UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN Y EVALUACIÓN DE USABILIDAD DE UN VIDEOJUEGO ONLINE PARA EL APRENDIZAJE DE LA ARMONÍA MUSICAL

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

CARLOS PATRICIO MENESES RODRÍGUEZ

PROFESOR GUÍA:
JAIME SÁNCHEZ ILABACA

MIEMBROS DE LA COMISIÓN:
LUCIANO RADRIGAN FIGUEROA
JOCELYN SIMMONDS WAGEMANN

SANTIAGO DE CHILE
2022

Resumen

La armonía musical es un tema importante dentro de la teoría musical. Pero también es difícil de aprender. Al punto de poder volverse una barrera en la educación musical de los estudiantes con baja motivación. Esto le da valor a poseer herramientas digitales que faciliten su aprendizaje. Por otra parte, una de las formas de facilitar la enseñanza y aprendizaje de un contenido es a través del uso de videojuegos educativos.

Estas dos situaciones juntas hacen que sea valioso que existan juegos educativos dirigidos a enseñar armonía musical. Pero en la actualidad existen pocos videojuegos disponibles que sirvan para enseñar armonía musical. En especial, a nivel de enseñanza y aprendizaje universitario.

Bajo estos supuestos, en el año 2020 la estudiante Daniela Medel diseñó, utilizando material concreto, un juego de puzle multijugador que puede ser jugado como juego de mesa llamado Tone Cluster. Diseño creado con el objetivo de que sus jugadores pudieran aprender armonía musical a través de su uso. En ese contexto, ella diseñó las interfaces usuarias gráficas que pudieran usarse en el futuro para implementar Tone Cluster como una aplicación de celular.

El juego Tone Cluster es un juego de puzle multijugador similar a Scrabble, en el cual los jugadores toman turnos para ubicar fichas que representan notas de manera de crear acordes con los que obtener puntos.

Este trabajo de título consiste en un rediseño digital de ese diseño previo del juego e interfaces gráficas, para posteriormente implementar el videojuego y finalmente evaluar la usabilidad del juego Tone Cluster para dispositivos celulares.

La aplicación de celular desarrollada permite jugar partidas de multijugador asíncrono del juego Tone Cluster. Gracias a que el juego es asíncrono, los jugadores no necesitan estar conectados al mismo tiempo para jugar las partidas. Y como el juego es para celular, pueden realizar jugadas desde cualquier lugar.

Se realizó una prueba de evaluación heurística con tres expertos y una prueba de usabilidad con diez usuarios, 9 de los 10 usuarios encontraron la aplicación buena o excelente. Los tres expertos calificaron globalmente los criterios de usabilidad de la aplicación como neutros, buenos o excelentes.

Agradecimientos

Agradezco a Daniela Medel, la diseñadora del juego y sus interfaces, quien siempre dio su apoyo para que este proyecto funcionara, además de apoyar la evaluación de usabilidad.

Este trabajo de Tesis se enmarca en el contexto del Fondo Basal para Centros de Excelencia, Proyecto FB0003, PIA-CONICYT, del Centro de Investigación Avanzada en Educación, CIAE, Universidad de Chile.

Tabla de contenido

1. Introducción

La armonía trata del estudio de los acordes y su relación entre sí. La comprensión de la armonía musical es esencial para la comprensión del lenguaje de la música [2]. Pero también es difícil de aprender. Y esta dificultad se puede convertir en una barrera en el aprendizaje musical de estudiantes con baja motivación [1].

Esto es debido a que el aprendizaje está en su mejor estado cuando se alcanza un estado de Flow [4], en el cual el estudiante puede comprometerse al estudio a pesar del desafío que implique. Para alcanzar este estado se requiere que la actividad que se realice sea estimulante y esté acorde a las habilidades del estudiante [7]. Pero el estudio formal de la armonía musical, desconectado del uso de tecnologías actuales [3], llega a ser tedioso y difícil. En vez siendo frustrante para los estudiantes.

En los últimos años ha habido un gran desarrollo en múltiples áreas que benefician el desarrollo de videojuegos educativos. Como el desarrollo de los dispositivos móviles, la comunicación permanente con otros usuarios y el mercado de aplicaciones. Sin embargo, un área del conocimiento donde su uso ha sido limitado es para la enseñanza de la armonía musical.

Con esta motivación, en el año 2020 la estudiante Daniela Medel Sierralta se tituló con el grado de Magíster [1] con una tesis en la que propuso el diseño de un prototipo de videojuego online para el aprendizaje de la armonía musical.

El objetivo de la investigación de la tesis en la que se propuso el juego diseñado fue promover una mayor conexión con la motivación intrínseca y el aprendizaje colaborativo para lograr los contenidos de aprendizaje, junto con propiciar la autonomía del estudiante y la colaboración de grupo como principal fuente de aprendizaje.

Para continuar con el objetivo de la enseñanza de armonía musical, en esta memoria se rediseñó, implementó y evaluó el juego de Tone Cluster. El desarrollo de la aplicación siguió la metodología ágil del Rapid Prototyping qué sigue un proceso iterativo de implementación en el que cada iteración fue evaluada con la diseñadora del juego y el profesor guía.

En esta memoria se implementó el videojuego propuesto como diseño en dicha investigación para dispositivos móviles y utilizando tecnologías de multijugador en línea. Para esto, se implementó un lado de cliente y un lado de servidor.

El lado de cliente es una aplicación para celulares desarrollada en Unity. Desde la aplicación desarrollada los usuarios pueden crear e ingresar a sus cuentas, agregar amigos, crear nuevas partidas con sus amigos y jugar en las partidas.

El lado del servidor utiliza la plataforma de Backend como Servicio PlayFab. Desde aquí se almacena la información de las cuentas de los usuarios, se autentican sus ingresos, se guardan las listas de amigos de los jugadores y se guardan sus partidas.

1.1. Objetivo General

El objetivo general de este trabajo de título es desarrollar y evaluar la usabilidad de un videojuego para dispositivos móviles utilizando tecnologías de multijugador en línea, para enseñar y aprender armonía musical, de manera que atraiga e involucre a los jugadores de mejor manera que los métodos actuales.

1.2. Objetivos Específicos

1. Rediseñar un juego para aprender y enseñar armonía musical, a partir del diseño original.
2. Implementar un videojuego para dispositivos móviles utilizando tecnologías de multijugador en línea, para aprender y enseñar armonía musical.
3. Evaluar la usabilidad del videojuego con usuarios finales y expertos con la finalidad de recibir retroalimentación sobre la aceptación de la aplicación y posibles mejoras de diseño.

1.3. Resumen de la Solución Desarrollada

En este trabajo de título se hizo un rediseño, implementación y evaluación del juego Tone Cluster. El cual fue diseñado con el objetivo de enseñar armonía musical.

La solución de software implementada es una aplicación de celular, donde los usuarios pueden crear cuentas y organizar grupos con amigos, para luego jugar partidas del juego musical, de tablero y fichas, Tone Cluster. En el cual los jugadores toman turnos para ubicar fichas que representan notas sobre un tablero con el objetivo de crear acordes y ganar puntos.

Además, para poder organizar a los jugadores y las partidas, se levantó un lado de servidor que le permite a los jugadores mantener partidas de multijugador de manera asíncrona y jugar por turnos sin requerir presencia simultanea en la aplicación de los jugadores.

La usabilidad del videojuego fue evaluada tanto por alumnos que estudian armonía musical como por profesores expertos en educación musical.

2. Marco Teórico

2.1. Aprendizaje a Distancia

La pedagogía de la educación a distancia se puede separar en tres generaciones. La pedagogía Cognitiva-Conductista, la Social-Constructivista y la Conectivista. En la pedagogía Cognitiva-Conductista, el estudiante es un receptor pasivo del conocimiento que se le entrega. En la pedagogía Social-Constructivista, en cambio, el estudiante es un agente activo de su propio aprendizaje [5]. Por último, en la pedagogía conectivista se entiende el aprendizaje como el proceso de construcción de redes de información, contactos y recursos que se aplican a problemas reales. Con foco en la capacidad de obtener y aplicar conocimiento por sobre la memorización.

El juego planteado en este estudio está en la categoría del aprendizaje Social-Constructivista, donde el aprendizaje es aplicado, en este caso, en un espacio de juego. Este espacio es compartido con otras personas. Con ello, se espera que la actividad social del usuario unida al uso de la aplicación facilite un aprendizaje significativo.

2.2. Videojuegos Educativos

El videojuego que se desarrolló en esta memoria es un juego educativo hecho para que lo usen estudiantes usando dispositivos móviles. Ejemplos de juegos educativos para dispositivos móviles incluyen los juegos Evolución, BuinZoo y Museo. Estos juegos serios fueron desarrollados para ser usados en el contexto específico de las actividades curriculares de cursos de enseñanza básica. [9]

Figura 1: Evolución

Estos juegos fueron usados con grupos de estudiantes de octavo básico en actividades de aprendizaje durante las cuales fueron al BuinZoo y al Museo Nacional de Historia Natural. Durante estas actividades, los estudiantes fueron divididos en grupos de cuatro y puestos a

resolver las preguntas planteadas en los juegos. Algunas preguntas se responden de manera individual y otras de manera grupal.

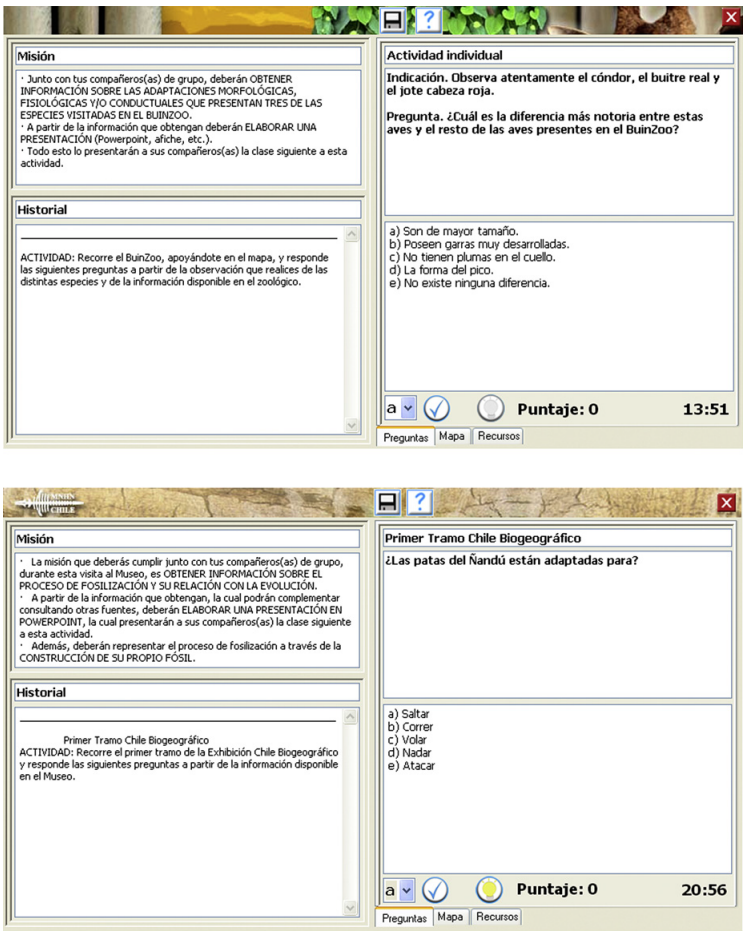


Figura 2: BuinZoo y Museo

Tras estas actividades, los estudiantes tuvieron que responder una encuesta y unas escalas de percepción de habilidades de resolución de problemas y habilidades de colaboración. Los datos obtenidos tras esta investigación apuntan a que el grupo experimental alcanzó una mayor percepción de sus habilidades de colaboración.

2.3. Juegos Educativos de Música

Un ejemplo de juego dirigido a enseñar armonía musical es In Harmony [10]. El cual es un programa educativo dirigido a niños de educación primaria que incorpora dos programas de software de educación musical: Teach, Learn, Evaluate! (TLE) [11] y Impromptu [12]

TLE consiste en una serie de cuatro ejercicios de actividades musicales. Impromptu es una aplicación en la cual el usuario puede organizar fragmentos de melodías conocidas representados en bloques para reconstruirlas o crear melodías nuevas.

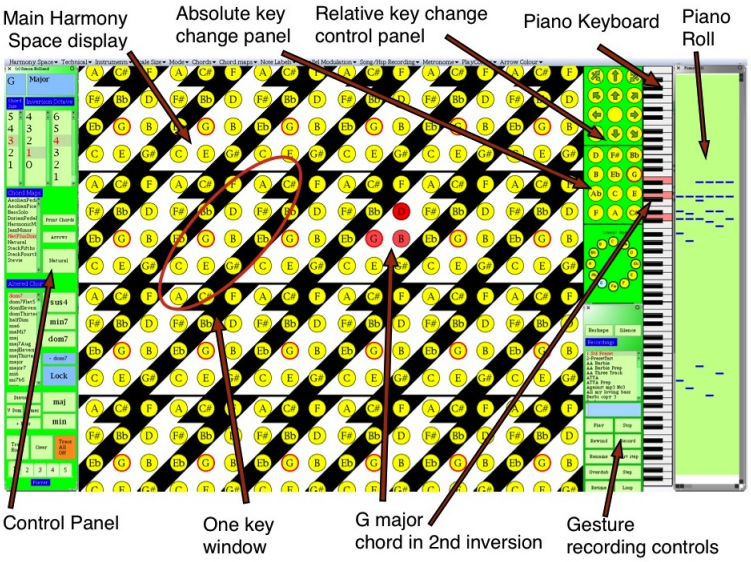


Figura 3: Harmony Space

2.4. Herramientas Educativas de Armonía Musical

Herramientas hechas con el objetivo de enseñar aspectos de la armonía musical incluyen Harmony Space [13], Harmonic Walk [14] y los *playgrounds* MusEdLab [15] Math, Science & Music [16] y Music Lab [17].

Harmony Space es una interfaz diseñada para facilitar el aprendizaje rápido, en especial por principiantes, de la teoría y uso práctico de la armonía tonal. En esta herramienta se pueden construir acordes en un "tablero" de notas dispuestas en intervalos específicos. Tiene por lo menos seis usos generales: Instrumento musical, herramienta para enseñar armonización básica, herramienta de aprendizaje para explorar la teoría de la música tonal, herramienta de aprendizaje por descubrimiento para componer y modificar secuencias de acordes, y notación para secuencias de acordes que no son obvios en notaciones convencionales.

Harmonic Walk es un ambiente interactivo donde la posición del usuario en un espacio rectangular genera distintos acordes. Donde el desafío del usuario es generar una progresión armónica caminando el camino correcto.

Una desventaja de estos dos softwares es que para que el estudiante pueda usarlo es necesaria la guía constante de un educador.

Los playgrounds son similares entre sí. Cada uno con pequeñas actividades digitales orientadas a la educación musical. Sin embargo, su contenido teórico musical es muy básico.

No se encontraron juegos para la enseñanza de la armonía musical a nivel universitario.

2.5. Usabilidad en Dispositivos Móviles

La ISO defina la usabilidad como "La medida con la que un producto se puede usar por usuarios determinados para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso concreto" (ISO 9241-11). Una parte importante de asegurarse que los usuarios tengan una experiencia positiva con el juego es asegurarse que la interfaz facilite el uso de la aplicación. [19]

El método preferido para medir la usabilidad en juegos es la evaluación por medio de heurísticas. [8] Se le hacen preguntas asociadas a características deseables en la aplicación a los usuarios que sirvan para medir la experiencia que tienen con la aplicación. Las respuestas a estas preguntas son según su opinión.

Existe desacuerdo de si la jugabilidad de un juego se debe considerar como un criterio de usabilidad.[8] EFlowGame es un cuestionario de usabilidad por heurísticas que considera la disfrutabilidad del juego. [7]

Debido a que la usabilidad depende del dominio sobre el cual se trabaja, existen muchas heurísticas posibles para medir la usabilidad de una aplicación. En contextos de aplicaciones móviles, por ejemplo, puede ser necesario evaluar el impacto que tiene el movimiento del usuario sobre el uso de la aplicación. [19] Existen revisiones sobre los distintos modelos de evaluación [19] y meta-heurísticas para poder elegir qué preguntas heurísticas realizar para evaluar juegos acorde a las ideas que se puedan usar para describir el juego [8]

2.6. Juegos Asíncronos

Los juegos de multijugador se pueden dividir entre sincrónicos y asincrónicos. Los juegos sincrónicos son aquellos donde todos los jugadores están realizando acciones en el juego al mismo tiempo. Mientras que los juegos asincrónicos son aquellos donde los jugadores juegan en secuencia en vez de simultáneamente. [6]

El juego que se desarrolló pertenece a la categoría de los juegos asíncronos. Por lo que cabe revisar otros juegos asíncronos y cómo lo ocupan.

