: *EscribirRespuestas()*, *MostrarRespuestas()Rpc* y *ComprobarRespuestas()*, responsables de enseñar la respuesta correcta a la pregunta formulada, así como de calcular la puntuación obtenida para cada jugador.
- **FaseComprobarRespuesta()**. Este método inicia el flujo para mostrar a los jugadores su puntuación obtenida, así como finalmente una tabla con las posiciones actualizadas de la partida.
- **FinPartida()**. Bloquea el flujo del código y llama a la presentadora para reproducir un mensaje de despedida.
- **ResetearPartida()**. Reinicia las variables necesarias para comenzar una nueva partida con las opciones escogidas al inicio, y desbloquea el flujo de la partida iniciándola otra vez.
- **InformarMovil()**. Método fundamental encargado de comunicarse con la clase **ControlMovil**. Esta clase, presente en cada *prefab* de los jugadores, posee un método llamado *InformarCambioRpc()*, al cual se le pasa un entero que representa una acción específica. Dicho entero se retransmite a todos los jugadores en su cliente y, mediante un *switch*, se ejecuta el método correspondiente para esa acción en todos los clientes, asegurando la sincronización de eventos entre ellos.

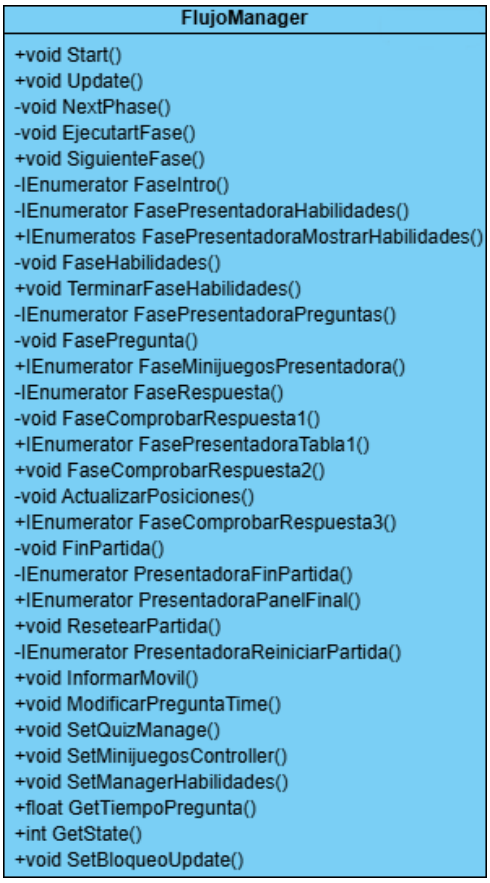


Figura 30: Diagrama de Clase FlujoManager

En los siguientes apartados se van a explicar algunas clases principales, las cuales son llamadas a través de los métodos explicados previamente.

4.3.2. Gestor Preguntas

La clase *GestorPreguntas* se encarga de cargar, organizar y proporcionar las preguntas del juego según la dificultad seleccionada y las preferencias del jugador. Para ello, se definen tres niveles de dificultad: **Primaria, Secundaria y Universidad**. Las preguntas se almacenan en listas separadas tanto para las preguntas generales como para las locales del jugador.

Cuando se cargan las preguntas desde la base de datos y los archivos locales, se ejecuta el método *SeleccionDificultad()*, que crea dos listas: *listaPreguntasPartida*, que contiene las preguntas de la dificultad seleccionada, y *listaInferior*, que almacena preguntas de una dificultad inferior. Esto se debe a que existe la probabilidad, definida por el jugador, de que pueda aparecer una pregunta más fácil.

Si el jugador decide jugar únicamente con preguntas personalizadas, la lista principal se inicializa primero con las preguntas locales. Si no se alcanza el mínimo de preguntas necesarias para la partida, definido también por el jugador, se ejecuta el método *AgregarPreguntasNecesarias()*, que añade de manera aleatoria preguntas generales hasta completar el número requerido.

Finalmente, la clase incluye el método **ObtenerPreguntaAleatoria()**, que devuelve una pregunta de la lista principal o de la lista inferior según la probabilidad establecida. Antes de devolverla, se mezclan aleatoriamente las respuestas con el método privado *Mezclar()*, para asegurar que el orden no sea siempre el mismo. La pregunta obtenida se elimina de la lista correspondiente; si dicha lista queda vacía, se vuelve a ejecutar el método *SeleccionDificultad()* para rellenarla automáticamente.

4.3.3. Opciones de Partida

La clase **OpcionesPartida** se encarga de gestionar y almacenar la configuración de la partida seleccionada por el jugador antes de comenzar el juego. Entre sus funciones se incluyen la selección de la dificultad, el número de preguntas, la activación de habilidades, minijuegos y tutoriales, así como la configuración de ciertos parámetros de la interfaz, como el porcentaje de preguntas inferiores.

Para mantener estas configuraciones entre sesiones, la clase utiliza *PlayerPrefs*, un sistema de Unity que permite guardar valores simples (como enteros, cadenas o booleanos) en el dispositivo. Esto asegura que, cuando el jugador vuelva a iniciar el juego, sus opciones previamente seleccionadas se restauren automáticamente.

Además, esta clase se encarga de actualizar los elementos de la interfaz (como menús desplegables, *toggles* y *sliders*) según los valores guardados, y proporciona métodos *getter()* para que otras clases puedan consultar la configuración actual de manera sencilla y segura.

4.3.4. Quiz Manager

La clase **QuizManager** también se puede considerar una de las más importantes de todo el sistema, ya que, de forma general, se encarga de gestionar todo lo relacionado con la presentación y control de las preguntas en la partida, siendo esta clase la que se comunica entre

clientes.

Esta clase está presente tanto en el host como en los clientes y maneja la función de actualizar la interfaz de usuario con la pregunta y las posibles respuestas, gracias a los siguientes métodos:

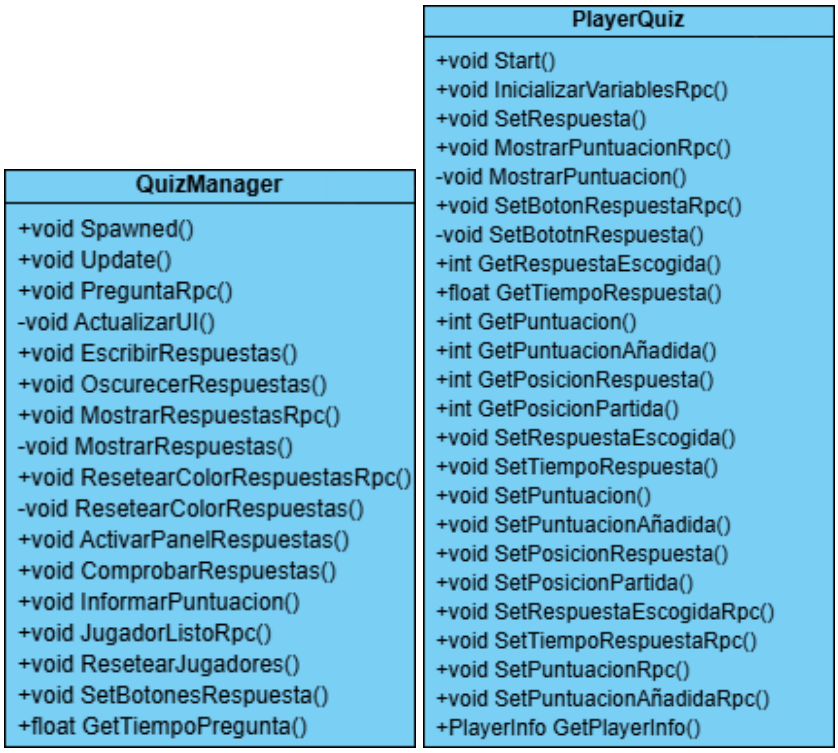
- **PreguntaRpc() y ActualizarUI()**. *PreguntaRpc()* es un método de red que solo puede ser llamado por el servidor y se propaga automáticamente a todos los clientes. Su función principal es ejecutar localmente, en cada cliente y en el servidor, el método *ActualizarUI()*, el cual se encarga de mostrar la pregunta y sus posibles respuestas en la interfaz, así como de habilitar que los jugadores puedan responder.

- **MostrarRespuestas()**. Este método es llamado desde el servidor hacia todos los clientes y cambia el color de los paneles de las respuestas, indicando con verde cuál es la respuesta correcta y con rojo la incorrecta. Además, bloquea los botones para que los jugadores no puedan interactuar hasta la siguiente pregunta.

- **ResetarColorRespuestas()**. Devuelve el color original a los paneles que fueron modificados previamente.

A su vez, esta clase también se encarga, junto con la clase **SistemaPuntuacion**, de gestionar las puntuaciones obtenidas por los jugadores en cada pregunta. El método *ComprobarRespuestas()* recorre la lista de jugadores en el servidor y calcula la puntuación obtenida por cada jugador en función del tiempo de respuesta. Tanto el tiempo de respuesta como la respuesta escogida se almacenan en la clase **PlayerQuiz**, y tras realizar el cálculo, el resultado se guarda en la variable *puntuacionAñadida* de la misma clase. Posteriormente, mediante el método *InformarPuntuacion()*, la puntuación obtenida en la pregunta se suma a la puntuación total de cada jugador y se reinician todas las variables correspondientes para el siguiente cálculo.

Por último, la clase también incluye el método *ResetearJugadores()*, que permite, desde el servidor, reiniciar todas las variables de cada jugador relacionadas con la partida.



(a) QuizManager (b) PlayerQuiz

Figura 31: Diagramas de Clases

4.3.5. Sistema de Puntuación

Para el sistema de puntuación se decidió que no solo los jugadores obtendrían puntos según respondiesen correctamente o no, sino que también dependería de la velocidad con la que lo hicieran respecto a otros jugadores.

De esta forma, el método *ObtenerPuntuacion()* de la clase **SistemaPuntuacion** se encarga de calcular la puntuación de un jugador en función de su orden de respuesta respecto al resto de participantes. Se calcula un valor proporcional basado en la posición relativa del jugador mediante una interpolación lineal (*Mathf.Lerp*) entre el puntaje máximo y mínimo definidos en la clase. El resultado se redondea al entero más cercano y se devuelve como la puntuación final del jugador.

La puntuación de los minijuegos, en cambio, se calcula únicamente en función del número de respuestas correctas que cada jugador haya obtenido durante el juego.

La puntuación se calcula después de cada pregunta o minijuego, pero la forma de mostrar

los puntos obtenidos junto con la puntuación total varía entre el host principal y los clientes. En el host, la clase **TablaPuntuaciones** se encarga de gestionar la interfaz que muestra las primeras posiciones de los jugadores y sus puntuaciones acumuladas. Esta clase utiliza una serie de *prefabs* para crear dinámicamente cada entrada de la tabla, mostrando el nombre del jugador, su puntuación total y, al lado, los puntos obtenidos en la pregunta más reciente.

Figura 32: Entrada Tabla Puntuaciones

Para los clientes móviles, en cambio, se actualizan directamente los componentes de *TextMeshPro* en el panel resultados. TextMeshPro es un componente de renderizado de texto en Unity que permite mostrar textos de alta calidad, con opciones de formato, estilos y efectos gráficos que superan al componente de texto estándar.

El flujo de mostrar los resultados a los jugadores se realiza de la siguiente manera:

- Primero, se activan los paneles correspondientes en cada dispositivo. En el host se llama al método **ActivarTabla()** de la clase *TablaPuntuaciones*, que muestra la tabla con las cuatro primeras posiciones y la puntuación acumulada hasta ese momento, mostrando al lado los puntos obtenidos en la pregunta actual. En los dispositivos móviles, en cambio, se muestra únicamente la posición del jugador y los puntos obtenidos en esa pregunta.
- Tras unos segundos, el host llama al método **ActualizarTablas()** de la misma clase. Este método elimina las entradas anteriores y genera la tabla actualizada con las nuevas puntuaciones y posiciones de los jugadores, incluyendo los puntos ya sumados de la pregunta reciente. En los móviles, solo se actualiza la posición del jugador y su puntuación total.

Por último, se decidió modificar el diseño relacionado con la visualización de las puntuaciones finales. Inicialmente, se planteó implementar una pantalla final que mostrara las puntuaciones globales de todas las partidas, actualizándose de forma continua. Sin embargo, se descartó esta mecánica al comprobar que no tenía demasiado sentido, ya que la puntuación máxima alcanzada depende en gran medida del número de jugadores. En partidas con un solo jugador, es más probable obtener una puntuación elevada, mientras que en aquellas con más

participantes, la puntuación tiende a repartirse, dificultando la comparación directa entre unas y otras.

4.3.6. Presentadora

La presentadora aporta un componente de dinamismo a las partidas, acompañando al jugador durante la sesión. Es ella quien se encarga de explicar los tutoriales y de ir guiando a los jugadores entre secciones.