Existen juegos para dispositivos móviles que aprovechan la asincronicidad en sus experiencias. Words with Friends es un juego basado en Scrabble para dispositivos móviles en el cual los jugadores pueden jugar sus turnos sin que los otros jugadores tengan el juego activado. Esto permite a los jugadores jugar múltiples partidas, en las cuales solo es necesario hacer un movimiento cada par de días.

Figura 4: Words with Friends

A su vez, Pokemon Go permite a los jugadores competir por controlar gimnasios, los cuales son puntos de interés para los jugadores. Para esto un jugador puede realizar una captura de gimnasio y después otro jugador puede intentar vencer una versión controlada por computadora del primer jugador para quitarle el control del gimnasio capturado. Este multijugador se realiza sin que los dos jugadores jueguen al mismo tiempo.

El uso que se le da a la asíncronicidad del juego es como el de Words with Friends. En que los jugadores pueden realizar sus jugadas en un periodo de tiempo de días.

3. Problema

3.1. Descripción del problema

La armonía musical es importante dentro de la teoría musical. Pero es difícil de aprender. Al punto de poder volverse una barrera en la educación musical de los estudiantes con baja motivación.

Para mejorar la capacidad de aprendizaje de armonía musical, existe la oportunidad de crear un nuevo juego con el cual enseñar la armonía musical, aprovechando los principios de la enseñanza constructivista y la recompensa endógena que es el juego.

En esta memoria se busca desarrollar y evaluar el juego Tone Cluster que fue diseñado con estos objetivos.

3.2. Requisitos de Usuario

Los requisitos de usuario describen lo que el sistema debe poder hacer en términos generales. Esto es, que servicios puede entregar el sistema a sus usuarios.

Los siguientes requisitos fueron obtenidos a través de entrevistas con la diseñadora del juego Daniela Medel y desde su tesis.

RU 01: Cuenta del jugador
Cada jugador debe tener su propia cuenta con la cual entrar a las partidas y a la cual se le asocie la información permanente de su uso de la aplicación. Se debe poder ingresar a la cuenta, registrar una cuenta nueva y recuperar la contraseña de la cuenta. Obtenido durante el desarrollo.

RU 02: Interfaz de Amigos
Aquí podemos ver nuestros amigos y agregar o quitar amigos. Además se pueden encontrar jugadores dentro de la misma aplicación y agregarlos como amigos para poder jugar con ellos.

RU 03: Multijugador online
Esto es muy importante para el cumplimiento de los criterios para un diseño exitoso de un videojuego educativo. La interacción con otros jugadores resulta esencial dentro de la concepción de constructivismo social planteada en este proyecto. Es por esta razón que es fundamental que se juegue con otras personas y no solamente con una inteligencia artificial. La importancia de hacerlo online es la posibilidad de jugar desde cualquier lugar y en cualquier momento, pudiendo jugar varias partidas a la vez sin la necesidad de hacerlo de manera sincrónica.

RU 04: Interfaz del Menú Principal
En esta interfaz se encuentran todas las funciones de la aplicación. Se puede entrar a la interfaz de Usuario en la parte superior. Luego se puede ir a las partidas activas y seguir

jugando con otros jugadores partidas pendientes y en un botón vistoso iniciar una nueva partida. Además, la aplicación cuenta con botones con íconos independientes para entrar a las Reglas, a Amigos, y a Configuración.

RU 05: Revisar las invitaciones a partidas
Los usuarios deben poder ver las invitaciones que les hayan mandado otros usuarios para unirse a una partida. Además de poder aceptar o rechazar estas invitaciones. Al aceptarla, el usuario es enviado a la partida para jugar.

RU 06: Interfaz del Juego
Luego de que el usuario escoge amigos, se llega al juego mismo. Aquí se despliega el tablero y diferentes funciones para ayudar en el desempeño del juego.

RU 07: Fichas de notas
Las fichas que se ubican en el juego tienen letras de notas en clave americana y un valor numérico. Las fichas que sean más difíciles de ubicar en un nivel dado deben dar un mayor puntaje. Por ahora, todas las fichas tienen un puntaje de 1.

RU 08: Mazo de fichas
Todas las fichas de la partida comienzan en una reserva o mazo. Durante la partida, se sacan las fichas de este mazo para ponerlas en las manos de los jugadores.

RU 09: Tablero de de la partida
El tamaño del tablero sobre el cual se pueden ubicar las fichas durante la partida es 15x15. Todos los jugadores pueden ver las fichas que hay sobre el tablero. Se puede aumentar y reducir el zoom de la cámara sobre el tablero.

RU 10: Mano de fichas
Cada jugador tiene una mano de 5 fichas que no pueden ser vistas por los otros jugadores. Cada jugador comienza con 5 fichas en su mano. Al final de cada turno, el jugador al que le tocó su turno recibirá la cantidad de fichas nuevas necesaria para tener nuevamente 5 fichas en su mano.

RU 11: Turnos de los jugadores
La partida se juega por turnos en orden según el orden de aparición en la lista superior de los jugadores.

RU 12: Uso del turno: Realizar una jugada sobre el tablero
Los jugadores pueden hacer una jugada ubicando fichas sobre el tablero cuando es su turno. Las fichas se ubican para crear acordes.

RU 13: Uso de turno: Cambiar fichas de la mano
En su turno, el jugador puede reemplazar todas las fichas que tenga en su mano. En orden, las fichas son devueltas al mazo, el mazo es revuelto y el jugador roba la misma cantidad de fichas que devolvió. Tomar esta acción termina el turno.

RU 14: Uso de turno: Pasar de turno
En lugar de poner fichas en el tablero, o cambiar fichas, un jugador puede optar por pasar, sea que tenga las fichas necesarias para crear una o varias palabras o no.

RU 15: Fase de Votación
El jugador puede poner en duda la validez de un acorde. Si el acorde que colocó otro jugador parece que no es correcto, el jugador puede votar por rechazarlo. Todos los jugadores deben rechazar el acorde para que el acorde en cuestión se remueva del tablero y el jugador que puso el acorde deberá ceder el turno.

RU 16: Puntajes de los jugadores
Los jugadores que participan en una partida tienen un puntaje que incrementan haciendo jugadas. Los puntos del turno se calculan sumando los valores de los números de las fichas, tomando en cuenta si pasó por una casilla de premio.

RU 17: Fin del juego
La partida se acaba si todas las fichas del mazo han sido robadas y el jugador al cual le toca el turno ha usado todas las fichas en su mano, o si todos los jugadores han hecho 2 jugadas consecutivas sin puntaje. Las cuales pueden ser cambios, pases o rechazos de jugadas.

RU 18: Sumar puntajes de la partida a los puntajes generales de los jugadores
Cuando termina la partida, el puntaje que cada jugador haya obtenido se suma a su puntaje general.

RU 19: Interfaz de Usuario
Aquí se pueden ver los logros y el nivel en que se encuentra el usuario dentro del juego.

RU 20: Interfaz de Ayuda
Esta interfaz es un menú que permite ir a otras interfaces de la aplicación. Desde aquí se puede ir a la interfaz de reglas y la interfaz de soporte.

RU 22: Interfaz de Reglas
En esta interfaz se muestran las reglas generales del juego.

RU 23: Enviar un mensaje de soporte
El usuario debe poder escribir un mensaje y enviarlo al equipo de soporte.

3.3. Requisitos de Software

Los requisitos de software describen el sistema en términos de cómo se debe implementar. Cada requisito de software cubre un caso de prueba o un caso de uso del usuario.

RS 01: Cuenta del jugador en la base de datos
La base de datos deberá guardar la información de las cuentas de los usuarios. La cuenta de cada jugador deberá tener asociados su correo electrónico, un nombre de usuario y una contraseña.
Origen: RU 01.

RS 02: Escena de ingreso de cuenta
La aplicación deberá tener una interfaz desde la cual se pueda ingresar a la cuenta. Además, esta interfaz deberá tener un botón con el cual pasar a la interfaz de registro de cuenta y un botón con el cual pasar a la interfaz de recuperación de contraseña. Esta interfaz es la primera que el usuario ve al iniciar la aplicación.
Origen: RU 01.

RS 03: Componentes de UI de ingreso a la cuenta
La escena de ingreso de cuenta deberá tener los componentes de UI para que el usuario pueda ingresar las credenciales de su cuenta para acceder al resto de la aplicación. Esto incluye campos donde se puedan ingresar el correo electrónico y la contraseña. Además, debe haber un botón con el cual iniciar sesión.
Origen: RU 01.

RS 04: Transacción de ingreso de cuenta
La aplicación y el servidor deberán poder manejar transacciones solicitadas para ingresar a una cuenta. Esta transacción deberá requerir un correo electrónico y una contraseña. Al enviar esta transacción, si la información de acceso es correcta, la aplicación debe acceder a la cuenta del usuario y cambiar a la escena del menú principal. Si la información es incorrecta, se debe desplegar un mensaje que avise que el correo electrónico o contraseña son incorrectos.
Origen: RU 01.

RS 05: Escena de registro de cuenta
La aplicación deberá tener una escena que incluya los componentes de UI necesarios para registrar una nueva cuenta de jugador. Además, la escena deberá tener un botón con el cual pasar a la interfaz de ingreso de cuenta y un botón con el cual pasar a la interfaz de recuperación de contraseña.
Origen: RU 01.

RS 06: Componentes de UI de registro de cuenta
El usuario deberá poder crear su propia cuenta de Tone Cluster desde dentro de la aplicación. Para esto, la escena de registro de cuenta deberá tener componentes de UI que sean campos donde se puedan ingresar la dirección de correo electrónico, el nombre de usuario, la contraseña y la confirmación de la contraseña. Además, debe haber un botón que se pueda apretar para registrar la cuenta nueva usando la información ingresada.
Origen: RU 01.

RS 07: Transacción de creación de cuenta
La aplicación y el servidor deberán poder manejar transacciones solicitadas para crear una cuenta nueva. Al crear una cuenta, el servidor deberá guardarla en la base de datos asociando su correo electrónico, nombre de usuario y contraseña. Al registrar la cuenta nueva, la aplicación debe pasar automáticamente a la escena de menú principal.
Origen: RU 01.

RS 08: Escena de restablecimiento de contraseña
La aplicación deberá tener una escena que incluya los componentes de UI necesarios para restablecer su contraseña. Además, debe tener un botón con el cual volver a la escena de

ingreso de cuenta y un botón con el cual ir a la escena de registro de cuenta.
Origen: RU 01.

RS 09: Componentes de UI de restablecimiento de contraseña
La aplicación deberá tener componentes de UI con los cuales el usuario pueda ingresar su dirección de correo electrónico para recibir un correo electrónico con el cual pueda ir un sitio web que le permita restablecer la contraseña de su cuenta de ToneCluster. Deberá haber un campo de texto para ingresar su correo electrónico y un botón con el cual comenzar el proceso de restablecimiento de contraseña.
Origen: RU 01.

RS 10: Transacción de restablecimiento de contraseña
El usuario deberá poder restablecer su contraseña ingresando su correo electrónico y recibiendo un correo con las instrucciones para reiniciar su contraseña. La aplicación y el servidor deberán poder manejar transacciones solicitadas para restablecer la contraseña.
Origen: RU 01.

RS 11: Escena de la interfaz de amigos
La aplicación deberá tener una escena desde la cual el usuario pueda revisar a sus amigos, agregar amigos nuevos y eliminar amigos existentes.
Origen: RU 02.

RS 12: Lista de amigos en la base de datos
Cada usuario deberá tener su propia lista de amigos en la base de datos.
Origen: RU 02.

RS 13: componente de UI de la lista de amigos
La interfaz de amigos deberá incluir una tabla que liste a los amigos que el usuario haya añadido.
Origen: RU 02.

RS 14: Transacción para obtener los amigos del usuario
La aplicación y el servidor deberán poder manejar transacciones solicitadas para obtener la información de los amigos del usuario. Esto incluye nombres de usuario y otros identificadores.
Origen: RU 02.

RS 15: Componentes de UI para agregar amigos
La escena de la interfaz de amigos debe incluir un campo de texto en el cual el usuario pueda ingresar el nombre de usuario de otro usuario y un botón con el cual confirmar la solicitud de agregar un amigo.
Origen: RU 02.

RS 16: Transacción para agregar amigos.
La aplicación y el servidor deberán poder manejar transacciones solicitadas para agregar a otro usuario a la lista de amigos del usuario. Al realizar la petición, si el otro usuario existe, el servidor lo agrega a la lista de amigos del usuario en la base de datos.
Origen: RU 02.

RS 17: Componentes de UI para quitar amigos
En la tabla que lista a los amigos, por cada amigo que aparezca, deberá haber un botón con una cruz que, al ser apretado, hace que se elimine al amigo de la lista de amigos del usuario.
Origen: RU 02.

RS 18: Transacción para quitar un amigo
La aplicación y el servidor deberán poder manejar transacciones solicitadas para quitar a otro usuario a la lista de amigos del usuario. Al realizar la petición, el servidor lo quita de la lista de amigos del usuario en la base de datos.
Origen: RU 02.

RS 19: Información de la partida en la base de datos
La base de datos deberá contener partidas accesibles por internet. La partida debe tener campos que guarden la información del juego. Tales como cuál es el jugador al que le toca su turno. Cada partida en la base de datos deberá tener su propia información separada de las otras.
Origen: RU 03.

RS 20: Acceso asíncrono a las partidas en la base de datos
La información de las partidas en la base de datos deberá ser persistente y accesible por los jugadores en cualquier momento. Permitiendo así que la partida avance aún si los jugadores no están conectados al mismo tiempo y que persista aún si no hay ningún jugador conectado.
Origen: RU 03.

RS 21: Escena de selección de amigos y creación de partida
La aplicación deberá tener una escena desde la cual el jugador pueda elegir a los amigos que invitará a su partida.
Origen: RU 03.

RS 22: Componentes de UI que listen amigos para invitar
La escena de creación de partida deberá incluir la lista de amigos del usuario. El usuario deberá poder escoger entre 1 y 3 amigos de la lista para que pasen a ser parte de la partida. Luego debe poder apretar un botón para crear la partida y mandar las invitaciones.
Origen: RU 03.

RS 23: Transacción para crear partidas online
Se debe poder crear partidas accesibles por internet. La partida creada debe tener asociados a sus jugadores. Los jugadores a los que esté asociada la partida deben poder ver las partidas que están jugando y acceder a ellas. La aplicación y el servidor deberán poder manejar transacciones solicitadas para crear una nueva partida.
Origen: RU 03.

RS 24: Transacción para enviar invitaciones
Cuando el usuario crea una nueva partida, se deben enviar invitaciones a todos los jugadores que este haya escogido para jugar. La aplicación y el servidor deberán poder manejar transacciones solicitadas para enviar las invitaciones a los otros jugadores.
Origen: RU 03.

RS 25: Escena del menú principal
La aplicación deberá tener una escena que incluya los componentes de UI para ver las partidas activas del jugador, ver sus invitaciones pendientes, la barra inferior de navegación, la información general del usuario con su nombre de usuario y puntaje general, y botones que lleven a las páginas del usuario, ayuda, y creación de una nueva partida.
Origen: RU 04.

RS 26: Puntaje general del jugador en la base de datos
La base de datos debe incluir el puntaje general de cada jugador como parte de su información.
Origen: RU 04.

RS 27: Petición para pedir el puntaje general del jugador
La aplicación y el servidor deberán poder manejar transacciones solicitadas para obtener el puntaje general del usuario.
Origen: RU 04.

RS 28: Lista en la UI que contiene las invitaciones a partidas
El usuario deberá ser capaz de ver las invitaciones a partidas que les hayan mandado otros usuarios en una lista. Esta lista deberá estar ubicada en el menú principal de la aplicación. Esta lista deberá mostrar las invitaciones como botones en un contenedor horizontal deslizable. Las partidas aceptadas o rechazadas dejan de aparecer en la lista.
Origen: RU 05.

RS 29: Transacción para pedir las invitaciones pendientes
La aplicación y el servidor deberán poder manejar transacciones donde se pidan las invitaciones pendientes que tenga un usuario. Cuando se resuelve esta petición, la aplicación debe hacer aparecer las invitaciones en la interfaz de lista de invitaciones a partidas.
Origen: RU 05.

RS 30: Contenedor de UI para aceptar o rechazar una invitación
El usuario deberá poder presionar el botón de una invitación para desplegar un contenedor de UI en el que se le muestre quién lo invitó a la partida y darle la opción de aceptarla o rechazarla.
Origen: RU 05.

RS 31: Componentes de UI para aceptar una invitación
En el contenedor de UI desplegado al presionar una invitación, deberá haber un botón que el usuario pueda apretar para aceptarla.
Origen: RU 05.

RS 32: Transacción para aceptar la invitación a una partida
La aplicación y el servidor deberán poder manejar transacciones solicitadas para aceptar una invitación. Al aceptar la invitación, el servidor deberá modificar en la base de datos el estado del usuario en la partida para que pase a ser participante, y la aplicación deberá cambiar a la interfaz de la partida aceptada.
Origen: RU 05.

RS 33: Componentes de UI para rechazar una invitación
En el contenedor de UI desplegado al presionar una invitación, deberá haber un botón que el usuario pueda apretar para rechazarla.
Origen: RU 05.