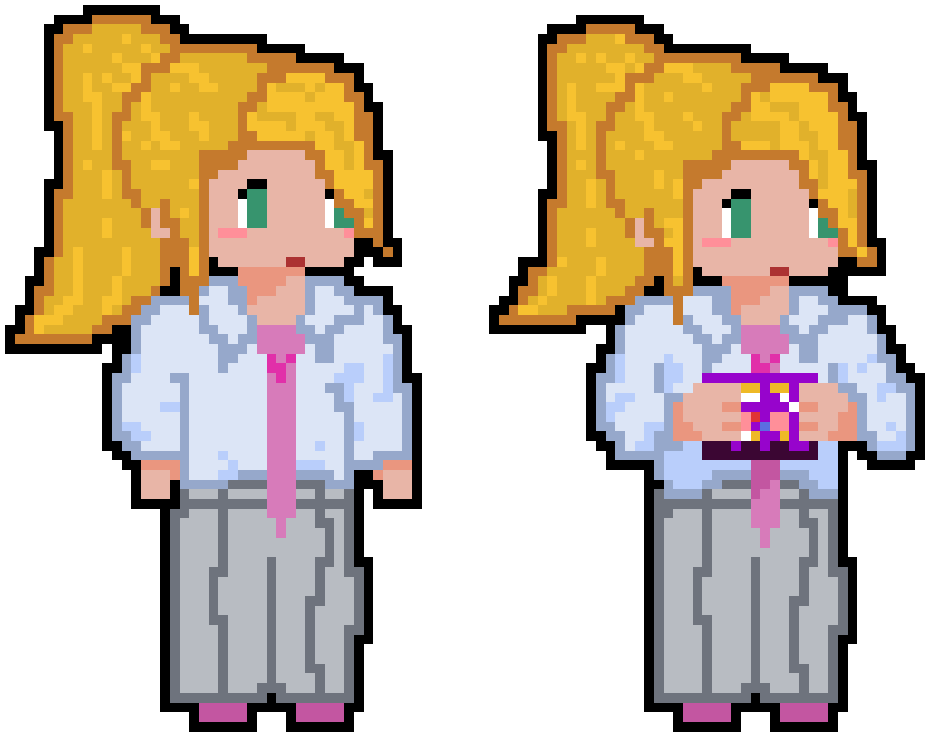


Figura 33: Diseño de la Presentadora

El diseño principal del personaje, así como todas sus animaciones, ha sido creado por María Aron.

La presentadora cuenta con tres animaciones principales: *Idle*, *Hablando sin tarjeta* y *Hablando con tarjeta*. En Unity, estas animaciones se gestionan mediante el componente **Animator**, que se encarga de controlar toda la lógica de animación. Dado que la presentadora no es un personaje que tenga que reaccionar a inputs del jugador, no es necesario calcular sus animaciones en base a estos; en cambio, desde el código se puede acceder al **Animator** y activar directamente la animación deseada de manera sencilla y precisa.

El funcionamiento de la presentadora está definido en la clase **Presentadora**, la cual incluye varios métodos encargados de controlar su interfaz y mostrar los diálogos al jugador. Los métodos más relevantes son los siguientes:

- **CargarTexto()**. Como se explicó en el apartado de **Base de Datos**, este método almacena los diálogos de la presentadora en un diccionario clave-valor, donde cada entrada contiene un identificador y el párrafo correspondiente.
- **ConfigurarPaneles()**. Se encarga de activar o desactivar los paneles de la interfaz según el identificador que se le pase como argumento, permitiendo mostrar únicamente la información relevante en cada momento.
- **Hablar()**. Se trata de una corrutina que muestra un diálogo en la interfaz y no finaliza hasta que todo el texto ha sido desplegado. Recibe como argumento un booleano que indica si la presentadora debe ejecutar la animación con tarjeta o sin tarjeta.

  Este método tiene dos versiones: una que recibe un identificador del diálogo que se desea reproducir, y otra que recibe directamente una cadena de texto, utilizada principalmente para mostrar las preguntas al jugador.

4.3.7. Tutoriales

Se han definido tutoriales para explicar distintas funciones del juego: la lógica principal de las preguntas y respuestas, el uso de habilidades y un tutorial específico para cada minijuego implementado. Cada tutorial se mostrará únicamente una vez por partida y siempre antes de la fase correspondiente a la función o minijuego al que está asociado.

Los tutoriales han sido creados utilizando la herramienta de *Unity Timeline*, que permite generar cinemáticas, animaciones complejas y secuencias de eventos de manera visual y cronológica. Dentro del *Timeline*, los elementos principales son las pistas (*tracks*), que representan diferentes tipos de acciones o eventos a lo largo del tiempo. Cada pista controla un aspecto concreto de los objetos en escena. Para los tutoriales se han usado principalmente tres tipos de pistas: *Activation Track*, que activa o desactiva objetos de la escena; *Animation Track*, que modifica valores y atributos de los componentes de los objetos; y *Signal Track*, que utiliza las señales de Unity, las cuales envían un mensaje en un momento específico a un componente objetivo, el *Signal Receiver*. En este caso, este receptor está colocado en la presentadora y

define la acción a ejecutar al recibir la señal. Con estos elementos se han diseñado todas las cinemáticas de los tutoriales.

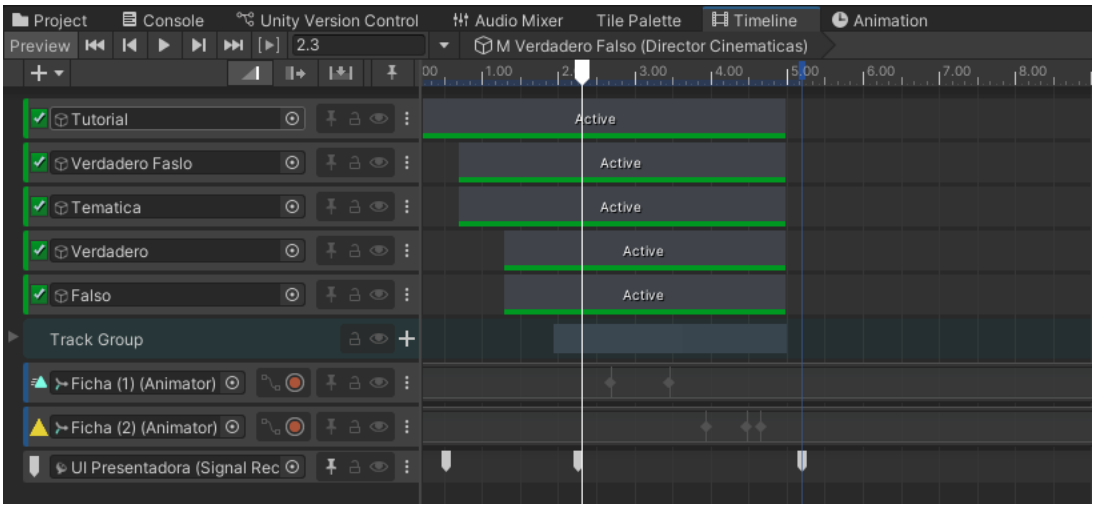


Figura 34: Herramienta Timeline

Para gestionar la lógica de los tutoriales se ha definido la clase **Tutorial**, en la que se incluyen varios métodos clave. El método *IniciarTutorial()* recibe como parámetro un id que indica qué tutorial debe ejecutarse. Por su parte, *PausarTimelinePresentadora()*, junto con el método *EsperarTexto()*, se encarga de reproducir los diálogos de la presentadora durante las cinemáticas. Finalmente, *TerminoClip()* indica que el tutorial ha finalizado, lo que permite que el flujo principal del juego espere a que concluya antes de continuar con la partida.

4.3.8. Fin de Partida

La clase **FinPartida** es la responsable de mostrar los resultados finales a los jugadores y de gestionar el flujo de decisiones al terminar la sesión. Existen dos opciones al finalizar: reiniciar la partida con la misma configuración o regresar al menú principal.

En cuanto a los resultados, en el host se muestran únicamente las tres primeras posiciones, simulando un podio en el que aparecen los avatares junto al nombre y la puntuación de cada jugador. Estos avatares cuentan con una animación de victoria específica según la posición obtenida. En el caso de los clientes, se muestra su puesto final y su puntuación total, añadiendo la animación de victoria si el jugador se encuentra entre los tres primeros.

La lógica de este proceso se implementa en el método *MostrarResultadosPartida()*, que recibe una lista ordenada de los jugadores y, mediante referencias internas, activa los componen-

tes necesarios para la visualización. A continuación, se invoca el método *InformarMovil()* de la clase **FlujoManager**, enviando el identificador correspondiente para que el cliente reciba también sus resultados. Finalmente, se inicia una corrutina en la que la presentadora despide la partida.

Tras esta corrutina se ejecuta el método *PanelFinal()* de la clase **FinPartida**, encargado de mostrar los botones disponibles al término de la sesión y de activar una pantalla de pausa en los dispositivos móviles. Entre las opciones se incluyen:

- **Finalizar partida**: botón asociado al método *TerminarPartida()*, que elimina todos los objetos marcados como *DontDestroyOnLoad* mediante la clase **ListaObjetosDontDestroyOnLoad**, que guarda una referencia de dichos objetos, y posteriormente desconecta a todos los clientes utilizando *DesconectarJugadores()* de la clase **FusionController**.

- **Reiniciar partida**: botón vinculado al método *ResetearPartida()*. Este desactiva el panel actual y delega en la clase **FlujoManager** la inicialización de las variables necesarias tanto del host como de los jugadores, permitiendo comenzar una nueva partida sin problemas.

4.4. Minijuegos

Los minijuegos representan una forma de hacer más divertida una partida de *Brain Battle*, ya que permiten a los jugadores romper la monotonía del flujo principal. En ellos se ponen a prueba otras habilidades, como la asociación rápida de conceptos o la memorización, entre otras. La fase de minijuegos únicamente se reproduce si el jugador ha decidido activarla, y aparece tras cada bloque de tres preguntas.

Durante la fase de diseño se plantearon distintos minijuegos, aunque finalmente solo se introdujeron dos en el sistema: **Tema Correcto** y **Verdadero o Falso**. Ambos comparten una estructura similar, en la que los jugadores deben mover una o varias fichas hacia paneles concretos.

En el minijuego **Tema Correcto**, el jugador debe mover una ficha hacia el panel correspondiente al tema indicado. Independientemente de si acierta o no, el juego cargará una nueva ficha hasta que se agote la lista de palabras o finalice el tiempo.

Por su parte, en **Verdadero o Falso** el jugador debe decidir si varias fichas que aparecen en pantalla pertenecen al tema central o no, colocándolas en los paneles de verdadero o falso según corresponda. Una vez que el jugador coloca todas las fichas de un grupo, aparece un nuevo conjunto mientras queden elementos en la lista asociada y el tiempo no haya finalizado.

Ambos minijuegos se basan en el concepto de fichas y paneles, sobre los cuales se desarrolla la mecánica de interacción. A continuación se explica la lógica detrás de estos elementos.

4.4.1. Ficha

La ficha es la pieza fundamental de los minijuegos. Está compuesta por un *Sprite2D*, diseñado por la artista del proyecto, y varios componentes, entre los que destacan un *TextMeshProUGUI*, utilizado para mostrar la palabra asociada a la ficha, y la clase **Ficha**.

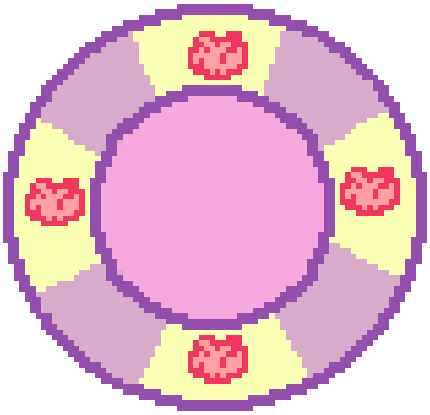


Figura 35: Diseño Ficha

La clase **Ficha** tiene como variables principales una referencia al componente mencionado *TextMeshProUGUI*, y una variable de tipo entero llamada *tema*, que representa el identificador de la ficha y se utiliza para determinar a qué panel corresponde como respuesta correcta.

Además, la clase implementa varias interfaces que permiten mover la ficha durante la partida:

- **IBeginDragHandler**: implementa el método *OnBeginDrag()*, que se ejecuta cuando la ficha es pulsada y comienza a arrastrarse.
- **IDragHandler**: implementa el método *OnDrag()*, encargado de la lógica mientras la ficha se encuentra en movimiento.

- **IEndDragHandler**: implementa el método *OnEndDrag()*, que contiene la lógica al finalizar el arrastre de la ficha.

La clase también define dos métodos propios:

- *ResetFicha()*, que devuelve la ficha a su posición original.
- *SetFicha()*, utilizado para establecer la palabra y el tema correspondiente ligados a ella.

Cabe destacar que en el método *Awake()*, ejecutado al iniciar la escena, se configura la propiedad **alphaHitTestMinimumThreshold** del componente *Image* con un valor de 0.1. Esto permite que el jugador solo pueda pulsar y arrastrar la ficha en las zonas dibujadas, evitando la interacción en áreas vacías de la imagen.

Con todo ello, la ficha puede moverse libremente por la pantalla de forma intuitiva y fluida.

4.4.2. Metodos de los paneles

Los paneles utilizados en ambos minijuegos cuentan con dos clases para su gestión, dependiendo del minijuego, aunque en esencia son muy similares y solo difieren en un detalle. Estas clases son **ComprobarVerdadPanel** y **ComprobarTemaCorrecto**. Ambas implementan la interfaz **IDropHandler**, que define el método *OnDrop()*, el cual se ejecuta cuando un objeto es soltado sobre el panel. El método *OnDrop()* recibe como parámetro una referencia al evento que lo invoca, en este caso el objeto arrastrado. Se comprueba que dicho objeto contiene el componente *Ficha* y, dependiendo de si su variable *tema* coincide con la variable homóloga de la clase del panel, se determina la acción a realizar:

- Si ambas variables coinciden, se suma una puntuación al jugador y se muestra un parpadeo verde durante un breve instante.
- Si no coinciden, se muestra un parpadeo rojo y no se añade puntuación.

La diferencia principal entre estas clases radica en la forma de resetear las fichas. Ambas llaman al método correspondiente de la clase de su minijuego, el cual se encarga de cargar la siguiente palabra o grupo de palabras.

4.4.3. Clase especificas para los minijuegos

Cada minijuego tiene asociada una clase específica: **VerdaderoFalso** y **AsignarTemas**. Ambas comparten ciertos métodos similares en su funcionamiento general, como los que permiten iniciar una cuenta atrás antes de comenzar el minijuego o reiniciar el mismo.

La principal diferencia entre ambas radica en la forma de gestionar las palabras:

- La clase **AsignarTemas** se encarga de extraer, una a una, las palabras relacionadas con el minijuego, mediante el método *CargarPalabraFicha()*, desde una lista recibida por la clase **MinijuegosControlador**, asignándolas a una única ficha.
- La clase **VerdaderoFalso**, mediante los métodos *ComprobarNumeroFichasActivas()* y *CargarPalabraFicha()*, gestiona un conjunto de fichas activas en la escena. Comprueba que, cuando todas se encuentran ocultas, debe asignar nuevas palabras desde la lista proporcionada, siempre que esta no se haya agotado.

4.4.4. Controlador de Minijuegos

La clase principal que controla todos los minijuegos es **MinijuegosControlador**. Su función es gestionar la selección y ejecución de los minijuegos durante la partida.

Mediante métodos como *ElegirMinijuego()* y *CargarMinijuego()*, la clase se encarga de escoger aleatoriamente un minijuego, siempre que no haya sido jugado previamente en la misma partida, y de cargar todos los objetos necesarios para su desarrollo, tanto en el cliente como en el host.

Asimismo, es responsable de indicar el final de la fase de minijuegos a través del método *FinalizarMinijuegoRpc()*, que notifica a los clientes que el tiempo ha finalizado y devuelve el control al flujo principal del juego.

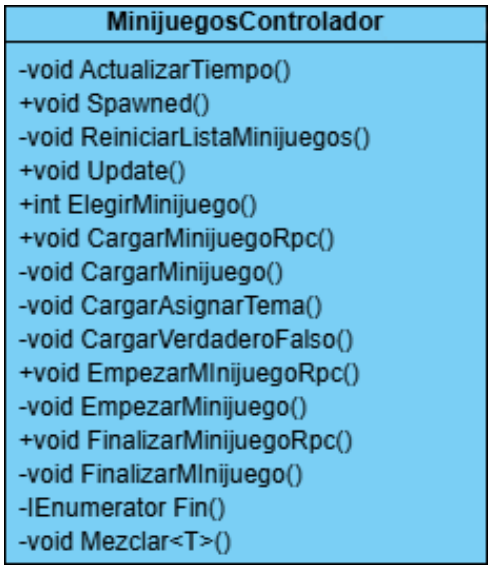


Figura 36: Diagrama de Clase MinijuegosControlador

4.5. Habilidades

Las habilidades están gestionadas principalmente por dos clases: **ManagerHabilidades** y **PlayerHabilidades**.

La fase de habilidades solo se desarrolla si el jugador activo ha decidido jugar con ellas y no se trata de la primera pregunta de la partida. Antes de cada cuestión se ejecuta esta fase. Todo comienza en el método *FaseHabilidades()* de la clase **FlujoManager**, que a su vez invoca los siguientes eventos de la clase **MinijuegosController**:

- *RepartirHabilidadesJugadores()*: busca todos los objetos correspondientes a los jugadores y, en cada uno de ellos, llama al método *AñadirHabilidadRandomRpc()* de la clase **PlayerHabilidades**. Este método añade de forma aleatoria una nueva habilidad, siempre que el jugador no posea ya tres. En partidas de un solo jugador, solo se asignan las habilidades que tienen sentido para este modo, mientras que en partidas con más jugadores, se puede añadir cualquier habilidad.

- A continuación, se activan los paneles necesarios y se muestran las habilidades almacenadas en el inventario del jugador.

- Finalmente, se inicia la fase de habilidades mediante el método *ActivarFaseHabilidadesRpc()*, habilitando que los jugadores puedan usar las habilidades recibidas o guardadas, durante un tiempo limitado.

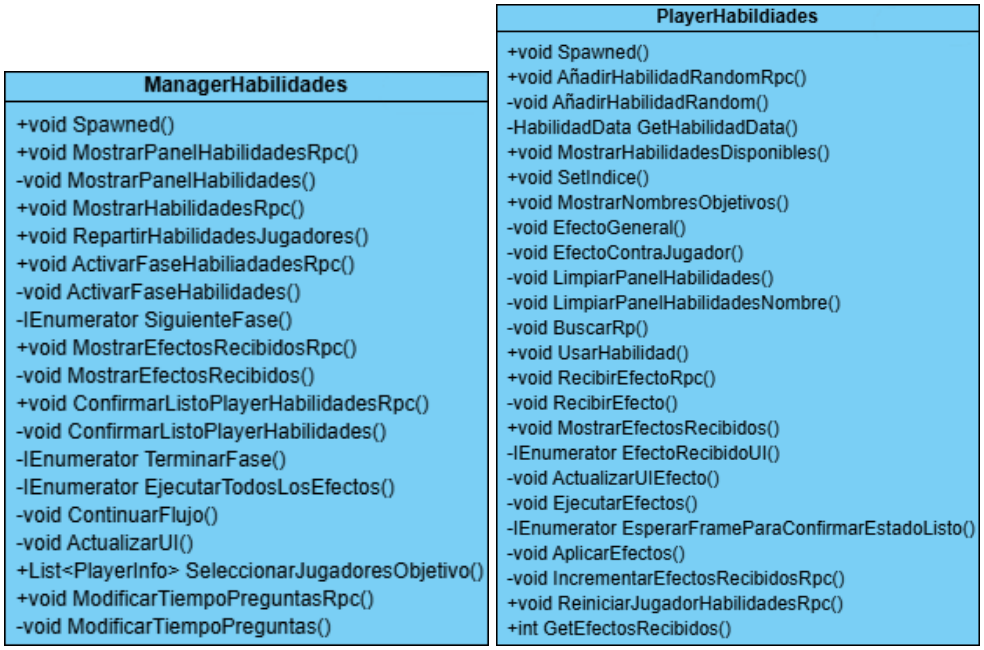
Cada habilidad del inventario tiene asociado el prefab **PrefabHabilidad**, que contiene un identificador, el nombre de la habilidad y su logo correspondiente. Cuando el jugador selecciona una habilidad, se ejecuta el método *MostrarNombresObjetivos()* de la clase **PlayerHabilidades**, que distingue entre:

- Habilidades de efecto general: muestran la opción *Todos*, lanzando el efecto de manera global.

- Habilidades dirigidas: requieren seleccionar jugadores concretos como objetivos. En este caso, se invoca el evento *SeleccionarJugadoresObjetivos()* de la clase **ManagerHabilidades**, que devuelve un conjunto de jugadores seleccionados en función de una lista de pesos. Dicha lista refleja qué jugadores han recibido más efectos a lo largo de la partida,

evitando que siempre recaigan sobre los mismos. Esta lógica está pensada especialmente para partidas con un número elevado de jugadores.

Una vez seleccionada la habilidad y su objetivo, se utiliza el método *UsarHabilidad()*, que envía la información mediante *RecibirEfectoRpc()*, indicando la habilidad utilizada y el jugador que la ejecuta. El jugador que recibe el efecto lo guarda en una lista. Cuando finaliza el tiempo de la fase de habilidades, cada cliente aplica los efectos almacenados de forma individual.



(a) ManagerHabilidades (b) PlayerHabilidades

Figura 37: Diagramas de Clases

A continuación se describen las habilidades implementadas en el proyecto.

- **Menos Tiempo**. Reduce el tiempo disponible en el flujo principal de las preguntas en una cantidad determinada.

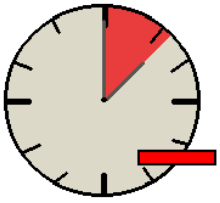


Figura 38: Logo Menos Tiempo

- **Más tiempo**. Aumenta el tiempo disponible en el flujo principal de las preguntas en una cantidad determinada.

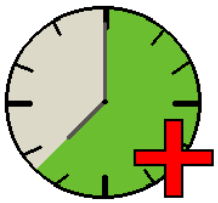


Figura 39: Logo Más Tiempo

- **Congelar**. Congela los botones de las respuestas, de modo que deben ser pulsados un número determinado de veces antes de poder seleccionarlos como opción.

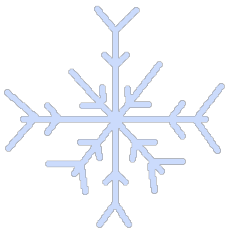


Figura 40: Logo Congelar

Figura 41: Efecto Hielo

- **Ensuciar**. Ensucia los botones de las respuestas, de modo que no puedan ser seleccionadas hasta que el botón se limpie. Este efecto se implementó utilizando el plugin **ScratchCard**. (Saietskyi, 2025)

Figura 42: Logo Ensuciar

Figura 43: Efecto Sucio

Los métodos que modifican el tiempo no pueden alterar el tiempo general más de un número determinado de veces. Esto evita que las partidas se alarguen excesivamente o que el tiempo de las preguntas pueda reducirse a cero o a valores negativos.

Además, se implementó un método en el **QuizManager** para oscurecer las respuestas durante la fase de preguntas, siempre que se haya disputado la fase de habilidades. Esta medida responde a que, si un jugador recibe un efecto negativo destinado a dificultar su rapidez al responder, no tendría sentido que pudiera ver la posición de la respuesta correcta en pantalla. De esta manera, se evita que los jugadores obtengan beneficios injustos y se aporta un elemento diferenciador a las partidas en las que se utilizan habilidades.

4.6. Preguntas Locales

Las preguntas locales o personalizadas son aquellas que los jugadores pueden agregar manualmente a la aplicación. Estas preguntas se almacenan en un archivo JSON separado, manteniendo diferenciadas las preguntas locales de las generales del sistema. Durante la partida, ambas bases de datos se combinan y mezclan, a menos que el jugador haya seleccionado la opción de jugar únicamente con sus preguntas personalizadas. En este caso, la base de datos de la partida estará formada únicamente por dichas cuestiones, siempre que exista el número mínimo necesario para jugar. Si no se alcanza dicho mínimo, se completará con preguntas de la base de datos general hasta cubrir la cantidad requerida.

El código encargado de agregar estas preguntas se encuentra en la clase **AgregarPreguntas**. Esta clase incluye el método *ValidarCampos()*, que garantiza la consistencia de la pregunta a agregar: que contenga cuatro opciones de respuesta, que solo una sea correcta, que pertenezca a una dificultad seleccionada y que el texto de la pregunta no esté vacío. Si todos los criterios se cumplen, se activa el botón de la escena asociado al método *BotonAgregar()*, el cual llama al evento *AgregarCuestion()* y, a su vez, borra los datos introducidos en la interfaz.

Además, se ofrece a los jugadores la opción de visualizar en una interfaz separada todas las preguntas que hayan agregado, así como la posibilidad de eliminarlas de la base de datos

si lo desean. Para ello, se ha desarrollado la clase **InstanciarPreguntasLocales**.

4.7. Avatares

Los avatares en *Brain Battle* aportan un elemento de personalización que permite a los jugadores diferenciarse mejor durante la partida.

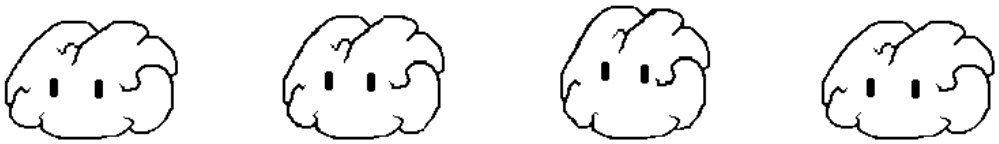


Figura 44: Animación idle del cerebro básico

Desde el inicio, se diseñaron como pequeños cerebros, con la idea de que se instancien automáticamente en la pantalla de espera del host cada vez que un jugador se conecta. Durante este tiempo, los avatares se mueven por la pantalla mientras los jugadores preparan sus perfiles, transmitiendo la sensación de que "nacen" con el jugador. Desde su cliente, cada jugador puede seleccionar una *skin* distinta para su cerebro y aplicar un color personalizado sobre el diseño base. El resto de avatares se incluyen como apéndice.

(a) Conejo (b) Científico (c) Planta (d) Punk

Figura 45: Conjunto de Avatares

Durante la partida, los avatares se muestran en las tablas de puntuaciones con su animación idle y, en los dispositivos móviles de los clientes, representan visualmente a los jugadores cuando lanzan una habilidad contra otro jugador.

Al final de la partida, el diseño del cerebro evoluciona, adquiriendo más cuerpo y reflejando un cerebro más desarrollado. Además, para las tres primeras posiciones, cada diseño de *skin* incluye una animación de victoria personalizada, reforzando la sensación de progreso y recompensa.

4.8. Audio

Para el audio en *Brain Battle* se utilizó el sistema que proporciona Unity, **Audio Mixer**. Este permite dividir el sonido del juego en varias pistas, de manera que se puedan regular los distintos volúmenes de forma independiente. Principalmente, en el juego se definieron dos pistas principales:

- **Música de fondo:** Incluye un subgrupo para los efectos sonoros que acompañan a la música.
- **Efectos de sonido del juego:** Contiene los sonidos propios de las interacciones, como los emitidos al pulsar un botón.

Se decidió que en los clientes móviles no existiera ningún sonido adicional al de los botones, con el fin de evitar un exceso de ruido en partidas con varios dispositivos reproduciendo distintos efectos o músicas. Por ello, el audio completo solo se implementa en el host principal.

La gestión del audio se realiza mediante dos clases principales:

- **AudioOpciones:** Se encarga de regular las opciones de volumen según las preferencias del jugador.
- **AudioManager:** Se encarga de reproducir los sonidos mediante el componente **Audio Source**. Un *Audio Source* es un componente que permite reproducir un clip de audio en la escena. Para que el audio pueda ser percibido correctamente, la escena debe incluir un *Audio Listener*, que actúa como receptor de los sonidos.

En cuanto a la selección de música y efectos, se estableció que durante la pantalla principal y las transiciones entre fases se reprodujeran canciones o sonidos alegres para amenizar la experiencia del jugador. Durante la fase de preguntas, su resolución y los minijuegos, se optó por pistas con mayor tensión, aunque con un ritmo suave para no distraer al usuario. Las

transiciones entre canciones se suavizaron con efectos sonoros llamativos para disimular el cambio; por ejemplo, al iniciar la fase de preguntas se reproduce un fuerte golpe de batería, y al mostrar la resolución suena un clip de arpa.