RS 34: Transacción para rechazar una invitación
La aplicación y el servidor deberán poder manejar transacciones solicitadas para rechazar invitaciones a partidas. Tras rechazar una invitación, el servidor deberá modificar en la base de datos el estado del usuario en la partida para que deje de ser invitado, y la aplicación deberá actualizar la lista de invitaciones para que la invitación rechazada ya no aparezca.
Origen: RU 05.

RS 35: Lista en la UI que contiene las partidas activas
Se usa la misma interfaz para ver las partidas activas que para ver las invitaciones pendientes. Al apretar sobre el botón de una partida, la aplicación deberá cambiar de escena a la del juego cargando la información de esa partida.
Origen: RU 03.

RS 36: Transacción de pedir partidas activas
La aplicación y el servidor deberán poder manejar transacciones que pidan la información de todas las partidas de las que el usuario es parte.
Origen: RU 03.

RS 37: Componente de UI de la barra de navegación
Las interfaces de la aplicación, excepto la interfaz del juego y las interfaces previas a ingresar a la cuenta, deberán tener una barra de navegación en la parte inferior. Esta barra deberá contener botones que lleven a las escenas del menú principal y a la escena de la lista de amigos.
Origen: RU 04.

RS 38: Escena del juego
La aplicación deberá tener una escena para la interfaz del juego. Esta deberá incluir una lista con los jugadores de la partida, el tablero de la partida, la mano del jugador, un menú desplegable, un botón para confirmar la jugada actual y una bolsa con la cantidad de fichas restantes en el mazo. Además, la escena deberá tener un botón para regresar al menú principal y un botón para ir a la página de ayuda.
Origen: RU 06.

RS 39: Transacciones para obtener la información pública de la partida
La aplicación y el servidor deberán poder manejar transacciones en las que la aplicación reciba información que todos los jugadores que participan en una partida deben poder ver.
Origen: RU 03.

RS 40: Transacciones para obtener la información privada de la partida
La aplicación y el servidor deberán poder manejar transacciones en las que la aplicación reciba información que solo sea visible por el jugador al que pertenece esa información. Al obtener la información de la partida desde la base de datos, los jugadores no deben obtener

la información privada de los otros jugadores.
Origen: RU 03.

RS 41: Componente de UI que liste a los jugadores de la partida
La escena del juego deberá contener una lista con los jugadores participantes en la partida. Por cada jugador debe aparecer su nombre de usuario y su puntaje.
Origen: RU 06.

RS 42: Nombres de usuario como parte de la información pública de la partida
Como parte de la información de cada partida, la base de datos deberá guardar una lista de los jugadores pertenecientes a la partida con sus nombres de usuario.
Origen: RU 06.

RS 43 : Elemento de UI del menú desplegable
En la escena del juego deberá haber un botón de menú en la esquina inferior izquierda de la interfaz. Al apretar este botón, se deberá desplegar un menú con un botón que, al ser apretado, permite volver al menú principal. Este menú se puede cerrar apretando fuera del mismo.
Origen: RU 06.

RS 44: Puntaje como parte de la información pública de la partida
Como parte de la información de cada partida, la base de datos deberá guardar los puntajes de los jugadores.
Origen: RU 16.

RS 45: Elemento de UI de cada ficha
Cada ficha en la aplicación deberá tener escrita encima su nota y su puntaje.
Origen: RU 07.

RS 46: Información de cada ficha
Cada instancia de ficha debe tener como propiedades su nota y su puntaje. Si está en el tablero, además deberá tener su posición.
Origen: RU 07.

RS 47: Mazo de fichas como parte de la información pública de la partida
Como parte de la información de cada partida, la base de datos deberá guardar las fichas que quedan en el mazo.
Origen: RU 08.

RS 48: Elemento de UI que cuente las fichas restantes
En la escena del juego deberá haber un ícono de bolsa sobre el cual haya un número con la cantidad de fichas restantes en el mazo.
Origen: RU 08.

RS 49: Elementos de UI para el tablero de la partida compuesto por casillas
En la interfaz del juego deberá haber un tablero compuesto por 255 casillas en las cuales se pueden ubicar fichas. Estas casillas están dispuestas en una cuadrilla con 15 filas y 15 columnas.
Origen: RU 09.

RS 50: Fichas sobre el tablero como parte de la información pública de la partida
Uno de los campos con información de la partida que se deberá guardar en la base de datos es una lista con las fichas que hay sobre el tablero. Cada ficha debe tener guardada su posición en el tablero y su nota.
Origen: RU 09.

RS 51: Cargar las fichas sobre el tablero
Cuando el jugador abra la interfaz de la partida, la aplicación deberá ubicar sobre el tablero las fichas que han sido ubicadas durante la partida. Incluyendo las fichas de la última jugada en un color diferente.
Origen: RU 09.

RS 52: Mover la cámara sobre el tablero
Si se viera todo el tablero al mismo tiempo, cada cuadro de este sería demasiado pequeño. Por lo tanto, el jugador debe poder mover la cámara sobre el tablero mientras esta se enfoca sobre solo una parte. El jugador debe poder mover la cámara arrastrando el dedo sobre el tablero. Con lo cual la cámara se mueve en la dirección opuesta del movimiento del dedo.
Origen: RU 09.

RS 53: Acercar y alejar la cámara del tablero
Se debe poder cambiar el nivel de zoom de la cámara y así cambiar entre ver todo el tablero y concentrarse en sólo una parte del tablero. El jugador reduce o aumenta el nivel de zoom apretando o separando dos dedos sobre el tablero.
Origen: RU 09.

RS 54: Elemento de UI de la mano
En la parte inferior de la escena del juego, deberá haber una barra horizontal sobre la cual las fichas del jugador pueden estar. Esta barra sirve como la mano del jugador en la aplicación.
Origen: RU 10.

RS 55: Guardar las fichas de cada mano como información privada de la partida
Uno de los campos con información de la partida que se deben guardar en la base de datos es una lista por cada jugador con las fichas que tiene en su mano. Cada ficha debe tener guardada su nota. Las fichas que cada jugador tiene en la mano deben ser visibles sólo para sí mismo.
Origen: RU 10.

RS 56: Cargar las fichas del jugador en su mano
La aplicación deberá poder tomar la información de cuáles son las fichas que el jugador tiene en su mano y usarla para llenar la mano del jugador en la UI con sus fichas.
Origen: RU 10.

RS 57: Arrastrar fichas desde la mano al tablero y viceversa
Cada jugador debe poder ubicar las fichas que tiene en su mano sobre el tablero durante su turno. Además de poner las fichas que recién ha puesto de vuelta a su mano. Al ubicar fichas en el tablero, deben ubicarse en una de sus casillas.
Origen: RU 07.

RS 58: Sonido de cada ficha
Por cada nota que pueden tener las fichas en la aplicación, debe haber un sonido de piano equivalente a esa nota. Cuando una ficha que está en la mano empieza a ser arrastrada, debe hacer sonar su nota.
Origen: RU 07.

RS 59: Repartir la mano inicial de los jugadores
Al iniciar la partida, el servidor deberá sacar 5 fichas del mazo por jugador para ponerlas en sus respectivas manos.
Origen: RU 10.

RS 60: Destacar al jugador al cual le toca el turno en la UI
En la UI que lista a los jugadores en la partida, se deberá hacer notar el turno del jugador actual destacando su nombre.
Origen: RU 11.

RS 61: Guardar el jugador al cual le toca el turno como información pública de la partida
Uno de los campos con información de la partida que se deben guardar en la base de datos es a qué jugador le toca el turno.
Origen: RU 11.

RS 62: Guardar la última jugada realizada como información pública de la partida
El servidor debe guardar la última jugada realizada. Cuando la aplicación coloca las fichas que están sobre el tablero en la interfaz del juego, deberá incluir estas fichas también.
Origen: RU 12.

RS 63: Componentes de UI para enviar una jugada
Si, en su turno, el usuario ha ubicado por lo menos una ficha desde su mano al tablero, entonces puede enviar la jugada realizada. Para enviar la jugada, el jugador debe presionar un botón que diga "Enviar" que se ubica en la parte inferior de la interfaz. El botón "Enviar" y el botón "Pasar" no deben ser accesibles al mismo tiempo.
Origen: RU 12.

RS 64: Transacción para enviar una jugada
La aplicación y el servidor deberán poder manejar transacciones realizadas para enviar una jugada. La transacción enviada debe incluir las fichas jugadas con sus respectivas notas y posiciones, las fichas restantes en la mano del jugador que realizó la jugada, y el puntaje total de la jugada. Al enviar una jugada, el servidor deberá modificar la partida de las siguientes maneras: Se pasa a la fase de votación, las fichas jugadas pasan a ser la jugada realizada del turno, el puntaje enviado es el puntaje tentativo de la jugada, y se rellena la mano del jugador que hizo la jugada. Además, al enviar una jugada, la aplicación deberá actualizar la interfaz del juego para reflejar los cambios realizados.
Origen: RU 12.

RS 65: Componentes de UI para devolver las fichas de la mano
En su turno, el jugador puede devolver las fichas de su mano al mazo presionando el símbolo

de bolsa y luego apretando un botón que diga Çambiar todas las fichas de la mano".
Origen: RU 13.

RS 66: Transacción para devolver las fichas de la mano
La aplicación y el servidor deberán poder manejar transacciones para cambiar las fichas de la mano. Al realizarse esta transacción, el servidor deberá modificar la partida en la base de datos para que las fichas en la mano del jugador sean revueltas en el mazo, se vuelva a llenar su mano, y sea el turno del siguiente jugador. Además, al cambiar fichas de la mano, la aplicación deberá actualizar la interfaz del juego para reflejar los cambios realizados.
Origen: RU 13.

RS 67: Componentes de UI para pasar de turno
La aplicación deberá permitir al usuario pasar de turno si no ha puesto ninguna ficha sobre el tablero. Para pasar, el jugador deberá poder presionar un botón que diga "Pasar" que se ubica en la parte inferior de la interfaz. El botón "Pasar" y el botón "Enviar" no deberán ser accesibles al mismo tiempo.
Origen: RU 14.

RS 68: Transacción para pasar de turno
La aplicación y el servidor deberán poder manejar transacciones para pasar de turno. Al realizarse esta transacción, el servidor deberá modificar la partida en la base de datos para que sea el turno del siguiente jugador. Además, al pasar de turno, la aplicación deberá actualizar la interfaz del juego para reflejar los cambios realizados.
Origen: RU 14.

RS 69: Guardar la fase actual del juego como información pública de la partida
Uno de los campos con información de la partida que se deben guardar en la base de datos es en qué fase se encuentra el turno. Después de que el jugador al que le toca el turno juega un acorde, se pasa a la fase de votación. Durante esta fase, si no es el turno del usuario, este puede votar por aprobar o rechazar el acorde jugado. Una vez que el acorde es aprobado o rechazado, se vuelve a la fase de juego.
Origen: RU 15.

RS 70: Votos de cada jugador como información privada de la partida
Uno de los campos con información de la partida que el servidor deberá guardar en la base de datos es el voto que cada jugador realizó durante la última fase de votación. Al iniciar la fase de votación, todos los votos deben reiniciarse a un estado neutro que indique que el jugador no ha votado.
Origen: RU 15.

RS 71: Componentes de UI para votar a favor de aprobar una jugada
El usuario deberá poder votar a favor de aprobar el acorde jugado. Para hacer esto, en la interfaz del juego deberá haber un botón que diga .ªprobar", visible durante la fase de votación cuando no es el turno del usuario.
Origen: RU 15.

RS 72: Transacción para votar a favor de aprobar una jugada
La aplicación y el servidor deberán poder manejar transacciones para que quede registrado

que el jugador votó a favor de la jugada más reciente.
Origen: RU 15.

RS 73: Componentes de UI para votar a favor de rechazar una jugada
El usuario deberá poder votar a favor de rechazar el acorde jugado. Para hacer esto, en la interfaz del juego deberá haber un botón que diga Rechazar", visible durante la fase de votación cuando no es el turno del usuario.
Origen: RU 15.

RS 74: Transacción para votar a favor de rechazar una jugada
La aplicación y el servidor deberán poder manejar transacciones para que quede registrado que el jugador votó en contra de la jugada más reciente.
Origen: RU 15.

RS 75: Aprobar la jugada en el servidor
Al aprobarse una jugada, se cambia el estado de la partida en la base de datos de las siguientes maneras: la jugada realizada durante el turno se ubica de manera definitiva en el tablero, se suman los puntos de la jugada al puntaje del jugador, se cambia la fase del juego de votar a jugar y se cambia al turno del siguiente jugador.
Origen: RU 15.

RS 76: Rechazar la jugada en el servidor
Al rechazarse una jugada, se cambia el estado de la partida de las siguientes maneras: la jugada realizada se descarta, las fichas jugadas se revuelven en el mazo, se cambia la fase del juego de votar a jugar y se cambia al turno del siguiente jugador.
Origen: RU 15.

RS 77: Calcular el puntaje de una jugada
Cuando el jugador va a enviar una jugada, la aplicación deberá calcular el valor en puntos de la jugada considerando casillas de premio.
Origen: RU 16.

RS 78: Guardar el puntaje de la última jugada como información pública de la partida
El servidor debe guardar el puntaje de la última jugada realizada. Este puntaje se debe sumar al puntaje del jugador en la partida solamente si su jugada es aprobada.
Origen: RU 16.

RS 79: Las casillas para fichas pueden ser casillas de premio
Las casillas del tablero deben tener un atributo para que cada casilla guarde su bonus de puntaje. Si la casilla no tiene bonus, este atributo indica que la casilla no suma puntos adicionales.
Origen: RU 16.

RS 80: Sumar el puntaje bouns al puntaje de la jugada
Cuando el usuario realiza una jugada ubicando fichas, la aplicación revisa y usa los bonus de las casillas recién cubiertas para aumentar el puntaje de la jugada realizada. No se ocupan los bonuses de casillas que no hayan sido cubiertas ese mismo turno.
Origen: RU 16.

RS 81: Casillas con una corchea
Las casillas con una corchea duplican el puntaje de la ficha ubicada encima.
Origen: RU 16.

RS 82: Casillas con dos corcheas
Las casillas con dos corcheas duplican el puntaje de toda la jugada realizada.
Origen: RU 16.

RS 83: Casillas con tres corcheas
Las casillas con tres corcheas triplican el puntaje de toda la jugada realizada.
Origen: RU 16.

RS 84: Rellenar la mano del jugador
Cuando el jugador al que le toca el turno realiza una jugada colocando fichas, inmediatamente se remueven fichas del mazo para colocarlas en la mano del jugador hasta que tenga 5 en la mano. Si no hay suficientes fichas en el mazo para llenar la mano, el jugador recibe todas las fichas que queden en el mazo y el resto de la mano queda vacía.
Origen: RU 10.

RS 85: Fin del juego como información pública en la partida
Uno de los campos con información de la partida que se deberá guardar en la base de datos es un campo que indique si la partida ha terminado.
Origen: RU 17.

RS 86: Terminar la partida cuando se acaba el mazo y la mano del jugador actual
Al final de cada turno, el servidor debe revisar si el mazo y la mano del jugador siguiente están vacías. De ser ese el caso, un campo de la partida que indique si esta ha terminado deberá cambiar a verdadero y no se deberá poder realizar más jugadas.
Origen: RU 17.

RS 87: Terminar la partida cuando se ha pasado dos rondas seguidas
Al final de cada turno que no haya terminado con una jugada con éxito, se debe sumar uno a la cantidad de turnos que han pasado sin realizarse una jugada. Si la cantidad de turnos pasados iguala a la cantidad de jugadores en la partida por dos, entonces un campo de la partida que indique si esta ha terminado deberá cambiar a verdadero y no se deberá poder realizar más jugadas.
Origen: RU 17.

RS 88: Sumar los puntajes de la partida a los puntajes generales de los jugadores en la base de datos desde el servidor
Al momento de terminar la partida, el servidor deberá sumar el puntaje de la partida de cada jugador al puntaje general que cada uno tiene asociado a su cuenta en la base de datos.
Origen: RU 18.

RS 89: Escena del Usuario
La aplicación deberá tener una escena donde el usuario pueda ver su nombre de usuario y su puntaje general.
Origen: RU 19.

RS 90: Escena de ayuda
La aplicación deberá tener una escena que contenga dos botones que el usuario pueda apretar. Un botón lo lleva a la escena de reglas y el otro lo lleva a la escena de soporte.
Origen: RU 20.

RS 91: Escena de reglas
La aplicación deberá tener una escena donde el jugador pueda ver las reglas del juego. Esta escena deberá contener texto que explique las reglas, imágenes que complementen el texto y botones que permitan navegar entre las distintas reglas.
Origen: RU 21.

RS 92: Escena de soporte
La aplicación deberá tener una escena desde la cual el usuario pueda enviarle un mensaje al equipo de soporte del juego.
Origen: RU 22.