4.9. Menús Finales

En este capítulo se presentan las interfaces finales más importantes del juego, mostrando el resultado de su diseño. Se mostrarán tanto la versión del host como la del cliente, en caso de que exista una modalidad de ambas pantallas.

4.9.1. Menú Principal

Figura 46: Menú Principal Host

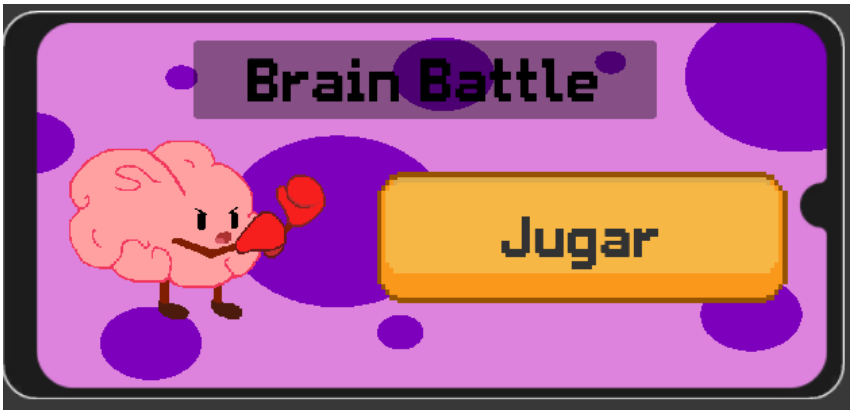


Figura 47: Menú Principal Móvil

4.9.2. Preguntas Personalizadas

Figura 48: Menú Preguntas Locales

Figura 49: Interfaz Agregar Pregunta

4.9.3. Configurar partida y jugador

Figura 50: Interfaz Configurar Partida

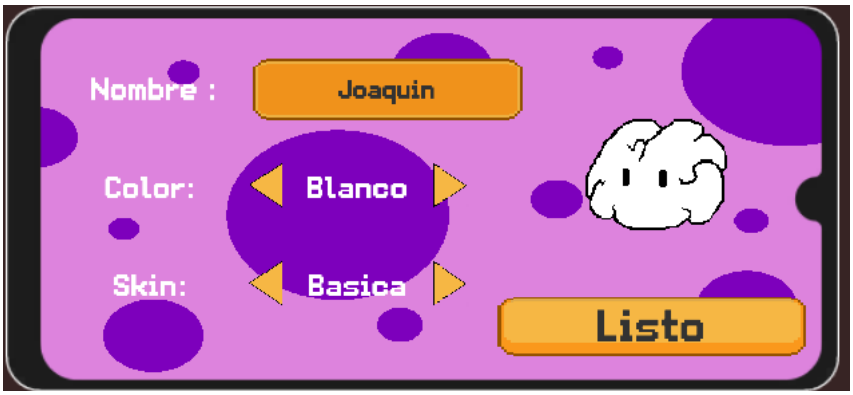


Figura 51: Interfaz Configurar Jugador

4.9.4. Interfaz principal de la partida

Figura 52: Interfaz General Presentadora

Figura 53: Interfaz Preguntas Host

Figura 54: Interfaz Preguntas Móvil

4.9.5. Habilidades

Figura 55: Interfaz Inventario Habilidades

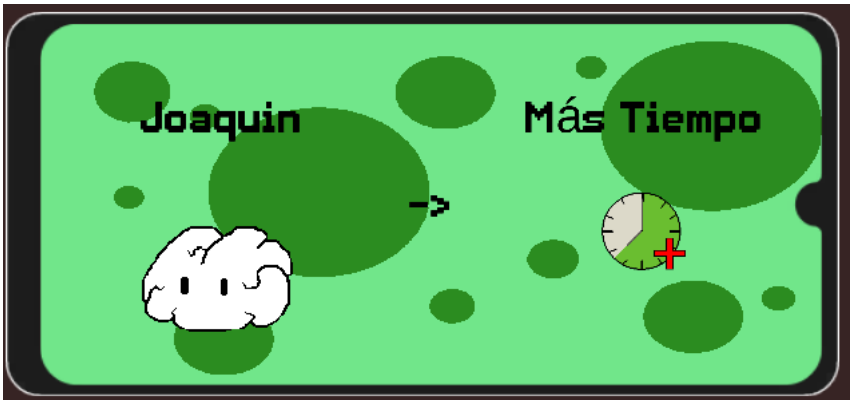


Figura 56: Interfaz Habilidades Recibidas

4.9.6. Minijuegos

Figura 57: Interfaz Tema Correcto

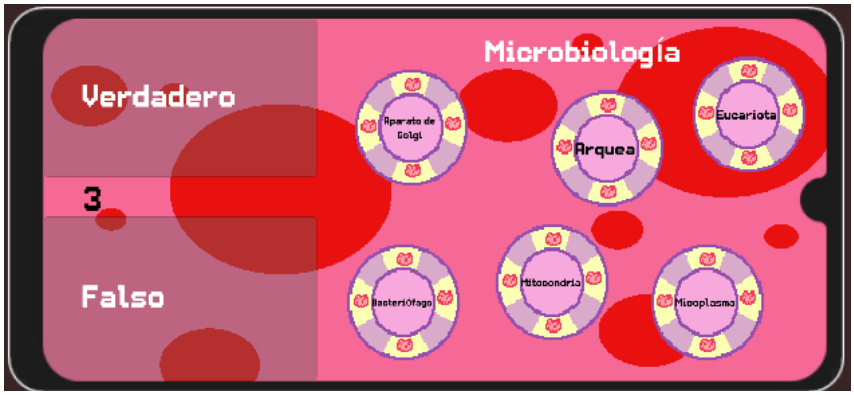


Figura 58: Interfaz Verdadero o Falso

4.9.7. Puntuaciones

Figura 59: Puntuaciones

Figura 60: Resultados Finales Host

Figura 61: Resultados Finales Móvil

4.9.8. Final de la Partida

Figura 62: Menú Final de la Partida

4.9.9. Error de conexión

Figura 63: Menú Error de Conexión

5
Pruebas

En este capítulo se presentan las pruebas realizadas sobre el sistema desarrollado para el videojuego *Brain Battle*. El objetivo principal de estas pruebas ha sido garantizar el correcto funcionamiento del juego en todas sus fases, comprobando tanto los aspectos técnicos como la experiencia de usuario.

La metodología de pruebas combinó verificaciones en un entorno controlado, donde se simularon partidas multijugador con distintas configuraciones, junto con pequeñas sesiones de prueba con usuarios reales. De esta forma, se pudieron detectar errores, comprobar la estabilidad del sistema y recoger impresiones relacionadas con la jugabilidad, la claridad de la interfaz y la usabilidad en dispositivos móviles.

5.1. Proceso de validación del sistema

Durante el desarrollo de *Brain Battle* se llevaron a cabo distintas pruebas con el objetivo de detectar y corregir errores de funcionamiento. Para ello, se diseñó una metodología específica orientada a evaluar los principales componentes del juego. Esta metodología consistió en definir los pasos a seguir en cada prueba, establecer el comportamiento esperado del sistema, ejecutar los casos de prueba, recopilar el feedback obtenido y, finalmente, comprobar si los resultados coincidían con lo previsto o si se producían errores que requerían corrección.

Las pruebas del sistema se realizaron utilizando tanto el entorno de desarrollo de Unity como un dispositivo móvil propio. En primer lugar, la aplicación del host se ejecutaba directamente en el editor de Unity en el PC. Para los clientes móviles, se empleó la herramienta **Unity Remote**, que permite ejecutar la aplicación en el propio editor de Unity mientras la interfaz del juego se visualiza y controla desde un teléfono conectado por cable. Esta opción resultó útil en las primeras fases de prueba, ya que facilitaba la depuración sin necesidad de compilar constantemente la aplicación.

En los casos en los que era necesario simular partidas con más de un cliente móvil, se optó por compilar la aplicación y ejecutarla en un teléfono propio, mientras que de forma simultánea se simulaba otro cliente directamente desde el propio entorno de Unity.

A continuación, se detallan los principales casos de prueba realizados.

5.2. Verificación sistema principal

- **Conexión de red:** Se comprueba que el servidor se crea correctamente y que los jugadores pueden unirse o desconectarse sin problemas. Además, se verifica que la partida no sufra errores en caso de que algún jugador pierda la conexión durante su desarrollo.
- **Flujo de la partida:** Se valida que el sistema avanza correctamente a través de las distintas fases del juego, sin bloqueos ni errores críticos.
- **Interacción con menús:** Se comprueba que es posible navegar por los distintos menús iniciales de forma fluida, sin quedar atrapado en estados de los que no se pueda continuar.
- **Gestión de jugadores:** Se verifica que los datos propios de cada jugador (nombre, puntuación, avatar, etc.) no se cruzan ni se intercambian con los de otros jugadores.
- **Avatares:** Se valida que los avatares seleccionados por los jugadores se muestran correctamente durante la sesión de juego y que no se modifican ni se mezclan con los de otros usuarios.
- **Selección de respuesta:** Se comprueba que los jugadores pueden escoger únicamente una respuesta por pregunta y que la fase de preguntas finaliza cuando todos los usuarios han contestado.
- **Tabla de puntuaciones:** Se verifica que las puntuaciones de los jugadores aparecen correctamente y que se actualizan de forma adecuada durante la sesión de juego.
- **Reinicio de partida:** Se comprueba que una nueva partida comienza con la configuración inicial especificada, sin arrastrar datos de partidas anteriores.
- **Sistema de audio:** Se valida que el audio permanece constante durante toda la partida, sin reproducirse en bucles infinitos ni interrumpirse inesperadamente.

5.3. Verificación minijuegos

- **Carga de minijuegos:** Se comprueba que se cargan correctamente todos los componentes necesarios para los minijuegos y que estos no se repiten durante la partida mientras queden otros minijuegos por jugar.
- **Conexión simultánea:** Se valida que todos los jugadores inician el minijuego de forma sincronizada, evitando que alguno obtenga ventaja sobre el resto.
- **Ficha:** Se comprueba que las fichas pueden moverse libremente y soltarse en los paneles correspondientes sin errores ni bloqueos.
- **Puntuación:** Se verifica que la puntuación obtenida en los minijuegos depende únicamente de las fichas colocadas correctamente y que esta se suma a la puntuación total del jugador.
- **Datos:** Se valida que la información de los minijuegos se reinicia al comienzo de cada partida, evitando arrastrar datos de partidas anteriores.

5.4. Verificación habilidades

- **Carga de habilidades:** Se comprueba que se activan correctamente los componentes necesarios para el desarrollo de esta fase.
- **Inventario:** Se valida que únicamente aparecen las habilidades nuevas generadas o las ya almacenadas en el inventario, comprobando además que el máximo de habilidades por jugador es de tres.
- **Distribución:** Se verifica que, en partidas con un solo jugador, únicamente se proporcionan las habilidades indicadas, mientras que con varios jugadores es posible repartir todas las habilidades.
- **Aplicación:** Se comprueba que las habilidades con efectos generales solo pueden lanzarse a todos los jugadores, mientras que las habilidades de efecto individual muestran la lista de jugadores como posibles objetivos.
- **Efectos:** Se prueba que los efectos aplicados y recibidos funcionan correctamente durante la partida.
- **Recuperación:** Se valida que, tras finalizar los efectos de habilidades como *Congelar* o *Ensuciar*, los botones vuelven a ser funcionales.

- **Almacenamiento:** Se comprueba que las habilidades no utilizadas permanecen guardadas en el inventario del jugador.

6

Conclusiones y Líneas Futuras

En esta sección final se presentan las principales conclusiones alcanzadas a lo largo del desarrollo del proyecto *Brain Battle*. Además, se exponen posibles vías de mejora y de ampliación del trabajo en el futuro, junto con una reflexión personal sobre el aprendizaje adquirido durante la realización de este Trabajo de Fin de Grado.

6.1. Conclusión final

El desarrollo de *Brain Battle* ha supuesto un proyecto con resultados muy satisfactorios. Se han creado dos aplicaciones completamente funcionales que cumplen los objetivos planteados inicialmente: diseñar un videojuego serio orientado a su uso en entornos educativos, ofreciendo al alumnado una manera más dinámica y entretenida de aprender.

Las pruebas planeadas con usuarios reales se llevaron a cabo de forma limitada debido al reducido tamaño de los grupos. Por este motivo, se decidió no incorporar los resultados de los análisis en la memoria, aunque las observaciones recogidas indicaron que la propuesta podría ser efectiva, permitiendo a los participantes disfrutar del juego y, al mismo tiempo, aprender contenidos relacionados con el tema central del juego. La experiencia de juego, combinada con mecánicas adicionales como minijuegos y habilidades especiales, parece tener el potencial de aportar variedad y enriquecer las partidas, favoreciendo la motivación y el interés de los participantes.

En el plano técnico, se han satisfecho los requisitos definidos en la fase de diseño. El sistema desarrollado presenta una arquitectura modular, robusta y sin fallos críticos, lo que facilita la incorporación de futuras mejoras y nuevas mecánicas. La interfaz de usuario, junto con los tutoriales diseñados, resultó clara, intuitiva y accesible, garantizando un aprendizaje sencillo

de las funcionalidades de la aplicación. Por su parte, el sistema de red se ha mostrado estable y seguro, sin incidencias relevantes en la conexión entre jugadores ni en la transmisión de datos.

En conclusión, *Brain Battle* puede considerarse un proyecto exitoso, ya que entrega un producto completo, sólido desde el punto de vista técnico y, al mismo tiempo, atractivo y divertido para los usuarios.