RS 93: Componentes de UI para enviar un mensaje de soporte
La UI de la escena de soporte deberá incluir un campo de texto donde el usuario pueda ingresar el tema del mensaje, un campo donde el usuario pueda ingresar el mensaje propiamente tal, y un botón para enviar el mensaje escrito.
Origen: RU 22.

RS 94: Transacción para enviar un mensaje de soporte
La aplicación y el servidor deberán poder manejar transacciones para enviar un mensaje que sea visible para el equipo de desarrollo.
Origen: RU 22.

3.4. Matriz de Trazado

La trazabilidad de una especificación de requisitos es la propiedad que refleja la facilidad de encontrar requisitos relacionados [26]. Es necesario mantener las relaciones entre los requisitos, sus fuentes y el diseño del sistema para poder analizar la motivación de los cambios propuestos y el impacto que estos cambios puedan tener sobre otras partes del sistema [24]. La información de la trazabilidad de requisitos se suele representar usando matrices de trazabilidad. Las cuales relacionan los requisitos entre sí [26].

La siguiente matriz de trazado conecta los requisitos de usuario y los requisitos de software. Es decir, por cada requisito de software indica el requisito de usuario del cual está basado.

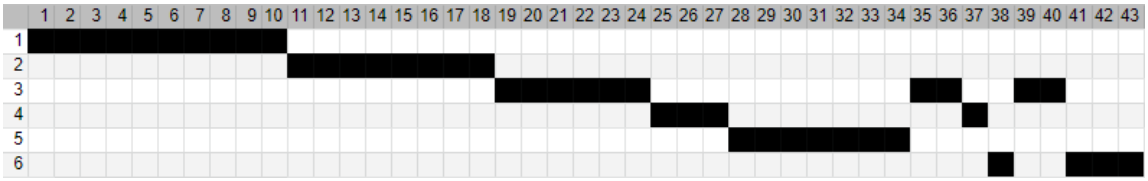


Figura 5: Matriz de trazado 1

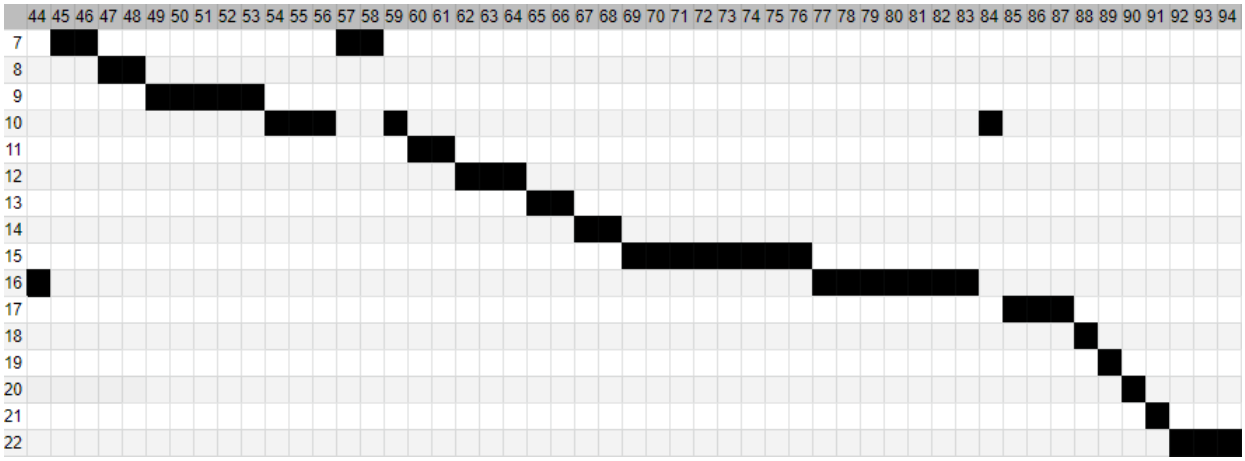


Figura 6: Matriz de trazado 2

3.5. Requisitos No Funcionales

Usabilidad

La aplicación requiere tener una buena usabilidad, ya que esta beneficia el que se genere un estado de flow en el usuario, el cual a su vez beneficia el aprendizaje. Para probar la usabilidad de la aplicación, se realizó la evaluación descrita en el capítulo 4.3. cuyos resultados son descritos en el capítulo 7.

3.6. Restricciones Significativas para la Arquitectura

Dispositivos móviles

El juego desarrollado debe poder correr en dispositivos móviles sin que el peso en recursos necesarios de la aplicación se lo impida.

Juego 2D

El juego de la aplicación toma lugar en un espacio de dos dimensiones. Así que se prefieren plataformas que permitan el desarrollo en 2D.

Juego en línea

La información de la partida debe compartirse entre los usuarios a través del internet.

Juego asíncrono

El estado de las partidas debe perdurar aún si todos los jugadores de la partida se desconectan. Las partidas pueden durar días, con lo cual los jugadores pueden jugar sus turnos sin prisa.

3.7. Restricciones sobre la Implementación y Evaluación

Costo

Para no tener que gastar dinero durante el desarrollo, se decidió que sólo se iban a usar herramientas y software gratuitos. Esto incluye los servidores para el juego en línea y las plataformas o motor para implementar el juego.

Tiempo de desarrollo

El tiempo de desarrollo de la memoria es limitado. Así que se favorecen las opciones de implementación que permitan terminar pronto con poco riesgo. Además, es un riesgo que la aplicación sea más grande de lo esperado.

3.8. Criterios de Aceptación

Para que el videojuego sea considerado válido, este debe ser aceptado por los usuarios finales [23, p.24-37]. Para conocer sus opiniones, se realizó una evaluación de usabilidad con ellos al final del proceso de desarrollo de la aplicación.

4. Trabajo Previo

4.1. Diseño Previo

El juego implementado durante esta memoria se llama Tone Cluster. Fue diseñado inicialmente por la estudiante de la Universidad de Chile Daniela Medel Sierralta, como su tesis para optar al grado de Magíster en Educación mención Informática Educativa [1]. Sus aportes se limitan al diseño del juego y de las interfaces. La implementación de la aplicación para celular y su evaluación fueron realizados en su totalidad por el memorista.

4.1.1. Diseño previo de las reglas del juego

El juego mismo es similar al juego Scrabble. Con la diferencia que, en vez de usarse letras para generar palabras, se usan fichas que representan notas en clave americana para crear acordes. Esta diferencia permite cambiar el objetivo con el que se diseña el juego. Pues deja de ser solamente un juego con el cual entretenerse y pasa a ser un juego con el cual enseñar de armonía musical a nivel de alumnos universitarios y profesionales de la música. Y dado que este tipo de juegos son muy escasos, este pasa a ser el aporte del presente trabajo.

Hay una cuadrilla de 15x15 donde se pueden ubicar fichas con letras que indican notas. Por turnos, los jugadores ubican fichas desde sus manos para formar grupos de notas que sean acordes. El primer acorde debe estar ubicado al centro de la cuadrilla, mientras que los otros acordes deben compartir una ficha con los acordes ya armados.

Las partidas se juegan de hasta 4 jugadores. En cada turno, el jugador al que le toca tiene que realizar una jugada valida. Sea esta jugar fichas en una posición válida, cambiar fichas de su mano o pasar de turno. Los jugadores pueden ubicar cualquier combinación de fichas si están todas en una hilera válida y unidas a lo largo de la hilera entre sí o por fichas ya puestas. Luego el jugador indica qué acorde puso.

Los otros jugadores luego pueden poner en duda la validez del acorde colocado. Todos los jugadores deben rechazar el acorde para que este se remueva del tablero y el jugador que lo colocó deberá ceder el turno.

Antes de iniciar una partida, se debe elegir el nivel de dificultad. El juego está dividido en 13 niveles de dificultad. El nivel elegido cambia las notas disponibles y con ello los acordes que se pueden construir.

4.1.2. Diseño previo de las interfaces

La interfaz principal de la aplicación es la interfaz para jugar partidas, como se ve en la figura 7. Se llega a esta interfaz luego de escoger a los amigos y el nivel en que se va a jugar. Aquí se despliega el tablero y diferentes funciones para ayudar en el desempeño del juego.

Figura 7: Diseño previo a la memoria de la interfaz del juego.

Además del modo principal de juego, hay páginas de la aplicación que le muestran al usuario información asociada a su cuenta y páginas con herramientas complementarias. Estas se muestran en la figura 8 y se describen en el orden que ahí aparecen a continuación:

1. Inicio: En esta interfaz se encuentran todas las funciones de la aplicación. Se puede entrar a la página de Usuario y Mensajes en la parte superior. Luego se puede ir a las partidas activas y seguir jugando con otros jugadores partidas pendientes y en un botón vistoso iniciar una nueva partida. Además, la aplicación cuenta con 6 botones con íconos independientes para entrar al Piano, al Círculo de Quintas, a Funciones Armónicas, a Reglas, a Amigos, y a Configuración.

2. Perfil de Usuario: Aquí se pueden ver los logros y el nivel en que se encuentra el usuario dentro del juego.

3. Ayuda: Se incorpora esta interfaz para ayudar en el conocimiento y uso de las reglas del juego, configuración de la cuenta, uso de herramientas (círculo de quintas y piano), inteligencia artificial, y el contacto con el soporte en caso de errores.

4. Configuración: Aquí se puede hacer configuraciones básicas del juego.

5. Amigos: Aquí podemos ver nuestros amigos y agregar amigos desde facebook o google. Además se pueden encontrar jugadores dentro de la misma aplicación y agregarlos como amigos para poder jugar con ellos. En esta interfaz se puede chatear y ver si están conectados para jugar.

6. Círculo de Quintas: En esta página se encuentra el círculo de quintas donde se puede explorar las diferentes tonalidades mayores y menores, y conocer las alteraciones de las diferentes escalas.

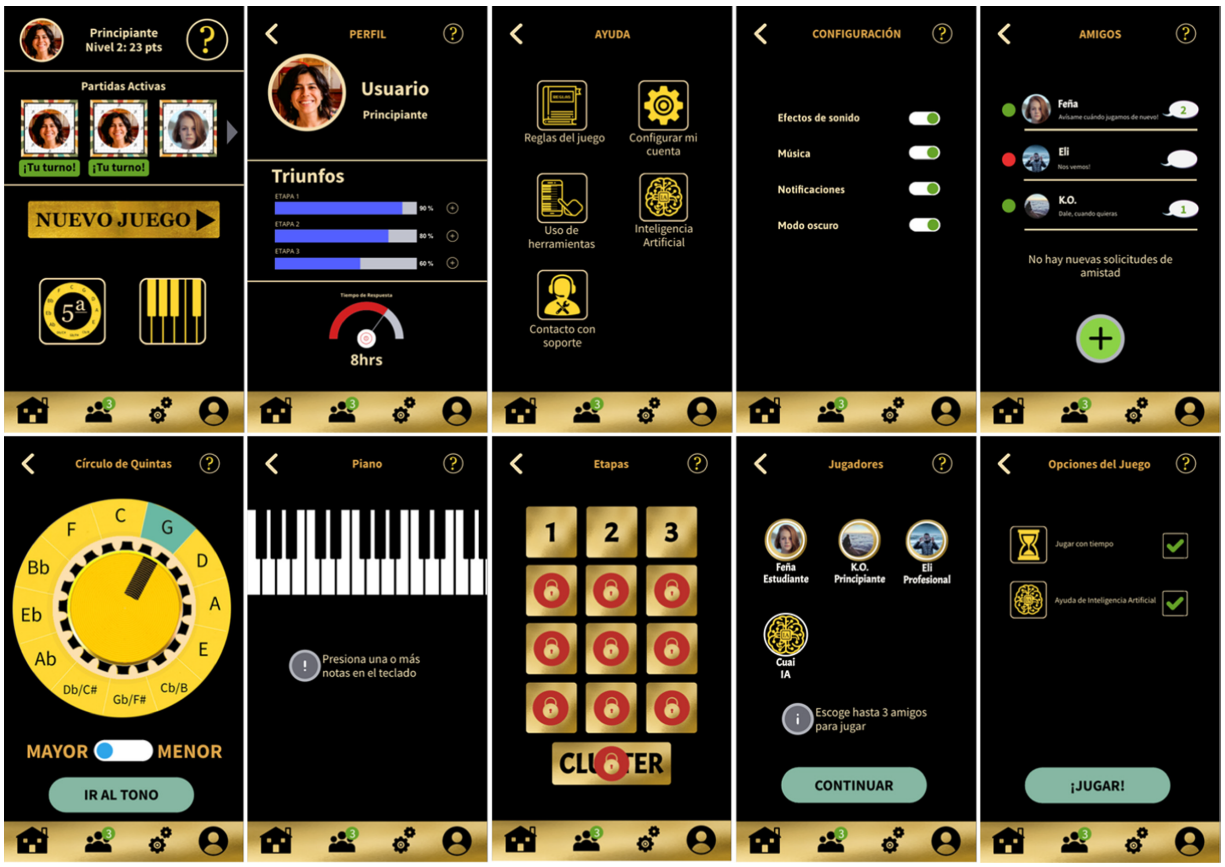


Figura 8: Diseño previo a la memoria de las interfaces: Inicio, Perfil, Ayuda, Configuración, Amigos, Círculo de Quintas, Piano, Etapas, Jugadores, Opciones del juego

7. Piano: Aquí se despliega un piano interactivo donde se pueden construir intervalos y acordes de todo tipo. Además, indica qué función cumple dicho acorde en diferentes tonalidades.

8. Etapas: Aquí se visualizan las diferentes etapas del juego, donde, para avanzar, se debe ganar una vez la etapa anterior.

9. Escoger Jugadores: Aquí se escogen los amigos con las cuales jugar la partida. Se incorporó la opción de jugar con la inteligencia artificial "Cuai". Se puede graduar el nivel de la misma (principiante, estudiante, profesional, experto, maestro).

10. Opciones del juego: Se incorporó esta interfaz antes del inicio del juego para definir opciones de tipo de juego, para incorporar diferentes niveles de desafíos.

4.2. Rediseño

Durante el proceso de desarrollo, el memorista fue descubriendo modificaciones y mejoras posibles en el diseño de la aplicación.

4.2.1. Botones separados por fase

En el diseño original de la interfaz para jugar partidas, el botón para rechazar una jugada es parte permanente de la interfaz.

4.2.2. Acercar y alejar la pantalla con un movimiento táctil

Acercar y alejar la pantalla con el movimiento táctil del "pellizco" En el diseño original de la aplicación, para poder acercar y alejar el enfoque sobre el tablero, se aprieta sobre el tablero una vez para acercar y una vez para alejar. En la implementación, esto se cambió a que el usuario puede alejar la pantalla presionando ambos dedos sobre el tablero y acercarlos y acercar la pantalla alejándolos. Esto permite mas rangos de tamaño sobre la pantalla.

4.2.3. Creación e ingreso a la cuenta

El diseño original de la aplicación requería guardar información del usuario. Pero no consideraba cómo iba a tener acceso el usuario a esa información. Para remediar esto, se implementó un sistema de ingreso de cuenta, con su respectivo registro de cuenta y cambio de contraseña.

5. Metodología

5.1. Metodología de Desarrollo

Para el desarrollo de esta aplicación, se siguió la metodología definida por Rapid Prototyping [21]. Esta metodología implica realizar prototipos de manera periódica con un avance gradual. Los cuales se muestran a usuarios evaluadores para obtener su retroalimentación de cómo debería ser el siguiente prototipo.

Rapid Prototyping involucra un proceso de tres pasos, repetidos las veces que sea necesario:
1. Prototipar: Crear un bosquejo visual de la solución o interfaz.
2. Revisar: Se comparte el prototipo con usuarios y evalúa si satisface sus necesidades y expectativas.
3. Refinar: Basado en esta retroalimentación, identificar áreas que necesitan ser mejoradas o clarificadas.

Para esta memoria, los usuarios evaluadores de este software fueron el profesor guía de la memoria y la autora del diseño de Tone Cluster Daniela Medel. Los usuarios evaluadores de esta metodología son distintos a los usuarios finales y expertos que realizaron la evaluación final.

Se eligió esta metodología por ser una metodología ágil que puede adaptarse a tiempos limitados, porque su retroalimentación constante la hace apropiada para el desarrollo de juegos y por la conexión cercana entre el desarrollador y los evaluadores.

5.2. Plan de Trabajo

Durante la primera mitad del año 2021, se realizó una interfaz básica del juego sobre el tablero. Esta está descrita en la sección 6.2. que describe el primer prototipo del juego. Esta interfaz permite ver el tablero del juego, tener fichas en la mano, deslizar fichas desde la mano al tablero y viceversa, mover la cámara sobre el tablero, y acercar o alejar la vista del tablero.

Para el desarrollo de esta memoria, se decidió implementar un prototipo del juego que contenga la parte central de la funcionalidad del juego completo. Esto es debido al gran alcance que el juego completo cubre, el cual no hubiese sido posible desarrollar en el tiempo disponible para implementar la memoria.

En primer lugar, fue crear una interfaz en la aplicación que le permita a los jugadores crear una cuenta con la cual identificarse durante las partidas, guardar información duradera del jugador como cuál es el nivel más alto que ha superado y en la cual queden guardados los datos de sus partidas actuales. Y una interfaz en la aplicación para ingresar a su cuenta.