6.2. Líneas de mejora

Aunque *Brain Battle* constituye en la actualidad un videojuego sólido y funcional, todavía presenta un amplio margen de mejora y crecimiento. Existen diversas características que podrían implementarse en el futuro con el fin de enriquecer la experiencia de juego y ampliar sus posibilidades. Algunas de las líneas de desarrollo que podrían explorarse son:

- **Nuevos minijuegos y habilidades.** Actualmente el juego solo dispone de dos minijuegos y cuatro habilidades. Se podrían diseñar e implementar más, con el fin de aportar mayor variedad a las partidas, reducir la repetitividad y añadir un componente estratégico gracias a la introducción de nuevas habilidades.

- **Categorías temáticas en las preguntas.** En la versión actual, las preguntas se generan de manera aleatoria. Sería interesante incorporar un sistema de clasificación por temas, en el que los jugadores pudieran votar la categoría de la siguiente pregunta, aportando dinamismo y personalización a las partidas.

- **Edición y gestión avanzada de preguntas.** Además de permitir añadir preguntas personalizadas, se podría implementar un sistema de edición para corregir errores o modificar características de las ya existentes.

- **Edición de minijuegos.** Otra posible mejora consistiría en habilitar la personalización de minijuegos, de manera que los jugadores pudieran adaptar su contenido a distintos temas o contextos educativos.

- **Mejoras en la interfaz gráfica.** Actualmente el diseño artístico es básico, con fondos planos y pocos elementos visuales. Una renovación estética aportaría mayor atractivo visual y una experiencia más inmersiva.

- **Optimización del sistema de audio.** El apartado sonoro es todavía muy limitado. Sería conveniente ampliar la variedad de efectos, mejorar la gestión de pistas musicales y ofrecer transiciones más fluidas entre ellas.

- **Ampliación de los diálogos de la presentadora.** La presentadora cuenta en este momento con un guion fijo. Se podría enriquecer su comportamiento con diálogos dinámicos, aleatorios y adaptativos en función del desarrollo de la partida y las acciones de los jugadores.

- **Implementación en sistemas iOS.** Una línea futura importante es la adaptación de la aplicación tanto para dispositivos móviles con iOS como para sistemas macOS, lo que permitiría ampliar significativamente la base de usuarios.

- **Partidas multijugador online automáticas.** En la actualidad, el sistema online requiere compartir un código de sala. Se podría añadir un modo de emparejamiento automático para que los jugadores se conecten con oponentes de cualquier lugar sin necesidad de códigos previos.

- **Matchmaking por niveles.** Relacionado con lo anterior, sería posible diseñar un sistema de rangos basado en la puntuación de los jugadores, con el fin de ofrecer partidas más equilibradas y desafiantes.

- **Reconexión de jugadores.** En la versión actual, un jugador desconectado no puede volver acceder a la partida de ninguna manera. Sería recomendable implementar un sistema que permita la reconexión manteniendo sus datos y puntuación.

- **Pruebas con usuarios reales.** Durante el desarrollo de este trabajo, se llevaron a cabo pruebas con usuarios reales, aunque en grupos reducidos y controlados. En futuras investigaciones sería recomendable realizar pruebas más representativas, involucrando grupos de mayor tamaño para obtener resultados más significativos.

6.3. Aprendizaje personal

El desarrollo de este Trabajo de Fin de Grado ha supuesto un reto de gran valor tanto a nivel personal como profesional. No solo me ha permitido poner en práctica los conocimientos

adquiridos a lo largo de la carrera, sino que también me ha enfrentado a nuevos desafíos que exigieron la adquisición de competencias adicionales. Entre ellas, destacan la capacidad de gestionar un proyecto desde cero y mantener su desarrollo de forma continuada durante un largo periodo de tiempo, la organización de tareas con previsión y planificación, la resolución de imprevistos y la búsqueda constante de soluciones que garantizasen la consecución de los objetivos.

A lo largo del proceso, fue necesario investigar y aprender el uso de nuevas tecnologías, como Photon Fusion 2 o diversas herramientas de Unity, recurriendo tanto a la documentación oficial como a foros especializados en desarrollo de videojuegos. Del mismo modo, el proyecto favoreció el desarrollo de la creatividad y del pensamiento crítico, ya que surgieron numerosas ideas que debían ser evaluadas con objetividad para discernir cuáles resultaban viables en el marco temporal y de recursos disponibles.

En resumen, este trabajo me ha permitido consolidar conocimientos adquiridos en distintas asignaturas, al mismo tiempo que me enfrentaba al diseño de una arquitectura principal para dos aplicaciones y a la definición de su estructura de datos. El resultado final, *Brain Battle*, constituye un logro personal significativo, fruto de mi pasión por los videojuegos y de la satisfacción de haber tenido la oportunidad de crear uno propio. Pese a las dificultades encontradas, cada obstáculo representó una oportunidad de aprendizaje que contribuyó a mi desarrollo personal y profesional.

Como conclusión, la realización de este proyecto ha sido una de las experiencias más desafiantes y gratificantes de mi formación, y las lecciones aprendidas en el camino han reafirmado mi vocación profesional: continuar trabajando en el ámbito del desarrollo de software, y en particular, en el mundo del desarrollo de videojuegos.

Referencias

Abt, C. C. (1970). *Serious Games*. Viking Press.

Aseprite. (2023). *Aseprite - Pixel Art Tool* [Accedido: 1-01-2023]. https://www.aseprite.org/

Atlassian. (2025). *Trello – Visual Tool for Organizing Work* [Accedido: 3-07-2025]. https://trello.com/

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development* [Disponible en línea: https://agilemanifesto.org/ (Accedido: 01-12-2024)]. Agile Alliance.

Eidos Interactive. (2000). *¿Quién quiere ser millonario? (videojuego)* [Accedido: 01-07-2025]. https://es.wikipedia.org/wiki/Who_Wants_to_Be_a_Millionaire%3F_(videojuego)

Exit Games. (2025a). *Photon Engine - Multiplayer Framework* [Accedido: 3-07-2025]. https://www.photonengine.com

Exit Games. (2025b). *Photon Fusion 2 Documentation* [Accedido: 3-07-2025]. https://doc.photonengine.com/fusion/current/fusion-intro

Hasbro. (1981). *Trivial Pursuit* [Accedido: 01-07-2025]. https://es.wikipedia.org/wiki/Trivial_Pursuit

itch.io. (2023). *itch.io - Open Marketplace for Independent Game Creators* [Accedido: 1-01-2023]. https://itch.io/

Microsoft Corporation. (2022). *Visual Studio Code* [Accedido: 2022]. https://code.visualstudio.com/

OpenAI. (2023). *ChatGPT, modelo de lenguaje basado en GPT-5 mini* [Accedido: 01-03-2023]. https://openai.com/chatgpt

Overleaf. (2021). *Overleaf - Online LaTeX Editor* [Accedido: 01-01-2021]. https://www.overleaf.com/

Relentless Software. (2006). *Buzz! El Gran Reto* [Accedido: 01-07-2025]. https://es.wikipedia.org/wiki/Buzz!

Saietskyi, K. K. (2025). *Scratch Card* [Accedido: 15-05-2025]. https://assetstore.unity.com/packages/tools/particles-effects/scratch-card-228309

Unity Technologies. (2022). *Unity User Manual 2022.3 (LTS)* [Accedido: 3-03-2025]. https://docs.unity3d.com/2022.3/Documentation/Manual/UnityManual.html

Unity Technologies. (2023). *Unity Asset Store* [Accedido: 1-01-2023]. https://assetstore.unity.com/

WinMerge Development Team. (2025). *WinMerge – File Comparison Tool* [Accedido: 1-05-2025]. https://winmerge.org/

Wish Studios. (2017). *Saber es Poder* [Accedido: 01-07-2025]. https://es.wikipedia.org/wiki/Saber_es_Poder

Apéndice A
High Concept

Brain Battle

Joaquín Martín Villa

1 High Concept

Brain Battle es un juego, multijugador local, de preguntas educativas sobre biología en el que cualquier persona podrá conectarse a la partida con su propio teléfono móvil, siendo así muy accesible para jugar en clase, y demostrar cuanto sabes sobre el tema a la vez que compites con tus compañeros.

El titulo tendra un estilo visual 2D y estara desarrollado para ordeandor y movil, siendo el PC el que actuara como pantalla principal para los jugadores y el telefono como mando para cada uno de los usuarios.

1.1 Mecánicas

Las mecánica principal consiste en responder el mayor número de preguntas correctamente y lo más rápido posible frente al resto de jugadores, intentando conseguir la mayor puntuación posible. Estas preguntas estarán ajustadas a niveles de dificultad que será seleccionado antes de comenzar la sesión. Cada cuestión tendrá cuatro respuestas, de las cuales una será correcta y las otras tres falsas, el jugador tendrá un tiempo fijo para responder dicha pregunta, una vez seleccionada la respuesta, esta ya no se podrá cambiar y habrá que esperar a que todos los demás jugadores respondan o a que se consuma todo el tiempo para conocer cuál es la respuesta verdadera, obteniendo la mejor puntuación aquel que haya respondido correctamente y más rápido.

También para hacer que cada partida tenga una experiencia diferente y más dinámica, cada cierto número de preguntas, se presentara un minijuego seleccionado entre una variedad de ellos. Y como característica más destacable se podrán introducir preguntas propias, y editar los contextos de los minijuegos, haciendo más personalizable el juego a cada jugador, de esta manera el juego se puedo llevar a cualquier ámbito y no solo estar centrado en el tema de la biología.

Como una mecánica adicional se puede valorar la introducción de bonus que beneficien al jugador, por responder un número de preguntas correctas consecutivamente. Por ejemplo una ventaja podría ser, que dé las cuatro respuestas, te elimine dos incorrectas, teniendo así solo que escoger entre dos opciones.

1.2 Funcionamiento principal

Tendremos dos aplicaciones, la primera se desplegara en el ordenador, esta actuara como pantalla principal para todos los jugadores y de host para la aplicación móvil. La pantalla de título se podrá manejar con ratón y teclado para navegar a través de sus opciones, pero para iniciar una partida debera, al menos, estar conectado un teléfono móvil a través de una conexión wifi, y a partir de aquí toda interacción con el juego se realizará a través del teléfono.

Por otra parte, tenemos la aplicación móvil, la cual actuara como mando para cada uno de los jugadores. Con ella podremos manejar la app del ordenador al completo, teniendo las opciones en nuestra pantalla del teléfono móvil y viéndose reflejado nuestra interacción en la pantalla del ordenador.

Ahora vamos a explicar el funcionamiento de una partida, lo primero de todo es que para poder comenzar todos los teléfonos conectados deberán indicar que están preparados para iniciar el juego. Una vez todos los jugadores estén listos, en la pantalla del ordenador se mostrará una pregunta con sus diversas opciones, tras enseñarse esto, en los móviles aparecerán estas opciones y se deberá elegir una simplemente pulsando en la casilla deseada. Se tendrá que esperar el tiempo indicado, o a que todos los jugadores respondan, y cuando esto suceda en el ordenador se mostrará cuál era la respuesta correcta y cuáles han sido las opciones escogidas por cada jugador. Finalmente, se mostrará una tabla con las posiciones y las puntuaciones obtenidas de cada usuario, para luego repetir el proceso hasta que todas las preguntas de la partida hayan sido respondidas.

Por último especificar, que es necesario que tanto la aplicación del ordenador, como todos los teléfonos móviles que quieran conectarse a la partida, deberán estar conectados a una misma red wifi, de esta forma el juego se considera multijugador local.

1.3 Estilo gráfico

El juego presentará un estilo visual 2D, siendo su enfoque pixel art. Con una temática de concurso de televisión, como los siguientes ejemplos:

¿Quién quiere ser millonario?

Ahora Caigo

La Ruleta de la Suerte

Atrapa un Millón

1.4 Público dirigido

Este juego está diseñado para un público joven y estudiantil, ofreciendo una forma más entretenida y accesible de poner a prueba sus conocimientos sobre el tema propuesto. Aunque el público principal son estudiantes de grado en Biología, su dificultad puede ajustarse para adaptarse también a alumnos de secundaria.

Además, gracias a la mecánica que permite introducir preguntas personalizadas, el juego no se limita a este ámbito, sino que puede adaptarse a otras áreas de estudio y carreras.

1.5 Ejemplos

Referencias tomadas en consideración para este diseño, son juegos como el "Trivial" o "Preguntados".

Preguntados

Trivial Pursuit

Apéndice B
Game Design Document

Game Document Desing

Joaquín Martín Villa

1. Conceptos Básicos

- **Título:** Brain Battle
- **Género:** Party Game
- **Plataforma:** PC y sistema operativo Android. Si es posible y hay tiempo se puede extender a TV.
- **Versión:** 0.1
- **Idea y objetivo:** El propósito de este juego es ayudar a afianzar conceptos y aprender otros nuevos en el campo de la biología, a través de una serie de preguntas y minijuegos que habrá que ir resolviendo durante la partida.

  Una serie de jugadores se conectará con sus teléfonos móviles a un host principal, en el cual competirán por responder las preguntas sobre el tema designado, lo más rápido posible entre ellos. Para hacer las partidas menos monótonas también se incluirán ciertos minijuegos entre cierto número de preguntas para dar variedad.
- **Tecnología:**
  - Unity editor.
  - C
  - Visual Studio
  - Hardware propio
- **Público:** Este juego está pensado para un público joven y estudiantil, ofreciendo una forma más entretenida y accesible de poner a prueba sus conocimientos sobre la biología. Además, gracias a la mecánica que permite introducir preguntas personalizadas, el juego no se limita solo a este tema, sino que puede adaptarse a otras áreas de estudio y carreras.
- **Estilo Visual:** Su estilo artístico será principalmente 2D, con un enfoque en el pixel art. La temática estará basada en los típicos concursos de televisión, en los que predomina un color principal y su paleta suele ser muy colorida. Además tendrá una presentadora principal, el cual guiará al jugador como su tutor en el juego.