En segundo lugar, fue implementar el que las partidas tengan permanencia, de manera que las jugadas ya hechas se mantengan almacenadas aún si ningún jugador tiene la aplicación

activada en el momento.

En tercer lugar, fue implementar una lista de amigos dentro de la aplicación con la cual los jugadores podrán invitar a sus amigos a jugar partidas de Tone Cluster.

Luego se diseño una interfaz con la que los jugadores pueden recuperar la contraseña de su cuenta en caso de olvidarla.

Después, se hizo que los jugadores puedan crear partidas en las que todos los jugadores participantes compartan el estado de la partida. Estas partidas son entre amigos y pueden ser vistas desde el menú principal. Al crear una partida, los otros jugadores reciben invitaciones a esa partida que pueden ver y rechazar o aceptar.

Se implementaron las clases y las transacciones necesarias para que la aplicación en el celular del usuario pueda comunicarse con los servidores donde se guarda la información del juego.

También se debió implementar los servicios mismos de la partida. Esto incluye las jugadas que puede realizar el jugador como enviar una jugada al servidor, pasar de turno y cambiar las fichas de su mano. Además de implementar servicios que no tienen que ver con las acciones del jugador como inicializar los datos de la partida.

Por último, se evaluó el juego. Se decidió una hora previamente para evaluarlo con usuarios y con expertos.

Durante todo el periodo de trabajo se escribió la memoria. En particular, se describió la arquitectura del software que se fue implementando.

La Carta Gantt del proyecto desarrollado se encuentra en el anexo.

6. Diseño de la solución

6.1. Arquitectura de la solución

6.1.1. Arquitectura de hardware

La aplicación funciona en celular. Esto permite a los usuarios llevar la aplicación a cualquier parte. Las partidas son guardadas en los servidores de PlayFab, a los cuales la aplicación en el celular del usuario puede acceder a través de internet.

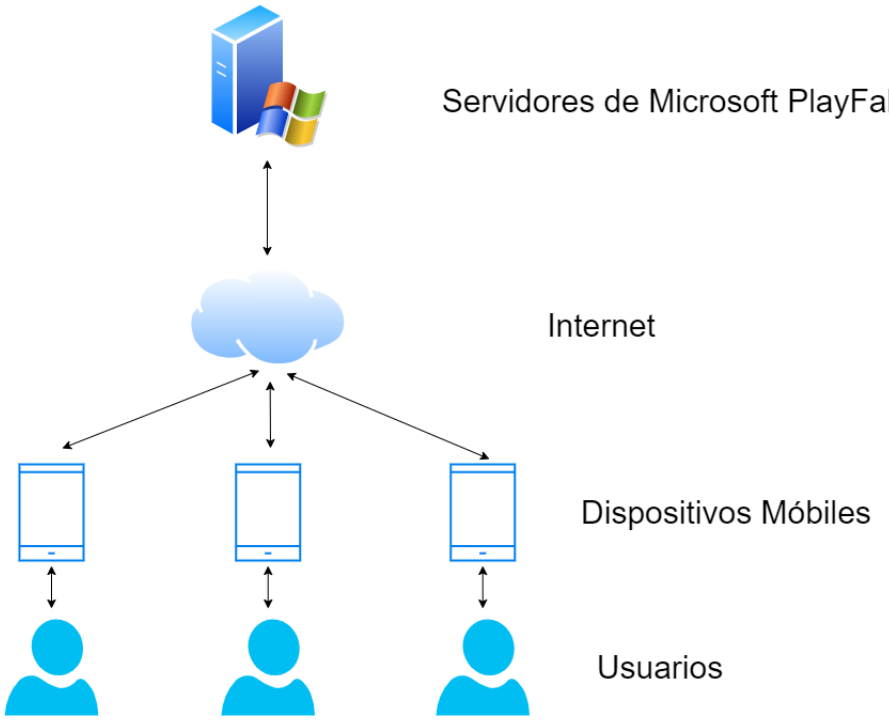


Figura 9: Arquitectura física

6.1.2. Arquitectura de software

El juego utiliza una arquitectura de cliente-servidor. El cliente se ejecuta en el dispositivo móvil del usuario en la aplicación construida desde Unity, mientras que el servidor se ejecuta en los servidores de PlayFab.

El cliente está encargado de recibir las entradas del usuario, actualizar la interfaz de usuario, procesar la lógica del juego del lado del cliente y de enviar peticiones y recibir respuestas al servidor.

El servidor está encargado de recibir las peticiones y enviar respuestas al cliente, procesar lógica de la partida en del lado del servidor, de manejar los datos que se transfieren con la base de datos y de mantener la base de datos.

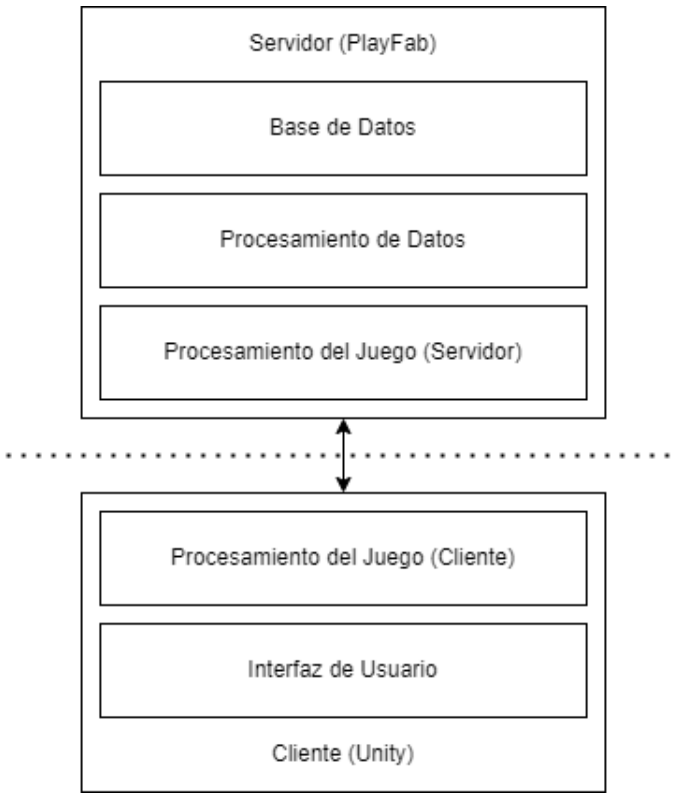


Figura 10: Arquitectura de Cliente y Servidor

Los distintos usuarios no interactúan directamente entre sí. Ni siquiera si son parte de la misma partida. En lugar de eso, la información de cada partida es guardada en el servidor y luego los jugadores pueden ver los cambios realizados por los otros jugadores. Esto permite usar una arquitectura de repositorio para el manejo de la información de las partidas y la comunicación entre los jugadores.

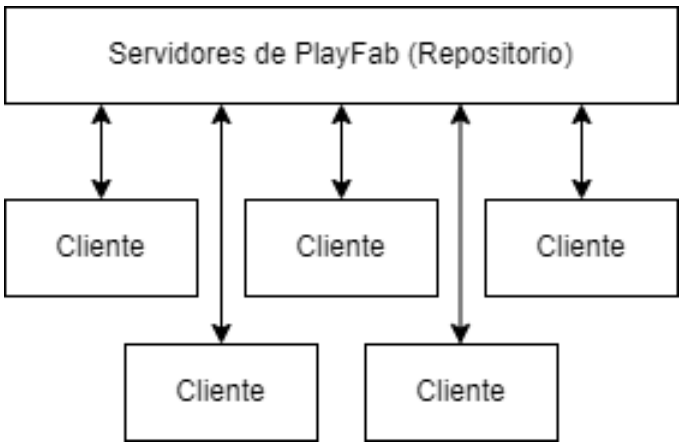


Figura 11: Arquitectura de Repositorio

La comunicación entre el cliente y el servidor utiliza el protocolo TCP, con un tráfico medido en la escena de la partida de 34 kilo-bytes por minuto mientras no se interactúa con la interfaz.

Las llamadas entre el cliente y el servidor están encriptadas acorde a estándares modernos [43] y es necesario un ticket de sesión para realizar las llamadas de API que requieren a un usuario autenticado. [44]

6.2. Experiencia de Usuario Diseñada

Durante la primera mitad del año 2021, el memorista desarrolló un mockup desde el cual se pudo seguir desarrollando el resto del juego. El objetivo de este mockup fue probar si era posible crear la funcionalidad del tablero usando Unity.

Los elementos de este mockup fueron suficientes para jugar una partida de dos jugadores en el primer nivel. Si bien con la diferencia a lo planeado de que los jugadores pueden ver las fichas del otro jugador.

Ambos jugadores empiezan con sus manos vacías. Aquí se colocan las fichas que pueden usar. Para obtener una ficha, se aprieta el botón verde de la esquina. Esto hace aparecer una ficha en la mano de color verde. Luego, se puede arrastrar la ficha desde la mano verde a la mano del jugador correspondiente.

Tomándose turnos, los jugadores pueden arrastrar las fichas en sus manos hacia el tablero. Al soltar una ficha sobre el tablero, esta se ubica en la casilla del tablero sobre la cual se soltó.

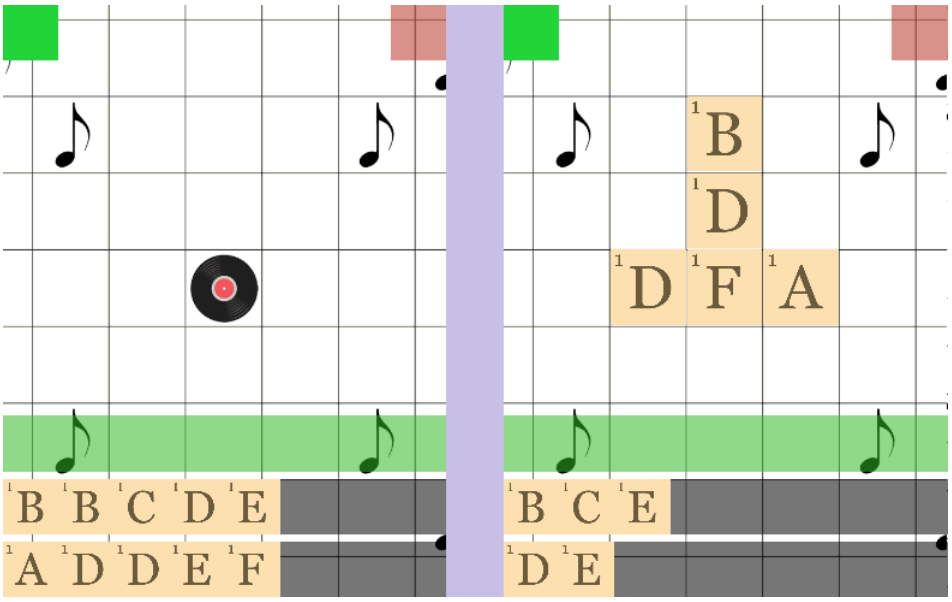


Figura 12: Deslizar fichas desde el rack hacia el tablero

Como es solo un mockup, no hay mecanismos todavía para hacer respetar ordenes de turnos o ubicaciones de las fichas.

Es posible arrastrar una ficha al cuadrado rojo de la esquina superior derecha. Esto hace

que la ficha desaparezca y vuelva a agregarse a las fichas que pueden aparecer al apretar el botón verde.

Además de mover fichas, es posible mover la cámara sobre el tablero. Para esto, se desliza el dedo sobre un espacio vacío del tablero en dirección opuesta de donde se quiere moverse.

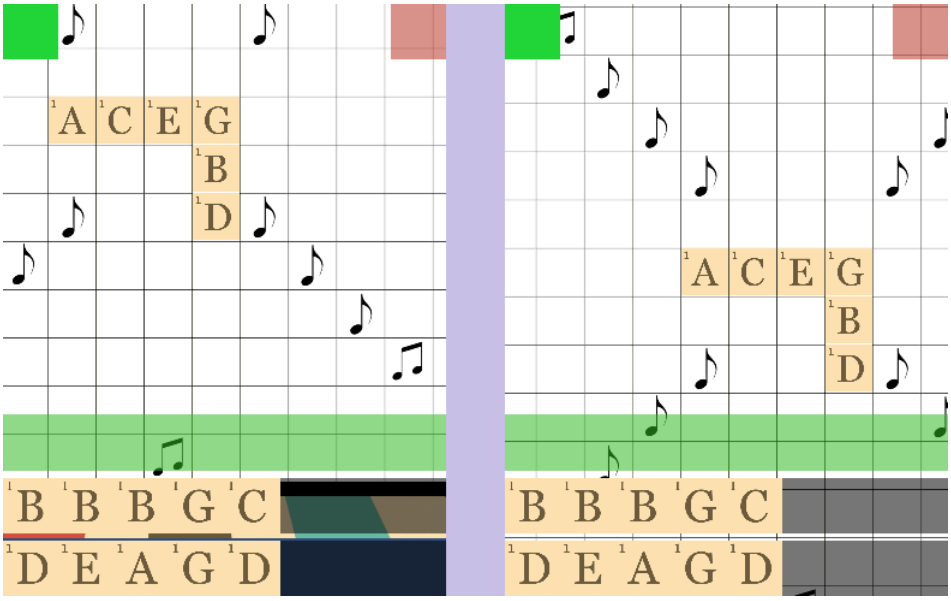


Figura 13: Deslizar la vista sobre el tablero

Por último, es posible acercar y alejar la cámara del tablero. Para esto se deslizan dos dedos sobre la pantalla del celular. Los dedos se juntan para alejar la cámara o se separan para acercar la cámara.

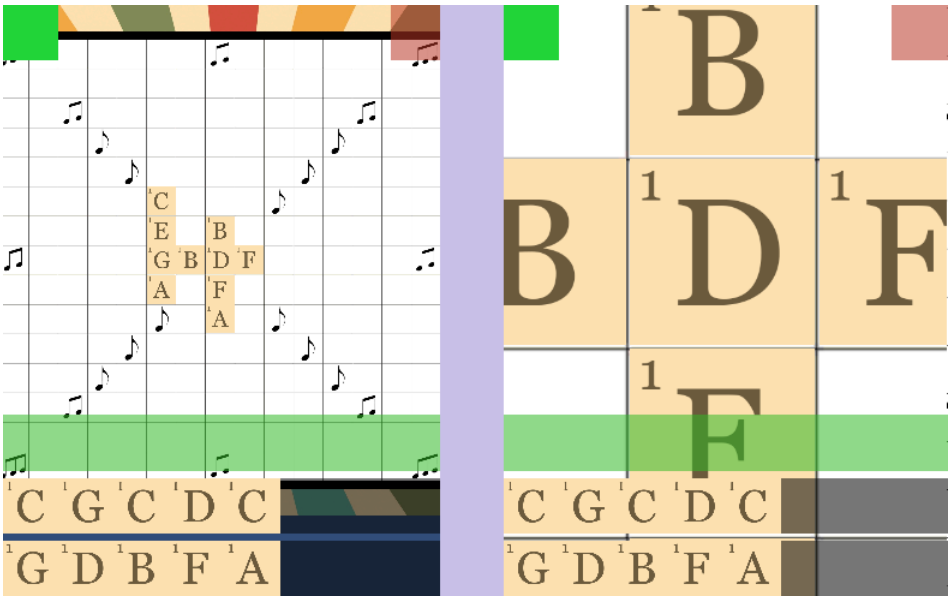


Figura 14: Alejar y acercar la vista al tablero

6.3. Mecánicas del Juego

La partida se juega por turnos. Cada turno tiene hasta dos fases: Una fase de juego y una fase de votación. Si se juega un acorde se pasa a la fase de votación, mientras que si se pasa o cambia la mano se pasa directamente a la fase de juego del turno del siguiente jugador.

El juego termina si se acaban las fichas en el mazo y la mano del jugador al que le iba a tocar el turno, o si han pasado dos rondas sin ubicar fichas en el tablero, ya sea porque las jugadas son rechazadas o los jugadores pasan o cambian sus manos.

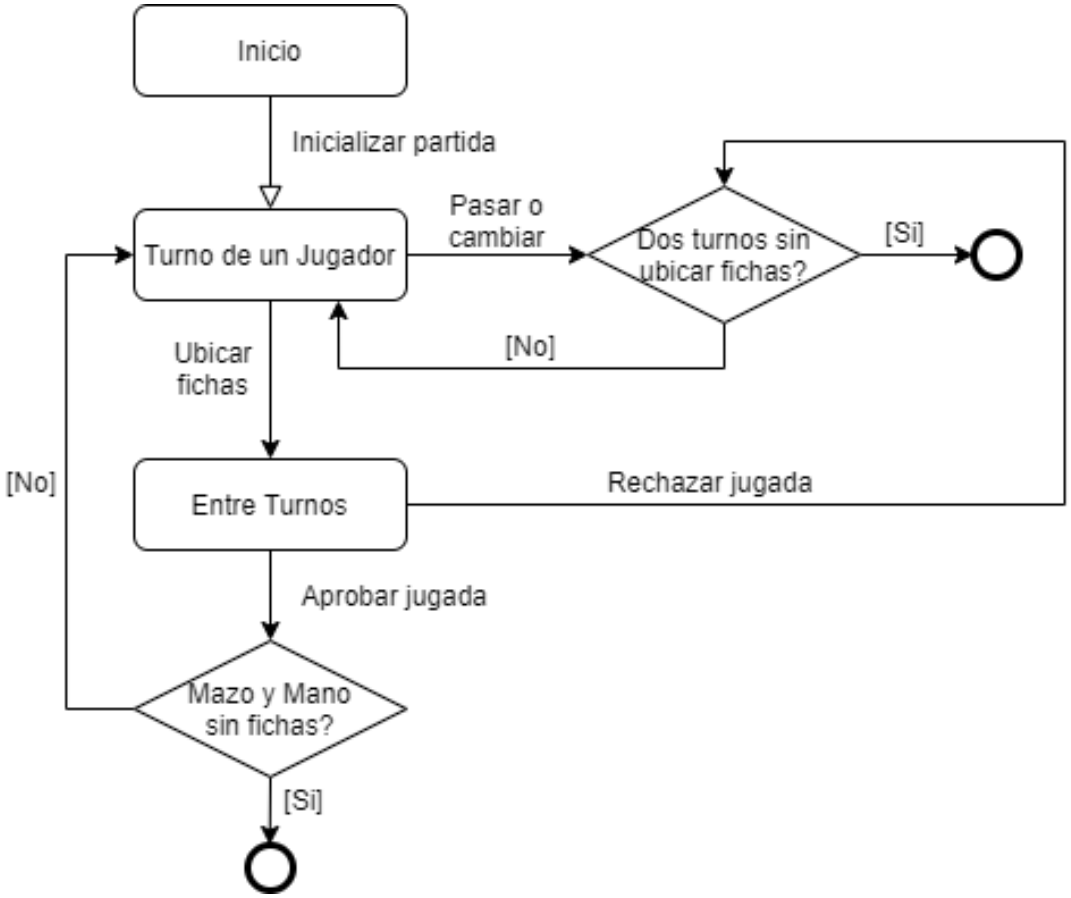


Figura 15: Flujo de turnos

6.4. Modelo de Datos

PlayFab tiene su propio sistema para manejar información persistente en el lado del servidor. Para el desarrollo de este videojuego se usó este sistema para guardar la información de los usuarios y de las partidas.

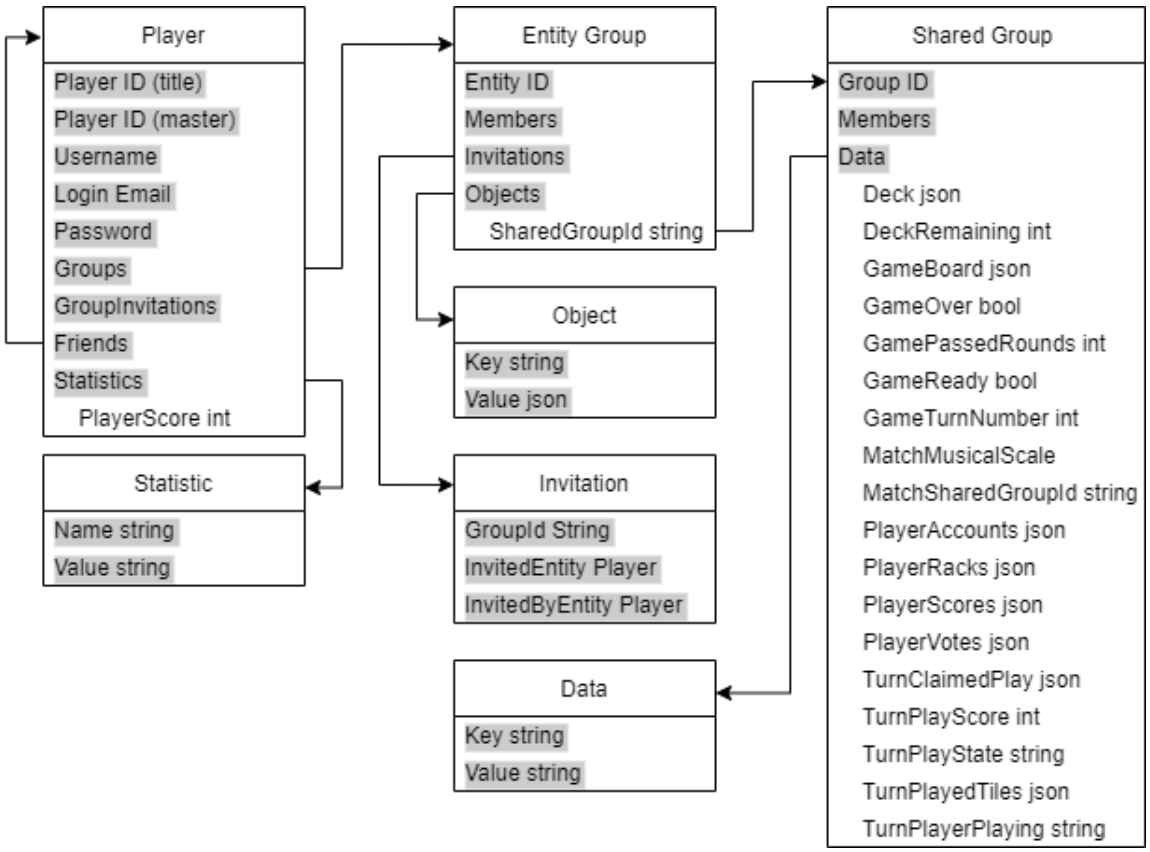


Figura 16: Modelo de datos usando PlayFab

A continuación se describen las entidades y sus parámetros.

6.4.1. Jugador

Cada jugador tiene su propia entrada. Puede acceder a los grupos a los que pertenece y a su lista de amigos. PlayFab es capaz de guardar mas información sobre los jugadores, como sus personajes e inventarios, pero los datos explicados más adelante son los que se usaron.

Player ID (title): El identificador único del la cuenta del jugador dentro del juego. En este caso, dentro de Tone Cluster.

Player ID (master): El identificador único de la cuenta del jugador en PlayFab. Si se usa el mismo correo electrónico para crear una cuenta en otro juego de PlayFab, comparten este ID.

Username: El nombre de usuario del jugador en el juego. En Tone Cluster se le muestra a los otros jugadores para identificar al jugador.

Login Email: Correo electrónico del jugador. Se usa para ingresar al juego y se puede usar para recuperar la contraseña.

Groups: Los grupos de entidad a los que pertenece el jugador. Para Tone Cluster estos grupos se usaron para guardar las partidas y sus jugadores.

GroupInvitations: Invitaciones del jugador a las partidas.

Friends: La lista de amigos del jugador. Se pueden obtener, agregar y quitar.

Statistics: Valores que el desarrollador puede personalizar en su aplicación para guardar información de los jugadores. Cada estadística tiene un nombre y un valor.

PlayerScore: Un número entero para guardar el puntaje total del jugador. Cada vez que el jugador termina una partida, el puntaje de esa partida se suma a este número. Este valor no es parte predeterminada de PlayFab.

6.4.2. Grupo de entidad

Para guardar las partidas y sus jugadores se usaron los grupos de entidad (Entity Group) [38]. Estos grupos pueden ser obtenidos por todos sus jugadores miembros. Para obtener los grupos a los que un jugador pertenece, basta con con una petición al servidor sin tener que saber nada sobre los grupos.

Una desventaja de los grupos de entidad es que solo pueden guardar hasta 5 objetos, los cuales son pares de llave-valor, y cada objeto solo puede pesar hasta 1000 bytes. Por lo que la información de la partida no se guardó usando estos grupos, si no que usando los grupos compartidos, al cual el grupo de entidad puede apuntar usando uno de sus objetos.

Entity ID: El identificador único del grupo de entidad.

Members: Los jugadores que pertenecen al grupo.

Invitations: Las invitaciones que se le envían a los jugadores para que puedan unirse al grupo. En el caso de Tone Cluster, se implementó que fueran enviadas al momento de empezar una nueva partida.

Objects: Pares de llave-valor que se pueden usar para guardar información del grupo. Un grupo solo puede tener hasta 5 objetos, y cada valor solo puede guardar hasta 1000 bytes.

SharedGroupId: Aquí se guarda el identificador del grupo compartido donde se guarda la información de la partida, para compensar que los grupos compartidos, a diferencia de los grupos de entidad, no son accesibles sin su identificador. Este es un valor personalizado para Tone Cluster y no es predeterminado de PlayFab.

6.4.3. Grupo compartido

Para guardar la información de las partidas se utilizaron los grupos compartidos (Shared Group)[39] [40]. Estos grupos pueden guardar hasta 30 pares de llave-valor, los cuales pueden pesar hasta 300kb.

Solo se puede acceder a un grupo compartido si se posee su identificador. No se puede enviar una petición al servidor para obtener los grupos compartidos a los que pertenece un jugador sin conocer los identificadores de los grupos. Debido a esto, se eligió usar grupos de entidad, que si pueden ser accedidos sin la necesidad de su identificador, para guardar los identificadores de los grupos compartidos.

Group ID: El identificador único del grupo compartido. Es necesario tener este identificador para acceder al grupo compartido.

Members: Jugadores pertenecientes al grupo compartido. No se usó al ser redundante con el grupo de entidad.

Data: Pares de llave-valor con los cuales guardar información. En Tone Cluster, se usaron para guardar la información de las partidas, siendo sus campos personalizados para esto.

Deck: Las fichas en la reserva. Es un string json que representa un arreglo que contiene fichas.

DeckRemaining: Un número entero con la cantidad de fichas restantes en la reserva.

GameBoard: Las fichas en el tablero de jugadas anteriores. Es un json de un arreglo con las fichas, las cuales tienen su posición en los ejes X e Y, además de su nota y puntaje.

GameOver: Un booleano que indica si la partida terminó.

GamePassedRounds: Un entero que indica la cantidad de rondas seguidas en las que un jugador ha realizado una jugada que no cambia el estado del tablero. Sirve para saber si la partida debe terminar antes de tiempo debido a la falta de jugadas.

GameReady: Un booleano que indica si la partida está lista. Se incluyó para evitar la posibilidad de carreras de datos mientras la partida se inicializa.

GameTurnNumber: Un entero que indica la cantidad de rondas que han pasado.

MatchMusicalScale: Un string con la nota de la escala de la partida. Es decir, guarda la dificultad con la que se eligió jugar la partida.

MatchSharedGroupId: El identificador de grupo compartido. Se guarda aquí también para hacer más fácil moverlo entre funciones.

PlayerAccounts: Un string json de un arreglo con la información de los jugadores de la partida.

```
[Serializable]
public class MatchPublicInfo
{
    public Board GameBoard;

    public int DeckRemaining;

    public string MatchId;
    public string MatchSharedGroupId;

    public string MatchMusicalScale;
    public string MatchRoundTime;

    public PlayerAccountList PlayerAccounts;
    public PlayerScoreList PlayerScores;

    public int GameTurnNumber;
    public int GamePassedRounds;
    public bool GameReady;
    public bool GameOver;

    public string TurnPlayerPlaying;
    public string TurnPlayState;
    public TurnPlayedTileList TurnPlayedTiles;
    public string TurnClaimedPlay;
```

Figura 17: Clase de datos **MatchPublicInfo** del lado de la aplicación con la información de la partida que todos los jugadores pueden ver.

PlayerRacks: Un string json de un arreglo con las fichas en la mano de los jugadores.

PlayerScores: Un string json de un arreglo con los puntajes de los jugadores.

PlayerVotes: Un string json de un arreglo con los votos de los jugadores. Pueden ser “none”, “approve” o “reject”.

TurnPlayScore: Un número entero con el puntaje de la última jugada realizada.

TurnPlayState: Un string con el estado del turno. Es “playing” si a algún jugador le toca jugar un acorde o “voting” si le toca a los otros jugadores aprobar su jugada.

TurnPlayedTiles: Un string json de un arreglo con las fichas que componen la última jugada realizada. Este campo se utiliza mientras se está en la fase de votación.

TurnPlayerPlaying: Un string json del jugador al cual le toca su turno, sea durante la fase de colocar un nuevo acorde o si se está votando la jugada que realizó.

6.4.4. Partidas

Cada partida se guarda en un par de clases. Un grupo de entidad y un grupo compartido. Para superar las desventajas de cada clase de grupo, se eligió usar ambos. El grupo de entidad guarda los jugadores que pertenecen a una partida y al identificador del grupo compartido, mientras que el grupo compartido guarda la información de la partida de Tone Cluster propiamente tal. De esta manera, los jugadores son capaces de acceder a todas sus partidas, y cada partida puede guardar mucha información.

7. Implementación de la Solución

7.1. Tecnologías de Implementación

El software desarrollado en esta memoria tiene dos partes: un front-end y un back-end. Sobre cada una se hicieron sus propias consideraciones y decisiones.

7.1.1. Tecnologías consideradas para el frontend

Para el desarrollo de la aplicación se consideraron tres formas de implementarlo: Unity [27], Unreal Engine [28] y realizarlo como un sitio web con HTML y javascript.

Se eligió usar un motor de desarrollo en vez de usar un framework de HTML y javascript porque los motores de desarrollo tienen un editor de escenas. Lo cual otorga más libertad al momento de modificar la interfaz de usuario. Lo cual hace más fácil modificar la interfaz para obtener una mejor usabilidad.

Entre los dos motores de juegos se consideró Unity y Unreal Engine. Unity tiene ventaja sobre Unreal Engine al momento de desarrollar juegos en 2D y es más ligero para los juegos de celular [29]. Así que se eligió Unity.

7.1.2. Tecnologías consideradas para el backend

La primera decisión sobre la implementación del back-end fue si desplegar el hardware de servidor propio o si usar un servicio de nube. Se prefirió usar un servicio de nube porque otorga más seguridad de disponibilidad, al ser menor probable que se caiga, y ahorra costo y tiempo.

La segunda decisión fue si usar un servicio de nube para desplegar un servidor propio o usar una plataforma de back-end como Servicio. Un BaaS o back-end como Servicio es una plataforma que automatiza el desarrollo del lado de back-end y se encarga de la infraestructura de nube [30]. Se prefirió el servidor como servicio para ahorrar tiempo.

Existen varias alternativas de servidor como servicio para un servidor de videojuegos [31] [32] [33]. De estas, las mejores son Firebase de Google y PlayFab de Microsoft por poseer ya hechos mecanismos de autenticación de cuentas de usuarios y tener bases de datos que permiten guardar la información una gran cantidad de usuarios de forma gratuita. Se optó por usar PlayFab en vez de Firebase por tener una mayor cantidad de documentación lista para usar para desarrollar un juego como el de esta memoria [34].

7.1.3. Unity

Para desarrollar el juego propiamente tal se usó Unity. Esta es una plataforma de desarrollo de aplicaciones con la cual se pueden desarrollar juegos para múltiples plataformas. Incluyendo dispositivos móviles.

Una de las ventajas de Unity es que posee múltiples paquetes de desarrollo en un amplio ecosistema de software. Lo cual permite acelerar el desarrollo de juegos y aplicaciones, al poder usar paquetes ya construidos para distintos usos.

7.1.4. PlayFab

La conexión entre los jugadores y el almacenamiento de las partidas del juego se manejó con PlayFab. Playfab es una herramienta gratuita con opciones de pago desarrollada por Microsoft que permite almacenar información de usuarios para juegos en línea.

El uso de PlayFab es necesario para que los jugadores puedan jugar múltiples partidas asíncronas, ya que no bastará realizar una conexión de Cliente/Servidor entre los dispositivos de los jugadores porque el juego debe permanecer aún si todos los jugadores se han desconectado.