2. Jugabilidad

En este apartado explicaremos la forma en la que los jugadores pueden interactuar con el juego.

- **Periféricos:** Para este proyecto se utilizarán, teclado y ratón para el host principal, y un dispositivo móvil por cada jugador que quiera conectarse.
- **Controles:** Dependiendo de a qué dispositivo nos referimos, sus controles serán distintos. Si hablamos del host principal, este interactuará con el ratón mayoritariamente con los elementos destacados, y con el teléfono móvil cambiaremos el ratón por la interacción manual, ya sea con un simple toque o deslizamientos en la pantalla.

  El host principal, su manejo solo será necesario al comienzo de la partida para su configuración. Una vez inicia la partida no necesitaremos interactuar con él, a no ser que deseemos cerrar de manera abrupta el juego.
- **Cámara:** Durante todo el juego, la cámara será fija. Todos los componentes mostrados serán en su mayoría paneles u objetos 2D, que serán activados o desactivados en la escena, dentro del rango fijado para la cámara.

  Además, tanto en dispositivos móviles, como en el monitor principal, la cámara deberá saber escalarse bien con la resolución del dispositivo en el que se ejecuta.
- **Conexión:** La forma en la que los jugadores se conectarán a la partida es la siguiente, se deberá iniciar una sesión en el host principal, en este punto será cuando se escojan las opciones disponibles para la partida, una vez escogidas el juego quedará en espera de que cada alumno se conecte a través de su dispositivo móvil. Establecida la conexión, el jugador también deberá configurar su perfil de participante, una vez hecho confirmará su participación pulsando un botón. Cuando todos los usuarios conectados confirmen su participación, el profesor podrá desde el host principal iniciar la partida, empezando una cuenta atrás para que los alumnos sepan que se está comenzando el juego.

3. Mecánicas

- **Preguntas:** La mecánica principal consistirá, en una serie de cuestiones, las cuales irán apareciendo en la pantalla principal y los jugadores deberán responder desde sus teléfonos móviles. La respuesta correcta se revelará tanto en la pantalla principal, como en los dispositivos móviles, una vez se cumpla el tiempo mínimo que será igual para todas las preguntas, o cuando todos los usuarios conectados hayan respondido.
- **Múltiples Opciones:** Cada cuestión dispondrá de cuatro respuestas posibles, tres de ellas serán incorrectas y solo una la solución correcta.

Los jugadores podrán elegir solo una de ellas, y una vez elegida una no podrán cambiarla. La resolución solo será visible cuando el tiempo para responder la pregunta haya concluido, o todos los jugadores elijan una opción. Esta respuesta se mostrará tras una breve pausa, que tiene como finalidad generar una pequeña tensión entre los jugadores por ver si su respuesta es la correcta.

- **Respuesta más veloz:** El tiempo en responder cada cuestión, tendrá un propósito esencial, ya que los jugadores más rápidos en contestar de forma correcta, obtendrán una puntuación mayor por ello.

- **Sistema de puntuación:** En cada partida, se generará una tabla de puntuaciones, que llevara la cuenta de los puntos de cada jugador, mostrando los jugadores con los mejores resultados después de cada pregunta.

  La forma de conseguir dicha puntuación, se basará en responder de forma correcta cada cuestión, y según la rapidez de cada jugador en elegir la respuesta correcta, sumara una puntuación más alta. Los jugadores que se equivoquen escogiendo, simplemente no sumaran nada en esa pregunta. Al final de la partida se enseñará la posición de cada jugador en su dispositivo móvil y un ranking en el host final, el jugador con mejor puntuación recibirá el título de la menta más sabia.

  Un ejemplo de su funcionamiento sería:

  En una partida de cuatro jugadores, si tres de ellos responden de forma correcta, estos recibirán 10 puntos de base por ello, además el jugador más rápido sumará otros 10 puntos extras, el segundo 5 y el último solo 1 punto extra.

- **Guardado de puntuaciones:** Existirá un ranking general en el juego, que guardará las diez mejores puntuaciones de todas las partidas. Tras finalizar una sesión de juego, las calificaciones obtenidas serán comparadas con dicho ranking, y se incluirán aquellas que superen a las ya guardadas, sustituyéndolas en el sistema.

- **Perfil jugador:** Cada jugador al conectarse al juego, deberá configurar un perfil, deberá escogerse un nombre, siendo este el que se usará para la tabla de puntuaciones. Aquí puede que se incluya también una opción de color, para dotar a cada usuario de una personalización más variada y única, esto ayudaría además a identificarse en la pantalla de una manera más fácil. (Pero dependerá de cómo se integre con la interfaz del juego).

- **Minijuegos:** Durante una partida, aparte de las preguntas principales también podremos encontrarnos con algún minijuego para dar variedad a la partida. Al iniciar la partida se podrá decidir si incluir esta función o no. Los minijuegos aparecerán cada cierto número de bloques de preguntas, por ejemplo si hay 12 preguntas, aparecerán después de cada 4 cuestiones, exceptuando al final que simplemente se finalizará el juego, y la elección del minijuego será aleatoria.

- **Acciones especiales:** Las acciones especiales serán beneficios para poder responder más fácil una pregunta, o inconvenientes para penalizar a otros jugadores. Estas habilidades o acciones las recibirán los usuarios cada cierto número de preguntas. Habrá un tiempo entre preguntas para decidir si usarlas o no, si deciden usarlas aparecerá una lista de otros jugadores en los que pueden usar dichas habilidades, siempre y cuando estas sean para penalizarlos. La lista no será aleatoria completamente, ya que se tendrá en cuenta el número de jugadores que pueden recibir una penalización, para así evitar que de forma aleatoria un jugador pueda recibir más de 1 o 2 efectos negativos. Por el contrario, si el jugador decide no usar dicha acción, no se acumulará si consigue recibir otra más adelantes, simplemente esta será sustituida por la nueva.

  Esta mecánica podrá ser activada o desactivada al inicio de la partida.

- **Dificultad:** Habrá diversos niveles de dificultad, que marcarán el estilo de preguntas que aparecerán durante el juego.

- **Agregar preguntas:** Los profesores, podrán agregar sus propias preguntas al sistema de base de datos que guarda todas las cuestiones. Se deberá indicar, el grado de dificultad, la pregunta en sí y sus cuatro respuestas, marcando cuál es la correcta.

  Como configuración adicional, se podrá decidir si deben aparecer sí o sí estas preguntas personalizadas en la sesión de juego, además si existen el número necesario de preguntas, se podrá decidir si durante la partida solo se muestra exclusivamente dichas cuestiones añadidas.

- **Modificar minijuego:** De la misma forma que la anterior también se podrá modificar los minijuegos, agregando nuevas cuestiones y respuestas, según el formato que estos tengan.

4. Minijuegos

Todos los minijuegos siguen una misma estructura, mostrarán diversas preguntas escondidas en un formato más dinámico y de forma rápida e individual, además estos solo serán reproducidos en los dispositivos móviles, en el host principal no aparecerá nada. Cada jugador tendrá un tiempo específico para intentar contestar el máximo de preguntas de forma correcta, si este se equivoca nunca penalizará, simplemente se mostrará la pantalla en rojo indicando que se ha equivocado, por el contrario, si acierta la pantalla se iluminará en verde, y dependiendo del minijuego aparecerá la siguiente propuesta. Cuando el tiempo se agota se sumará la puntuación según el número de respuestas correctas que haya conseguido cada jugador.

Estos minijuegos aparecen cada cierto número de preguntas, y antes de comenzar se explicará como es su jugabilidad, para que todos los jugadores la conozcan.

A continuación se explican los diversos minijuegos:

- **Tema Correcto -** Tendremos a cada lado de la pantalla dos temas diferentes, y en el centro aparecerá una ficha con un elemento relacionado con uno de estos dos temas. El jugador deberá relacionar la ficha con el tema que crea que le corresponda, esto lo hará arrastrando dicho elemento a unos de los lados de la pantalla que estará representado con un área de color diferenciado, para poder distinguir cada tema y donde puede soltarla. Ya falle o acierte, volverá a aparecer una nueva ficha.

- **Verdadero o Falso -** Aparecerán diversas fichas en pantalla, estas serán frases que pueden ser ciertas o no, el jugador deberá indicar cuál cree que es la respuesta arrastrando la ficha al campo correspondiente. Una vez pasado el tiempo se comprobarán cuáles han sido acertadas y cuáles no. Estas fichas, mientras el minijuego este en marcha, pueden ser cambiadas de bando todas las veces que se crea necesario. El juego intentará sacar siempre un equilibrio entre las frases para que no todas sean falsas ni verdaderas.

- **Conecta Elementos -** La pantalla estará dividida en dos partes, tendremos a un lado diversas fichas que representa varios elementos distintos entre sí, y al otro lado una única ficha, que representa un elemento relacionado con solo una de las otras fichas. El jugador deberá intentar relacionar esta ficha con la índica de las otras, esto lo hará tocando una ficha de un lado, y arrastrando el dedo, se generará una línea que deberá conectar con una ficha del otro lado.

  Si ambas están relacionadas, estas dos desaparecerán y serán reemplazadas por otras, si se equivoca el jugador, la línea será borrada y deberá intentarlo otra vez.

- **Completa la frase -** Como su nombre indica, irán apareciendo en pantalla frases que el jugador deberá completar con las fichas que saldrán en la parte inferior de la pantalla del móvil. Tendrá que arrastrar la respuesta que crea correcta al lado superior de la pantalla.

5. Acciones Especiales

Los nombres deberán ser indicativos de lo que hacen, para que el jugador con una simple lectura comprenda cuáles son su función.

- **50/50 -** De las cuatro opciones que se muestran normalmente, esta habilidad eliminara dos incorrectas, dejando al jugador con un 50 % de probabilidades de acertar o fallar.

- **Puntuación Doble -** El jugador que se beneficie de esta acción, multiplicara por dos los puntos obtenidos en la siguiente pregunta.

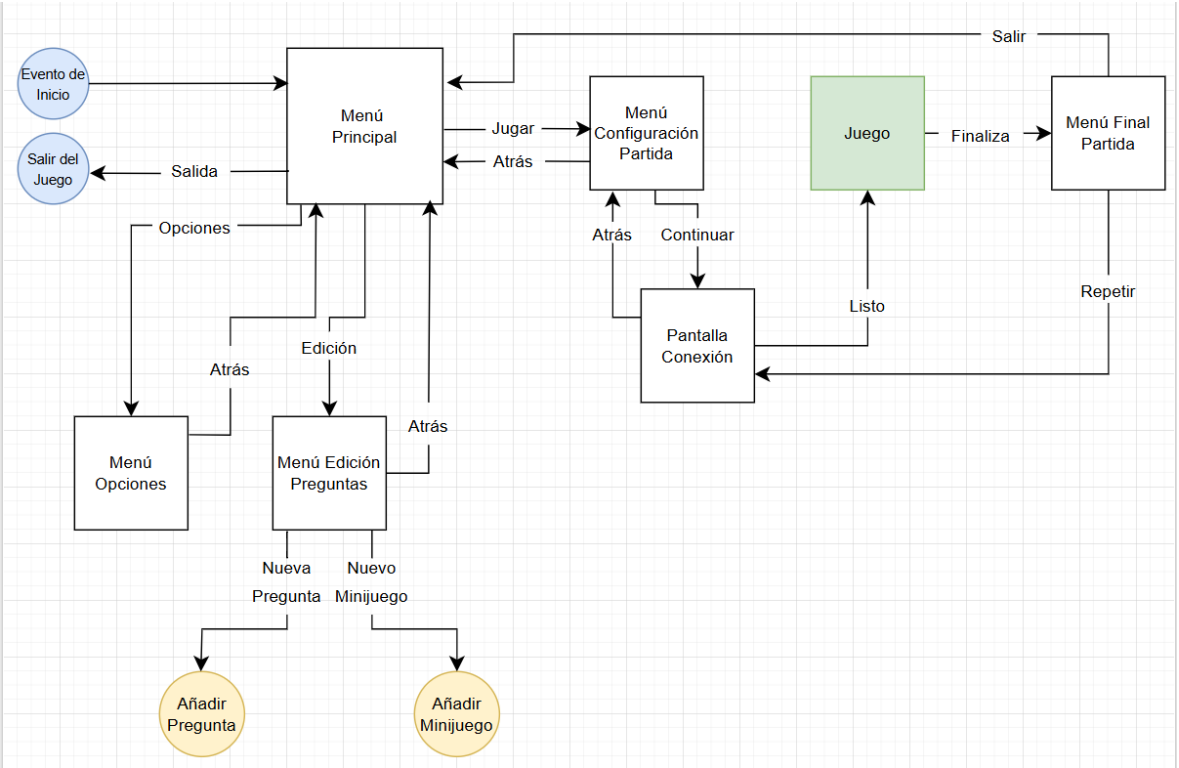
- **Más tiempo -** Esta es una habilidad que afectara a nivel general, añadiendo más tiempo de respuesta a las preguntas.
- **Congelar -** Congelará la pantalla de un jugador escogido, este deberá tocar varias veces la pantalla para poder escoger una de las opciones de respuesta.
- **Ensuciar -** Ensuciará la pantalla de un jugador escogido, dicho jugador deberá limpiar la pantalla pasando el dedo por su dispositivo móvil, creando un efecto de que está pasándole una servilleta.
- **Encadenar Respuestas -** Las opciones de respuesta tendrán uno o varios candados, que el jugador escogido deberá quitar simplemente tocándolos.
- **Oscurecer Respuestas -** Cada una de las respuestas aparecerán en negro, debiendo tocar el jugador dicha respuesta para que aparezca.
- **Menos tiempo -** Esta habilidad afectará a nivel general, quitando tiempo de respuesta para todos los jugadores.