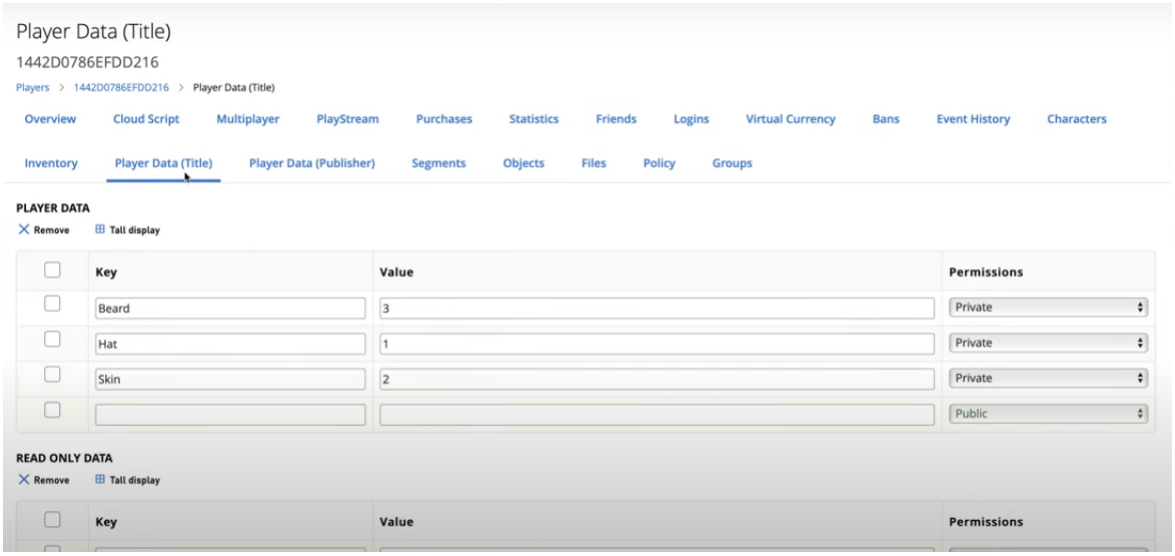


Figura 18: Playfab. Ejemplo de información de usuarios

PlayFab permite la creación de grupos de jugadores. En estos grupos se puede guardar información. Con esta funcionalidad, se pueden crear partidas asíncronas con información persistente aun cuando todos los jugadores están desconectados y a la cual todos los jugadores tienen acceso.

Otras funcionalidades de PlayFab incluyen el manejo de cuentas de usuario (ingreso, registro y recuperación de contraseña), poder guardar información de cada usuario, manejar

una lista de amigos por cada usuario y manejar invitaciones a grupos. Funcionalidades que no se usaron incluyen servidores de multijugador (solo hasta 750 horas gratis), matchmaking, tablas de puntajes mas altos y chat de voz. PlayFab también ofrece sistemas de análisis de datos en tiempo real y opciones de monetización. [35]

7.2. Servicios de cliente y servidor

Los cambios de interfaz y las acciones del usuario llevan a que la aplicación y los servidores tengan que interactuar para actualizar la interfaz en el cliente y modificar información en el servidor. Para hacer esto se utiliza una arquitectura de cliente-servidor. Al mismo tiempo que la interfaz se actualiza siguiendo una arquitectura de Modelo-Vista- Controlador (MVC). El MVC beneficia la usabilidad al facilitar cambiar la interfaz [25].

En la aplicación, los controladores mandan peticiones a los repositorios. Que obtienen el modelo desde el servidor. Una vez que el controlador recibe la respuesta de la petición, actualiza la vista con la información recibida.

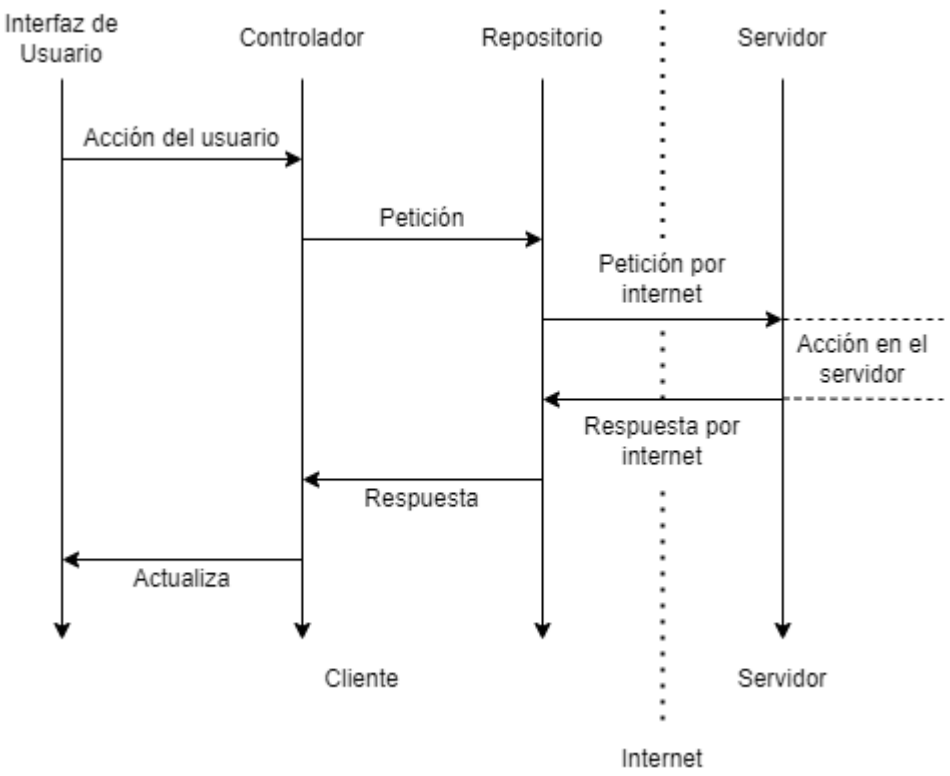


Figura 19: Peticiones entre el cliente y el servidor

7.2.1. Servicios predeterminados de PlayFab

Estos son servicios que PlayFab ya tiene implementados. Le otorgan a las aplicaciones que usan PlayFab una forma directa de interactuar con la información guardada en el servidor.

Estos servicios cubren casos de uso comunes para los juegos en línea [22].

Ingreso a la cuenta

Esta transacción cubre el requisito de software RS 04: Transacción de ingreso de cuenta.

El ingreso a la cuenta en la aplicación es manejado por el controlador **LoginController**. El cual llama al repositorio **PlayFabLoginEmail**. El repositorio usa la solicitud predeterminada de PlayFab **LoginWithEmailAddressRequest**. La cual tiene como parámetros a una dirección de correo electrónico, una contraseña, y el id de título del juego.

Un llamado a esta solicitud retorna el resultado del ingreso. Si el ingreso es correcto, entonces el servidor autoriza a la aplicación a realizar otras solicitudes sobre información asociada a la cuenta ingresada.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using PlayFab;
using PlayFab.ClientModels;

public class LoginController : ClientModelsController
{
    LoginEmail EmailLoginRepository = new LoginEmail();
    UnityEvent<string> eventOnLogin = new UnityEvent<string>();

    public void Start()
    {
        SetRepository(EmailLoginRepository);
    }

    public void SubscribeEmailLogin(UnityAction<string> call, string Email, string Password)
    {
        eventOnLogin.AddListener(call);
        EmailLoginRepository.SubscribeLoginWithEmail(OnEmailLogin, Email, Password);
    }

    private void OnEmailLogin(LoginResult result)
    {
        string Username = result.InfoResultPayload.AccountInfo.Username;
        string MasterPlayerId = result.PlayFabId;
        string TitlePlayerId = result.InfoResultPayload.AccountInfo.TitleInfo.TitlePlayerAccount.Id;

        PlayerAccount UserAccount = new PlayerAccount(Username, MasterPlayerId, TitlePlayerId);
        GlobalPlayerData.SaveUserAccount(UserAccount);

        eventOnLogin.Invoke(TitlePlayerId);
    }
}
```

Figura 20: Controlador de ingreso a la cuenta

Registrar una cuenta

Esta transacción cubre el requisito de software RS 07: Transacción de creación de cuenta.

El registro de cuentas en la aplicación es manejado por el controlador **AccountRegisterController**. El cual llama al repositorio **PlayFabAccountRegister**. El repositorio usa la

solicitud predeterminada de PlayFab **RegisterPlayFabUserRequest**. La cual tiene como parámetros a una dirección de correo electrónico, una contraseña, y el id de título del juego.

Un llamado a esta solicitud retorna el resultado del registro. Si el registro es correcto, entonces el servidor crea una nueva cuenta con la información entregada.

```
public class PlayFabLoginEmail : PlayFabClientModelsRepository
{
    UnityEvent<LoginResult> eventOnLogin = new UnityEvent<LoginResult>();

    public void SubscribeLoginWithEmail(UnityAction<LoginResult> call, string Email, string Password)
    {
        eventOnLogin.AddListener(call);
        LoginUsingEmail(Email, Password);
    }

    public void LoginUsingEmail(string argEmail, string argPassword)
    {
        var InfoRequest = new GetPlayerCombinedInfoRequestParams
        {
            GetUserAccountInfo = true
        };

        var request = new LoginWithEmailAddressRequest
        {
            Email = argEmail,
            Password = argPassword,
            InfoRequestParameters = InfoRequest
        };

        PlayFabClientAPI.LoginWithEmailAddress(request, OnLoginSuccess, OnError);
    }

    private void OnLoginSuccess(LoginResult result)
    {
        eventOnLogin.Invoke(result);
    }
}
```

Figura 21: Repositorio de ingreso a la cuenta

Restablecer la contraseña

Esta transacción cubre el requisito de software RS 10: Transacción de restablecimiento de contraseña.

El restablecimiento de contraseñas en la aplicación es manejado por el controlador **PasswordResetController**. El cual llama al repositorio **PlayFabPasswordResetter**. El repositorio usa la solicitud predeterminada de PlayFab **SendAccountRecoveryEmailRequest**. La cual tiene como parámetros a una dirección de correo electrónico y el id de título del juego.

Un llamado a esta solicitud retorna el resultado de haber enviado el correo. Si el envío es correcto, entonces el servidor envía un correo electrónico al usuario con las instrucciones para restablecer su contraseña.

Pedir la lista de amigos

Esta transacción cubre el requisito de software RS 14: Transacción para obtener los amigos del usuario.

El llamado al servidor para pedir la información de los amigos de un usuario es manejado en la aplicación por el controlador **FriendsGetController**. El cual llama al repositorio **PlayFabFriendsGetter**. El repositorio usa la solicitud predeterminada de PlayFab **GetFriendsListRequest**. La cual no tiene parámetros obligatorios, pero debe ser llamada después de que el usuario haya iniciado sesión. Tras lo cual obtiene la lista de amigos asociada a la cuenta ingresada.

Un llamado a esta solicitud retorna el resultado de haber agregado al amigo. Si el envío es correcto, entonces el servidor agrega al amigo a la lista de amigos del usuario.

Agregar un nuevo amigo

Esta transacción cubre el requisito de software RS 16: Transacción para agregar amigos.

El llamado al servidor para agregar un nuevo amigo es manejado en la aplicación por el controlador **FriendAddController**. El cual llama al repositorio **PlayFabFriendAdder**. El repositorio usa la solicitud predeterminada de PlayFab **AddFriendRequest**. La cual recibe como parámetro el nombre de usuario del amigo por agregar.

Un llamado a esta solicitud causa una respuesta que contiene un arreglo de instancias de la clase predefinida de PlayFab FriendInfo. Cada instancia de FriendInfo posee el id maestro y el nombre de usuario del usuario amigo.

Quitar un amigo

Esta transacción cubre el requisito de software RS 18: Transacción para quitar un amigo.

El llamado al servidor para quitar un amigo de la lista de amigos del usuario es manejado en la aplicación por el controlador **FriendRemoveController**. El cual llama al repositorio **PlayFabFriendRemover**. El repositorio usa la solicitud predeterminada de PlayFab **RemoveFriendRequest**. La cual recibe como parámetro el id maestro de la cuenta del amigo por quitar.

Un llamado a esta solicitud retorna el resultado de haber quitado al amigo. Si el envío es correcto, entonces el servidor quita al amigo de la lista de amigos del usuario.

Pedir las estadísticas del usuario

Esta transacción cubre el requisito de software RS 27: Petición para pedir el puntaje general del jugador.

El llamado al servidor para obtener el puntaje general del jugador es manejado en la aplicación por el controlador **PlayerStatisticsGetController**. El cual llama al repositorio

PlayFabPlayerStatisticsGetter. El repositorio usa la solicitud predeterminada de PlayFab **GetPlayerStatisticsRequest**. La cual no tiene parámetros obligatorios, pero debe ser llamada después de que el usuario haya iniciado sesión. Tras lo cual obtiene las estadísticas asociadas a la cuenta ingresada.

Un llamado a esta solicitud causa una respuesta que contiene un arreglo de instancias de la clase predefinida de PlayFab StatisticValue con todas las estadísticas guardadas del jugador. Cada instancia de StatisticValue posee el nombre de la estadística que contiene y su valor. La estadística PlayerScore tiene como valor el puntaje general del jugador.

Pedir las invitaciones recibidas

Esta transacción cubre el requisito de software RS 29: Transacción para pedir las invitaciones pendientes

El llamado al servidor para obtener las invitaciones pendientes del usuario es manejado en la aplicación por el controlador **InvitationsGetController**. El cual llama al repositorio **PlayFabInvitationsGetter**. El repositorio usa la solicitud predeterminada de PlayFab **ListMembershipOpportunitiesRequest**. La cual no tiene parámetros obligatorios, pero debe ser llamada después de que el usuario haya iniciado sesión. Tras lo cual obtiene las invitaciones a grupos de entidad pendientes asociadas a la cuenta ingresada.

Un llamado a esta solicitud causa una respuesta que contiene un arreglo de instancias de la clase predefinida de PlayFab GroupInvitation con todas las invitaciones pendientes del usuario. Cada instancia de GroupInvitation posee entidades correspondientes al grupo, al usuario que invita y al usuario que es invitado.

Aceptar una invitación

Esta transacción cubre el requisito de software RS 32: Transacción para aceptar la invitación a una partida.

El llamado al servidor para aceptar una invitación a un grupo de entidad es manejado en la aplicación por el controlador **InvitationChoiceController**. El cual llama al repositorio **PlayFabInvitationAccepter**. El repositorio usa la solicitud predeterminada de PlayFab **AcceptGroupInvitationRequest**. La cual tiene como parámetro el grupo de entidad al que el usuario acepta unirse.

Un llamado a esta solicitud causa una respuesta vacía EmptyResponse.

Rechazar una invitación

Esta transacción cubre el requisito de software RS 34: Transacción para rechazar una invitación.

El llamado al servidor para rechazar una invitación a un grupo de entidad es manejado en la aplicación por el controlador **InvitationChoiceController**. El cual llama al repositorio

PlayFabInvitationDecliner. El repositorio usa la solicitud predeterminada de PlayFab **RemoveGroupInvitationRequest**. La cual tiene como parámetro el grupo de entidad al que el usuario rechaza unirse y su propia entidad de cuenta.

Un llamado a esta solicitud causa una respuesta vacía EmptyResponse.

Ver las partidas activas

Esta transacción cubre el requisito de software RS 36: Transacción de pedir partidas activas.

El llamado al servidor para pedir las partidas activas del usuario es manejado en la aplicación por el controlador **GroupsGetController**. El cual llama al repositorio **PlayFabGroupsGetter**. El repositorio usa la solicitud predeterminada de PlayFab **ListMembershipRequest**. La cual no tiene parametros obligatorios, pero debe ser llamada después de que el usuario haya iniciado sesión. Tras lo cual obtienen los grupos de entidad en los cuales el usuario es integrante.

Un llamado a esta solicitud causa una respuesta que contiene un arreglo de instancias de la clase predefinida de PlayFab GroupWithRoles con los grupos a los que pertenece el usuario. Cada instancia de GroupWithRoles posee un objeto al que representa

7.2.2. Servicios personalizados de PlayFab

Además de los servicios predeterminados de PlayFab, es posible implementar servicios personalizados que puedan ser llamados desde la aplicación al servidor. El código de estos servicios se escribe en el sitio de PlayFab directamente desde el navegador en un servicio de automatización, donde se ocupa el lenguaje de programación hecho a la medida para esta funcionalidad CloudScript, el cual está basado en JavaScript.

Obtener la información de una partida

En el lado de la aplicación, la clase **MatchInfoLoadController** llama a la clase **MatchInfoGetRequestHandler** para que llame a las funciones personalizadas **getMatchPrivateInfo** y **getMatchPublicInfo**. Las cuales se encargan de recibir en el lado de servidor las peticiones de enviar al cliente la información de una partida. Ambas funciones tienen como argumento el identificador del grupo compartido de la partida.

Enviar una jugada

En el lado de la aplicación, la clase **MoveSendController** llama a la clase **PlayFabMoveSender** para que llame a la función personalizada **receiveMove**. La cual se encarga de recibir en el lado de servidor las jugadas hechas. receiveMove tiene como argumentos el identificador del grupo compartido, las fichas jugadas en el turno, la mano restante del jugador y el puntaje de la jugada realizada.

```
var request = new ExecuteCloudScriptRequest
{
    FunctionName = "receiveMove",
    FunctionParameter = new
    {
        SharedGroupId = MatchSharedGroupId,
        PlayedTiles = argPlayedTiles,
        Rack = argRackTiles,
        TurnScore = argTurnScore
    }
};
PlayFabClientAPI.ExecuteCloudScript(request, OnSendMoveSuccess, OnError);
```

Figura 22: PlayFabMoveSender. Petición al servidor para ejecutar la función receiveMove

```
handlers.receiveMove = function(args) {
    if (args && args.SharedGroupId && args.PlayedTiles && args.Rack && args.TurnScore) {
        var SharedGroupId = args.SharedGroupId;

        var Keys = [
            "TurnPlayerPlaying"
        ];
        var ResultStrings = GetSharedGroupObject(SharedGroupId, Keys);
        var ValueObjects = ConvertDictJsonToObject(ResultStrings);
        var TurnPlayerPlaying = ValueObjects["TurnPlayerPlaying"];

        if (currentPlayerId != TurnPlayerPlaying) {
            log.debug("No es su turno");
            return;
        }

        var PlayedTiles = args.PlayedTiles;
        var Rack = args.Rack;

        SetRack(SharedGroupId, currentPlayerId, Rack);
        SetTurnMove(SharedGroupId, currentPlayerId, PlayedTiles);
        RefillRack(SharedGroupId, currentPlayerId);

        SetSharedGroupObject(SharedGroupId, "TurnPlayScore", args.TurnScore);
    }
}
```

Figura 23: Función **receiveMove** en el lado del servidor

Pasar de turno

En el lado de la aplicación, la clase **MoveSendController** llama a la clase **PlayFabMovePasser** para que llame a la función personalizada **passMove**. La cual se encarga de recibir en el lado de servidor el movimiento de pasar el turno. La función **passMove** tiene como argumentos el identificador del grupo compartido, y el identificador del jugador que pasa su turno.

```
SkipTurnMove = function (SharedGroupId, MasterPlayerId) {
    var Keys = [
        "PlayerAccounts",
        "TurnPlayerPlaying"
    ];

    var ResultStrings = GetSharedGroupObject(SharedGroupId, Keys);
    var ValueObjects = ConvertDictJsonToObject(ResultStrings);

    //BEGIN BODY
    var PlayerAccounts = ValueObjects["PlayerAccounts"].PlayerAccounts;
    var TurnPlayerMasterPlayerId = ValueObjects["TurnPlayerPlaying"];

    //Set next player
    NextPlayerPlaying = GetNextPlayerMasterPlayerId(PlayerAccounts, TurnPlayerMasterPlayerId);
    ValueObjects["TurnPlayerPlaying"] = NextPlayerPlaying;
    //END BODY

    var ValueStrings = ConvertDictObjectToJson(ValueObjects, Keys);
    SetSharedGroupObjectImpl(SharedGroupId, ValueStrings);

    //Check if match ends from an empty rack
    TryEmptyRackAndDeckMatchEnding(SharedGroupId);
}
```

Figura 24: Función **skipTurnMove** usada por **handlers.passMove** y **handlers.shuffleRack**. Cambia al jugador activo y revisa si la partida terminó

Revolver la mano

En el lado de la aplicación, la clase **MoveSendController** llama a la clase **PlayFabRackShuffler** para que llame a la función personalizada **shuffleRack**. La cual se encarga de recibir en el lado de servidor la jugada de pasar el turno. passMove tiene como argumentos el identificador del grupo compartido, y el identificador del jugador que pasa su turno.

```
handlers.shuffleRack = function(args) {
    if (args && args.SharedGroupId && args.Rack) {
        var SharedGroupId = args.SharedGroupId;

        var Keys = [
            "TurnPlayerPlaying",
        ];

        var ResultStrings = GetSharedGroupObject(SharedGroupId, Keys);
        var ValueObjects = ConvertDictJsonToObject(ResultStrings);

        //BODY
        var TurnPlayerPlaying = ValueObjects["TurnPlayerPlaying"];

        if (currentPlayerId != TurnPlayerPlaying) {
            log.debug("No es su turno");
            return;
        }

        var Rack = args.Rack;

        EmptyRack(SharedGroupId, currentPlayerId, Rack);

        var ValueStrings = ConvertDictObjectToJson(ValueObjects, Keys);
        SetSharedGroupObjectImpl(SharedGroupId, ValueStrings);

        RefillRack(SharedGroupId, currentPlayerId);

        SkipTurnMove(SharedGroupId);
    }
}
```

Figura 25: Función **shuffleRack** en el lado del servidor. Llama a **EmptyRack** para descartar la mano y luego a **RefillRack** para robar una mano nueva

Aprobar el acorde de otro jugador

En el lado de la aplicación, la clase **TurnPlayVoteController** llama a la clase **PlayFabApproveMoveVoter** para que llame a la función personalizada **voteApproveMove**. La cual se encarga de recibir en el lado de servidor la petición de aprobar el último acorde jugado.

Rechazar el acorde de otro jugador

En el lado de la aplicación, la clase **TurnPlayVoteController** llama a la clase **PlayFabRejectMoveVoter** para que llame a la función personalizada **voteRejectMove**. La cual se encarga de recibir en el lado de servidor la petición de rechazar el último acorde jugado.

Inicializar una partida

En el lado de la aplicación, la clase **MatchCreateController** llama a la clase **PlayFabMatchValuesInitializer** para que llame a la función personalizada **initializeToneClusterMatchValues**. La cual se encarga de recibir en el lado de servidor la petición de inicializar los datos de una nueva partida de Tone Cluster.

Enviar un mensaje a soporte técnico

En el lado de la aplicación, la clase **DiscordMessageSendController** llama a la clase **PlayFabDiscordMessageSender** para que llame a la función personalizada **sendDiscordMessage**. La cual se encarga de recibir en el lado de servidor la petición de enviar un mensaje por Discord. Esta interacción se usa para enviar los mensajes al soporte técnico.

7.3. Implementación del Servidor

7.3.1. CloudScript

CloudScript es el lenguaje de programación usado por PlayFab para ejecutar funcionalidades personalizadas en el lado de servidor. Permite que luego el lado de cliente llame la ejecución de funciones implementadas al servidor.

Para el lado de servidor de Tone Cluster se implementaron 45 funciones en CloudScript. Aquí se describen:

Inicializar una partida

handlers.initializeToneClusterMatchValues: Inicializa los valores de una nueva partida de ToneCluster.

```
handlers.initializeToneClusterMatchValues = function(args, context) {
    if (args && args.SharedGroupId && args.PlayerAccounts && args.MatchMusicalScale) {
        var SharedGroupId = args.SharedGroupId;
        var MatchMusicalScale = args.MatchMusicalScale;
        var PlayerAccounts = args.PlayerAccounts;

        InitializeDeck(SharedGroupId, MatchMusicalScale); //key Mazo
        InitializeBoard(SharedGroupId); //key Tablero
        InitializePlayers(SharedGroupId, PlayerAccounts); //keys Jugadores
        InitializeMatchPermanentInfo(SharedGroupId, MatchMusicalScale, 0); //keys Info Permanente
        InitializeTurnInfo(SharedGroupId, PlayerAccounts); //keys Estado del Turno
        InitializeGameInfo(SharedGroupId); //keys Info hasta ahora

        DealInitialRacks(SharedGroupId);
        SetSharedGroupStringify(SharedGroupId, "GameReady", true);
    }
}
```

Figura 26: La función **initializeToneClusterMatchValues** inicializa los datos de la partida

InitializeDeck: Inicializa la reserva de fichas de la partida.

InitializeBoard: Inicializa el tablero vacío de la partida.

InitializePlayers: Guarda la información de los jugadores como un valor en la partida, junto a sus manos, un puntaje inicial de 0 y un voto vacío.

InitializeMatchPermanentInfo: Guarda la información permanente de la partida, esto es, el identificador del grupo y la dificultad de la partida.

InitializeTurnInfo: Inicializa la información que define cada turno, esto es, el jugador que está jugando el turno, el estado de turno (jugando o votando), el puntaje de la jugada propuesta y las fichas jugadas.

InitializeGameInfo: Inicializa la información que cambia con el paso del juego sin definir un turno, esto es, la cantidad de rondas jugadas, la cantidad de rondas seguidas sin puntaje, si la partida está lista y si la partida terminó.

Inicializar la información de los jugadores

CreatePlayerAccounts: Guarda la información de los usuarios al inicio de la partida.

CreatePlayerRacks: Crea las manos vacías de los jugadores al inicio en la información persistente de la partida.

CreatePlayerScores: Crea los puntajes de los jugadores. El valor inicial de los puntajes es cero.

CreatePlayerVotes: Crea los votos de los jugadores. El valor inicial de los votos es "None".

Inicializar las fichas

GetDeck: Entrega un arreglo de fichas para usarse de reserva.

Shuffle: Revuelve la reserva de fichas. Se le pasa un arreglo y retorna el arreglo revuelto.

DealInitialRacks: Reparte las manos iniciales de los jugadores. Por cada jugador, se retiran 5 fichas de la reserva y se insertan en su mano.

Obtener información de la partida

handlers.getMatchPublicInfo: Permite al lado del cliente obtener la información de una partida que es accesible a todos los jugadores de la misma.

handlers.getMatchPrivateInfo: Permite al lado del cliente obtener la información secreta de un jugador de la partida. Esto es, las fichas en su mano y si aprueba la última jugada hecha.

Enviar una jugada

handlers.receiveMove: Permite al jugador enviar un acorde de fichas para jugarlo.

SetTurnMove: Esta función recibe como argumento una jugada de fichas en el tablero, las cuales guarda como la jugada hecha en el turno. Luego, pasa a la fase de votación.

SetRack: Recibe como argumento el identificador de un jugador y un arreglo de fichas. Luego, hace que estas fichas sean las fichas en la mano del jugador identificado.

RefillRack: Rellena la mano de un jugador con fichas de la reserva hasta que esta mano tenga cinco fichas.

```
//BODY
ValueObjects["PlayerRacks"][MasterPlayerId] = Rack;
CurrentRackSize = ValueObjects["PlayerRacks"][MasterPlayerId].Tiles.length;

for (var i = 0; i < CurrentRackSize; i++) {
    if (ValueObjects["PlayerRacks"][MasterPlayerId].Tiles.length > 0) {
        var Tile = ValueObjects["PlayerRacks"][MasterPlayerId].Tiles.pop();
        ValueObjects["Deck"].Tiles.push(Tile);
    }
}

ValueObjects["Deck"].Tiles = Shuffle(ValueObjects["Deck"].Tiles);
```

Figura 27: Cuerpo de la función **RefillRack** usanda por **handlers.receiveMove** y **handlers.shuffleRack**. Inserta fichas de la reserva a la mano hasta que la mano esté llena

Pasar de turno

handlers.passMove: Permite al jugador pasar su turno.

SkipTurnMove: Cambia al jugador al que le toca el turno por el siguiente, a la vez que revisa si se debe terminar la partida.

Revolver las fichas de la mano

handlers.shuffleRack: Permite al jugador revolver su mano.

EmptyRack: Revuelve todas las fichas en la mano de un jugador a la reserva.

```
CurrentRackSize = ValueObjects["PlayerRacks"][MasterPlayerId].Tiles.length;
MissingTilesSize = MaxRackSize - CurrentRackSize;

for (var i = 0; i < MissingTilesSize; i++) {
    if (ValueObjects["Deck"].Tiles.length > 0) {
        var Tile = ValueObjects["Deck"].Tiles.pop();
        ValueObjects["PlayerRacks"][MasterPlayerId].Tiles.push(Tile);
    }
}
ValueObjects["DeckRemaining"] = ValueObjects["Deck"].Tiles.length;
```

Figura 28: Cuerpo de la función **EmptyRack** usanda por **handlers.shuffleRack**. Inserta fichas de la mano a la reserva hasta que la mano esté vacía

Aprobar una jugada

handlers.voteApproveMove: Le permite al jugador aprobar el último acorde jugado. Llama a la función VoteApprove.

CheckTurnPlayStateIsVoting: Revisa si la partida está en la fase de votación.

VoteApprove: Se llama cuando un jugador vota para aprobar el último acorde jugado. Llama a las funciones ApproveMove y EndVoting.

ApproveMove: Toma las fichas en la última jugada realizada y las ubica permanentemente en el tablero. Además, suma el puntaje de la jugada al puntaje del jugador.

Rechazar una jugada

handlers.voteRejectMove: Le permite al jugador votar por rechazar el último acorde jugado. Llama a la función VoteReject.

VoteReject: Se llama cuando un jugador vota para rechazar el último acorde jugado. Llama a las funciones SetRejectMove y CheckIsMoveRejected. Si el acorde es rechazado, llama a RejectMove y a EndVoting

SetRejectMove: Cambia el voto del jugador para guardar que rechaza el último acorde jugado.

```
SetRejectMove = function (SharedGroupId) {
    var Keys = [
        "PlayerVotes"
    ]; // <- CAMPOS POR CARGAR

    var ResultStrings = GetSharedGroupObject(SharedGroupId, Keys);  // <- CARGAR CAMPOS JSON
    var ValueObjects = ConvertDictJsonToObject(ResultStrings);      // <- JSON A OBJETOS

    //BEGIN BODY
    ValueObjects["PlayerVotes"][currentPlayerId] = "Reject";
    //END BODY

    var ValueStrings = ConvertDictObjectToJson(ValueObjects, Keys); // <- OBJETOS A JSON
    SetSharedGroupObjectImpl(SharedGroupId, ValueStrings);          // <- GUARDAR CAMPOS JSON
}
```

Figura 29: La función **SetRejectMove** cambia el voto del jugador

RejectMove: Revuelve las fichas de la última jugada realizada de vuelta a la reserva.

CheckIsMoveRejected: Revisa los votos de los jugadores para ver si aprueban el último acorde jugado. Si todos votan por rechazar, llama a RejectMove para rechazar la jugada.

Acabar el turno

EndVoting: Termina la fase de votación para comenzar la siguiente fase de juego. Deshace los votos, cambia el jugador activo y cambia el estado de juego a "Playing".

GetNextPlayerMasterId: Obtiene el identificador del jugador al que le toca el próximo turno.

TryEmptyRackAndDeckMatchEnding: Revisa si la mano de un jugador y la reserva están vacías de fichas para terminar la partida.

Puntajes totales de los jugadores

handlers.initializePlayerTitleScore: Inicializa el puntaje total de un jugador con valor cero. Esta función se ejecuta cuando el jugador registra una nueva cuenta.

AddFinalScores: Suma los puntajes finales de la partida de cada jugador a sus respectivos puntajes totales asociados a sus cuentas.

Manejo de JSON

ConvertDictObjectToJson: Toma un diccionario de objetos CloudScript y los convierte en un diccionario de strings JSON.

ConvertDictJsonToObject: Toma un diccionario de strings JSON y los convierte en un diccionario de objetos CloudScript.

IsJsonString: Revisa si un string es un string JSON.

Cargar y guardar campos de una partida

SetSharedGroupStringify: Guarda un campo en un grupo compartido. Recibe una llave como nombre del campo y un objeto como valor, que debe ser convertido en JSON antes de guardarse.

```
SetSharedGroupStringify = function(argSharedGroupId, Key, DataObject) {
    var DataString = JSON.stringify(DataObject);
    SetSharedGroupObject(argSharedGroupId, Key, DataString);
}

SetSharedGroupObject = function(argSharedGroupId, Key, Data) {
    var DataPair = {
        [Key] : Data
    };
    SetSharedGroupObjectImpl(argSharedGroupId, DataPair);
}

SetSharedGroupObjectImpl = function(argSharedGroupId, argDataPairs) {
    try {
        var request = {
            SharedGroupId : argSharedGroupId,
            Data : argDataPairs
        };
        server.UpdateSharedGroupData(request);
    } catch (e) { log.debug(e); throw e; }
}
```

Figura 30: La función **SetSharedGroupObject** y similares las pueden usar las otras funciones del lado del servidor para cambiar los valores de las partidas.

SetSharedGroupObject: Guarda un campo en un grupo compartido. Recibe una llave como nombre del campo y un string JSON como valor.

SetSharedGroupObjectImpl: Guarda un diccionario de valores en JSON como campos en un grupo compartido.

GetSharedGroupObject: Carga un diccionario de valores en JSON desde los campos de un grupo compartido.

```
GetSharedGroupObject = function(argSharedGroupId, argKeys) {
    try {
        var request = {
            SharedGroupId : argSharedGroupId,
            Keys : argKeys
        };
        var SharedGroupDataResponse = server.GetSharedGroupData(request);
        var SharedGroupData = SharedGroupDataResponse.Data;
        return SharedGroupData;
    } catch (e) { log.debug(e); throw e; }
}
```

Figura 31: La función **GetSharedGroupObject** puede ser llamada por las otras funciones del lado del servidor para obtener los valores de las partidas.

Soporte técnico

handlers.sendDiscordMessage: Envía un mensaje a una conversación de la aplicación de mensajería Discord.

7.3.2. Reglas

Las "Reglas" [42] son instrucciones que se ejecutan en respuesta a eventos detectados desde el lado del servidor. Pueden usarse, por ejemplo, para ejecutar una función a la misma hora todos los días. En este proyecto solo se usó una regla.

Inicializar el puntaje total de un jugador

La inicialización del puntaje total de un jugador es realizada por una regla que se ejecuta cuando un usuario crea una cuenta nueva. Esta función solo se ejecuta una vez por cuenta. Al ejecutarse, el puntaje total del jugador se inicializa con valor 0.

7.4. Implementación del Cliente

Unity utiliza objetos para representar las entidades que se encuentran en una escena. Estos objetos se organizan en una jerarquía de padres-hijos y pueden contener Scripts como componentes que implementan funcionalidad. A continuación se explican los nodos en la escena de la partida y sus Scripts.