Cabe indicar que los efectos generales de tiempo solo se podrán aplicar a la vez un número determinado de veces, que se decidirá en la producción según el nivelado que se vea necesario.

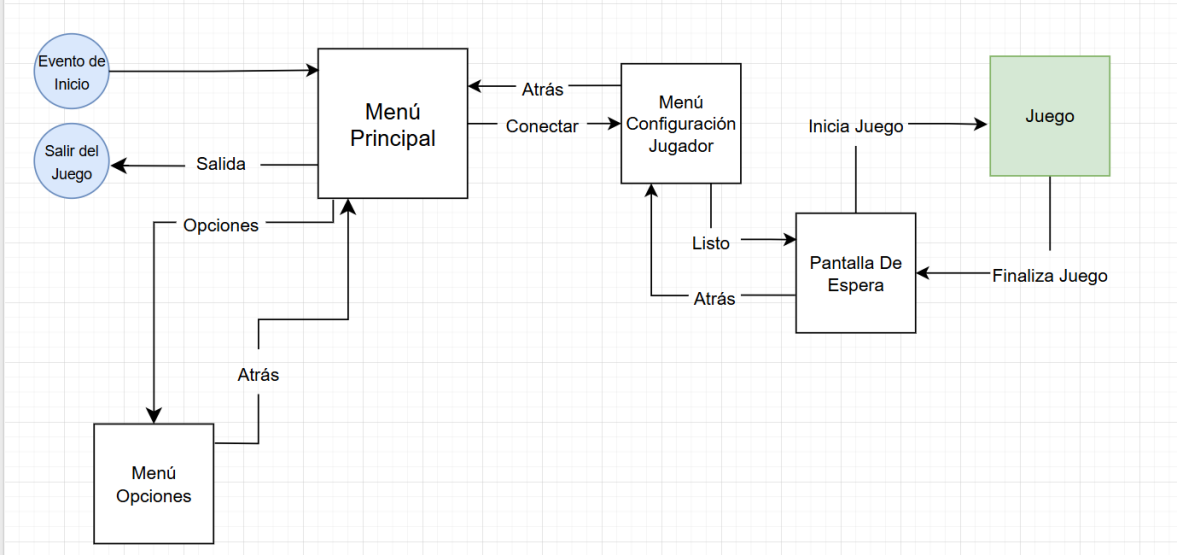
Antes de cada pregunta, los jugadores tendrán un tiempo específico para decidir si utilizar o no su habilidad, una vez concluido este tiempo, en la pantalla principal se indicara que acción se ha utilizado, quien la ha utilizado y contra quien en caso de que sea una acción lanzada a otro jugador.

6. Flujo de Pantallas

6.1. Host Principal



6.2. Dispositivo Móvil



7. Interfaces

En este apartado vamos a comentar las diversas interfaces y menús que existirán en el juego. Cabe destacar que su diseño grafico está sujeto a cambios, ya que falta su desarrollo, por ende lo que vamos a ver son el diseño de las funciones básicas que tendrá cada menú.

7.1. Host Principal

- **Menú Principal**

Jugar → Inicializará la partida.

Opciones → Abrirá el menú opciones.

Salir → Cerrará la aplicación.

+ → Abrirá el menú para añadir una nueva pregunta o editar un minijueg

- **Menú Opciones**

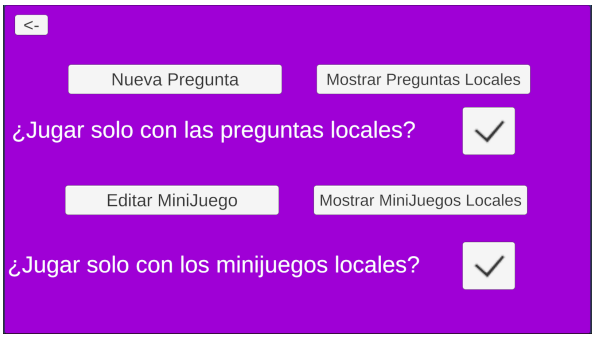
Sliders → Habrá tres sliders que controlen todo el volumen del juego, estos serán uno general que actuara como máster, uno específico para la música que sonara de fondo y por último también otro especifica solo para los efectos de sonido que puede haber durante el juego.

Créditos → Activará una pantalla con los créditos del desarrollo.

Tutorial → Aquí el jugador podrá habilitar o deshabilitar el tutorial, si está activado esto implicara que al inicio del juego se explicara como funciona este mismo y cada vez que haya un minijuego este también tendrá una breve explicación inicial. Si el tutorial está deshabilitado estos momentos no sucederán.

Flecha → Volver al menú principal.

- **Menú Edición Preguntas**



Nueva Pregunta → Abrirá el menú para agregar una nueva pregunta a la base de datos.

Mostrar Preguntas Locales → Activará una nueva ventana, donde se podrá ver todas las preguntas que han sido agregadas en este ordenador.

¿Jugar solo con las preguntas locales? → Con la opción activada, se intentará que todas las preguntas que aparecen en el juego sean extraídas de las añadidas por el jugador. Si no son las suficientes, estas deberán aparecer mezcladas con preguntas de la base de datos general.

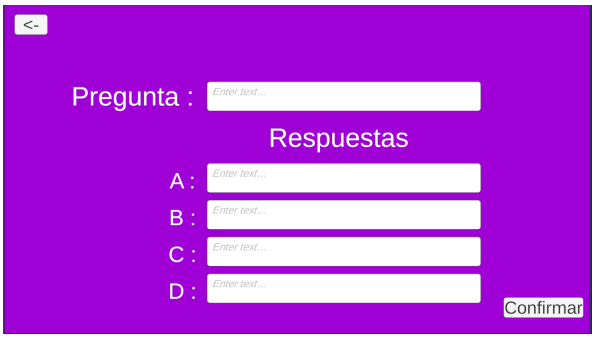
Editar Minijuego → Abrirá el menú correspondiente para seleccionar que formato de minijuego quieres agregar.

Mostrar Minijuegos Locales → Nos mostrará una ventana, con los minijuegos que han sido modificados en este ordenador.

¿Jugar solo con los minijuegos locales? → De nuevo tenemos una casilla, para decidir si en nuestra sesión de juego, los minijuegos que aparezcan serán extraídos de nuestra base de datos local o general.

Flecha → Volver al menú principal.

- **Añadir Pregunta**



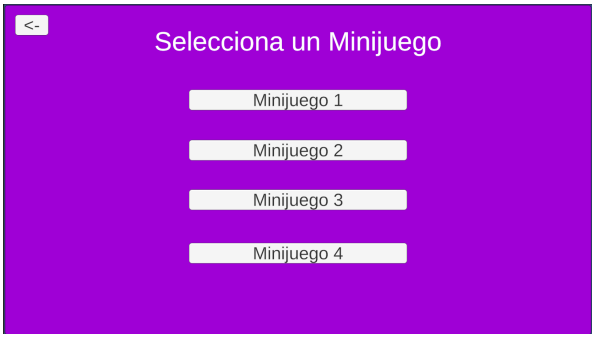
Pregunta (Input Field) → Introducir la pregunta que se desea añadir.

Respuestas (Input Field) → Tendremos cuatro inputs fields para añadir las respuestas a la pregunta, con un checkbox deberemos indicar cuál dé las respuestas es la correcta, solo pudiendo existir una solución válida.

Confirmar → Al pulsar dicho botón, se comprobará que todos los campos estén rellenos y que las condiciones dadas sean válidas. Si todo esto es así se añadirá la pregunta.

Flecha → Volver al menú de edición de preguntas.

- **Añadir Minijuego**



Seleccionar un minijuego → El jugador podrá escoger que estilo de minijuego quiere modificar. Llevándolo a la interfaz correspondiente para cada minijuego.

Flecha → Volver al menú de edición de preguntas.

Tanto la interfaz de edición de cada minijuego, como los menús para ver las preguntas y minijuegos locales todavía no han sido definidos.

- **Menú Configuración Partida**

Dificultad → Se podrá ajustar el nivel de dificultad para la partida, siendo estas opciones, primaria, secundaria y universidad.

Número de Preguntas → Tendremos varias opciones para establecer un número de preguntas. Estas opciones están por definir.

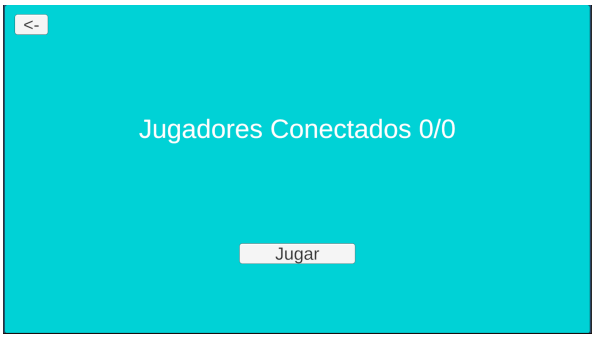
Minijuegos → Habilitar o deshabilitar la partida con minijuegos.

Acciones Especiales → Habilitar o deshabilitar la partida con acciones especiales.

Siguiente → Se procederá a iniciar el servidor para que los jugadores se vayan conectando con sus dispositivos móviles a la partida, con las especificaciones indicadas para esta misma.

Flecha → Volver al menú principal.

- **Pantalla de conexion**



Jugadores Conectados → En esta pantalla, el servidor ya estará montada e iniciado, esperando a la conexión de los jugadores a través de sus dispositivos móviles. En la pantalla se irá indicando de forma dinámica los jugadores que se irán conectado, siendo este el segundo número del 0/0, y el primer número representaran los jugadores que han pulsado el botón de listo y por ende están preparados para el juego.

Jugar → Iniciará la partida.

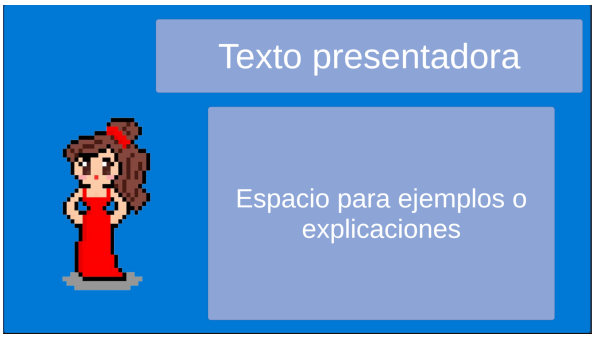
Flecha → Volver al menú de configuración de partida.

- **Menú Final**

Si → Volverá a la pantalla de conexión. Manteniendo el servidor encendido y como los mismos jugadores conectados.

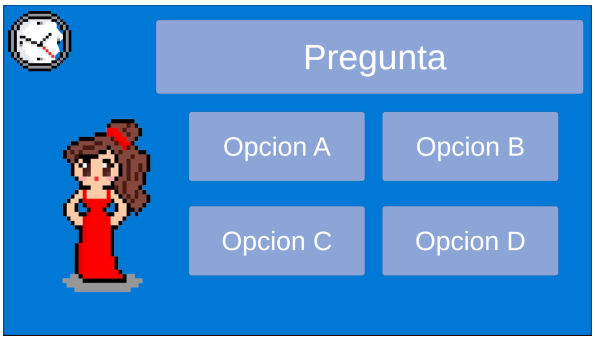
No → Desconectará el servidor y expulsara a todos los jugadores, además volverá al menú principal.

- **Interfaz Juego**



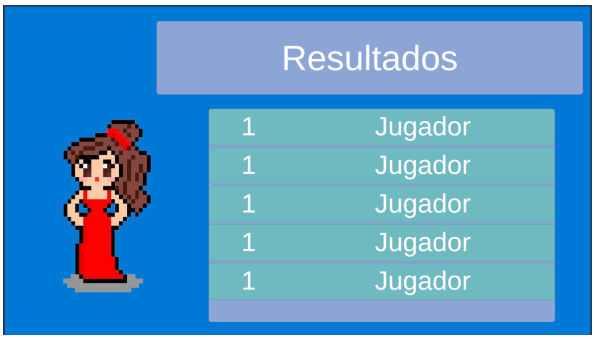
Definición → Esta será la interfaz más básica durante la partida, tendremos una presentadora que dirigirá al público a través de la partida, realizando comentarios o explicando las diversas situaciones que pueden surgir. Para ello está especificado un lugar para su texto, que será como una especie de bocadillo de comic, y un espacio especificado para mostrar ejemplos u otras cosas.

- **Interfaz de las Preguntas**



Definición → En esta interfaz se muestra la pregunta y sus diversas respuestas. En la esquina superior izquierda tenemos un reloj, que vendrá a representar el tiempo restante para responder. La presentadora será la que anuncie la pregunta para el público.

- **Interfaz Resultados**

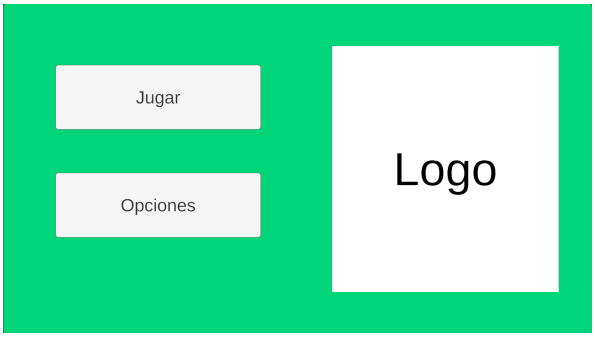


Definición → Se mostrarán las posiciones más altas en la tabla de resultados, especificando la posición, el nombre y los puntos del jugador.

7.2. Dispositivo Móvil

La interfaz para el dispositivo móvil está pensado que sea en horizontal.

- **Menú Principal**



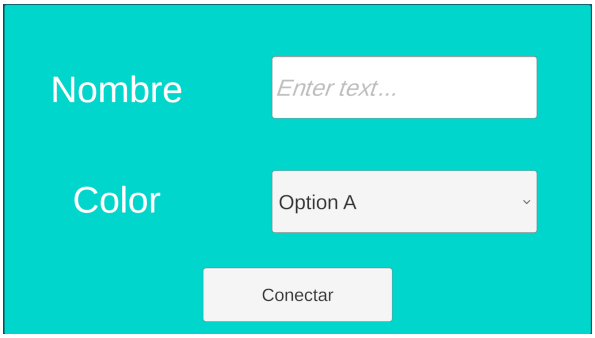
Jugar → Activa el menú de configuración del jugador.

Opciones → Activa el menú de opciones.

- **Menú Opciones**

Sonido → Se podrá regular el sonido de tres formas, con un slider general que afectara a cualquier tipo de sonido del juego, otro para la música de fondo, y un último para los efectos de sonido del juego.

- **Menú Configuración Jugador**



Nombre → El jugador deberá especificar un nombre propio, para mostrar durante la partida.

Color → Se podrá escoger entre varios colores, para así dar más personalización a los jugadores, y encontrar de una forma más visual su nombre en la tabla de resultados.

Conectar → Pasará a la pantalla de conexión.

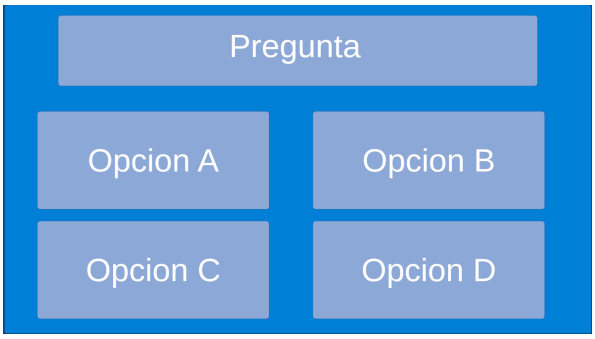
- **Pantalla de Conexión**

  Esta interfaz tiene dos partes, la primera es la interfaz de intento de conexión, en la cual el dispositivo móvil intentara conectarse al servidor a través de la conexión wifi local. Si esta conexión falla o tiene exceso de tiempo, deberá indicárselo al jugador con la opción de volver a reintentarlo.

  La segunda parte es la interfaz una vez conseguido establecer conexión.

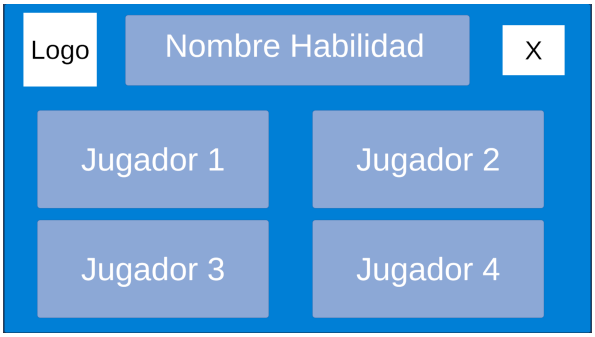
  Listo → Este botón activará un símbolo visual que le indica al jugador de manera clara que está preparado para iniciar la partida, también podrá volver a pulsar el botón para indicar que deja de estar preparado. La partida no podrá comenzar hasta que se pulse este botón.

- **Interfaz de las Preguntas**



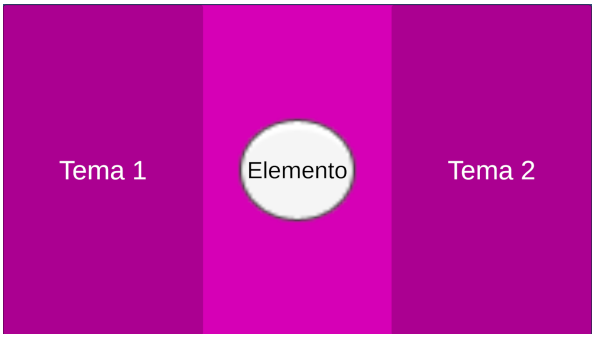
Definición → Aparecerá la pregunta con sus diversas opciones. Si queda poco tiempo y el jugador todavía no ha pulsado una respuesta, la pantalla parpadeará. Cuando el jugador haya tomado una respuesta, esta interfaz desaparecerá y se mostrará una nueva señalando que ya se ha respondido.

- **Interfaz Acciones Especiales**

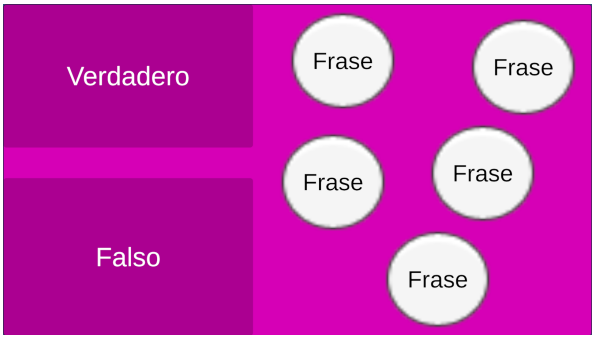


Definición → Tendremos en la parte superior, el logo de la habilidad, el nombre, y un botón con el cual podemos decidir no usar en este momento dicha acción. En el resto de la pantalla tendremos, los nombres de los diversos jugadores a los que podemos lanzar la acción, o en caso de que sea una acción que solo nos podemos aplicar a nosotros mismos, solo aparecerá nuestro nombre.

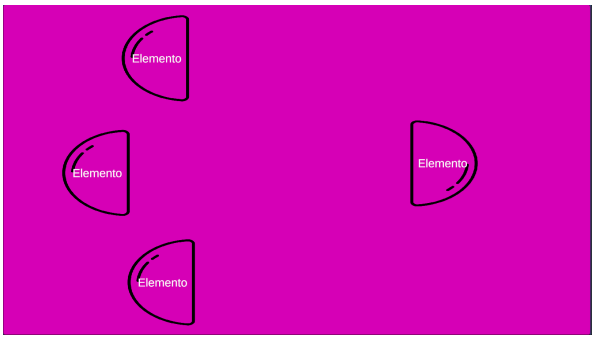
- **Minijuego. Tema Correcto**



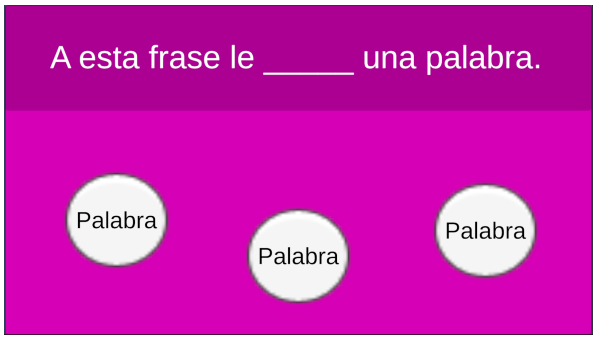
- **Minijuego. Verdadero o Falso**



- **Minijuego. Conecta Elementos**

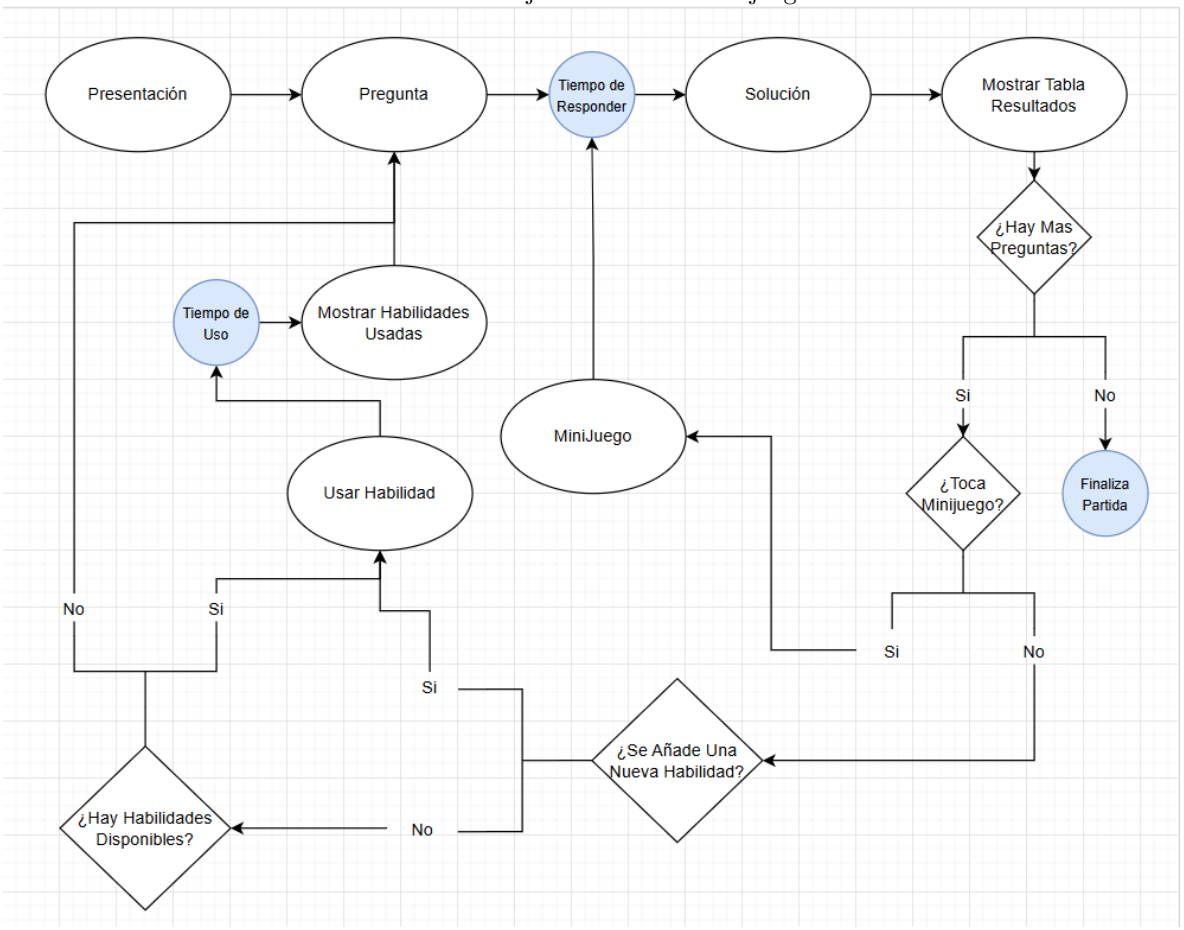


- **Minijuego. Completa la Frase**



8. Flujo del Juego

A continuación mostraremos como es el flujo de una sesión de juego.



Cuando la conexión de un usuario falle durante este flujo, se le indicará con una pantalla que ha sido desconectado. En el momento que consiga recuperar la conexión, deberá esperar a la transición del siguiente estado para poder reconectarse a la partida.

9. Presentadora

El juego contendrá una presentadora, que guiara al jugador en cada sesión de juego, presentado y formulando las preguntas para los jugadores, enseñando la tabla de resultados, etc.

Si los tutoriales están activados, además este personaje explicará al comienzo cuáles son las mecánicas y su funcionamiento, y cada vez que toque un minijuego, ella también explicara como funciona este mismo.

Tendrá principalmente dos animaciones, una de idle y otra hablando, que se irán turnando según el personaje hable o espera a la interacción de los jugadores.

10. Música y Sonidos

La música se dividirá en dos grupos, música de fondo y efectos de sonido. La música de fondo, será un conjunto de temas, con temática alegre, que irán sonando de forma aleatoria durante la partida. Los efectos de sonido, se compondrán, de por ejemplo un sonido al pulsar un botón, otro cuando el jugador responda correctamente una pregunta o cuando la falle, los textos de la presentadora tendrán un efecto para indicar que está hablando y otras ideas o necesidades que vayan surgiendo durante el proyecto.

Apéndice C
Diseño de Avatares

1 Avatar Basico

Figure 1: Basico

2 Diseño de avatares

(a) Conejo

(b) Científico

(c) Lazo

(d) Fiesta

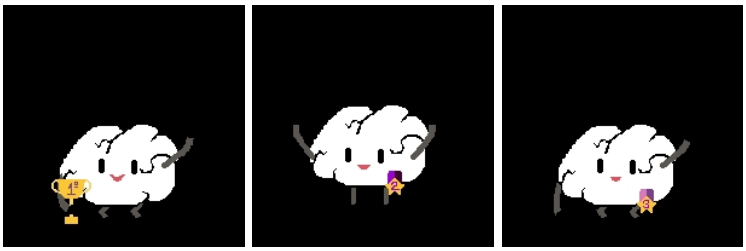
Figure 2: Conjunto de Avatares

(a) Malvado (b) Planta

(c) Rockero (d) Zombie

Figure 3: Conjunto de Avatares

3 Avatares Resultados Finales

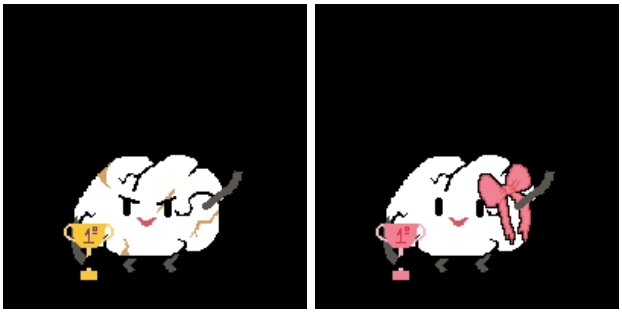


(a) Primer Puesto (b) Segundo Puesto (c) Tercer Puesto

Figure 4: Avatar Basico

(a) Primer Puesto (b) Segundo Puesto (c) Tercer Puesto

Figure 5: Avatar Planta



(a) Malvado (b) Lazo

(c) Cientifico (d) Zombie

Figure 6: Avatares Victoria

(a) Fiesta

(b) Conejo

(c) Rockero

Figure 7: Avatares Victoria

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA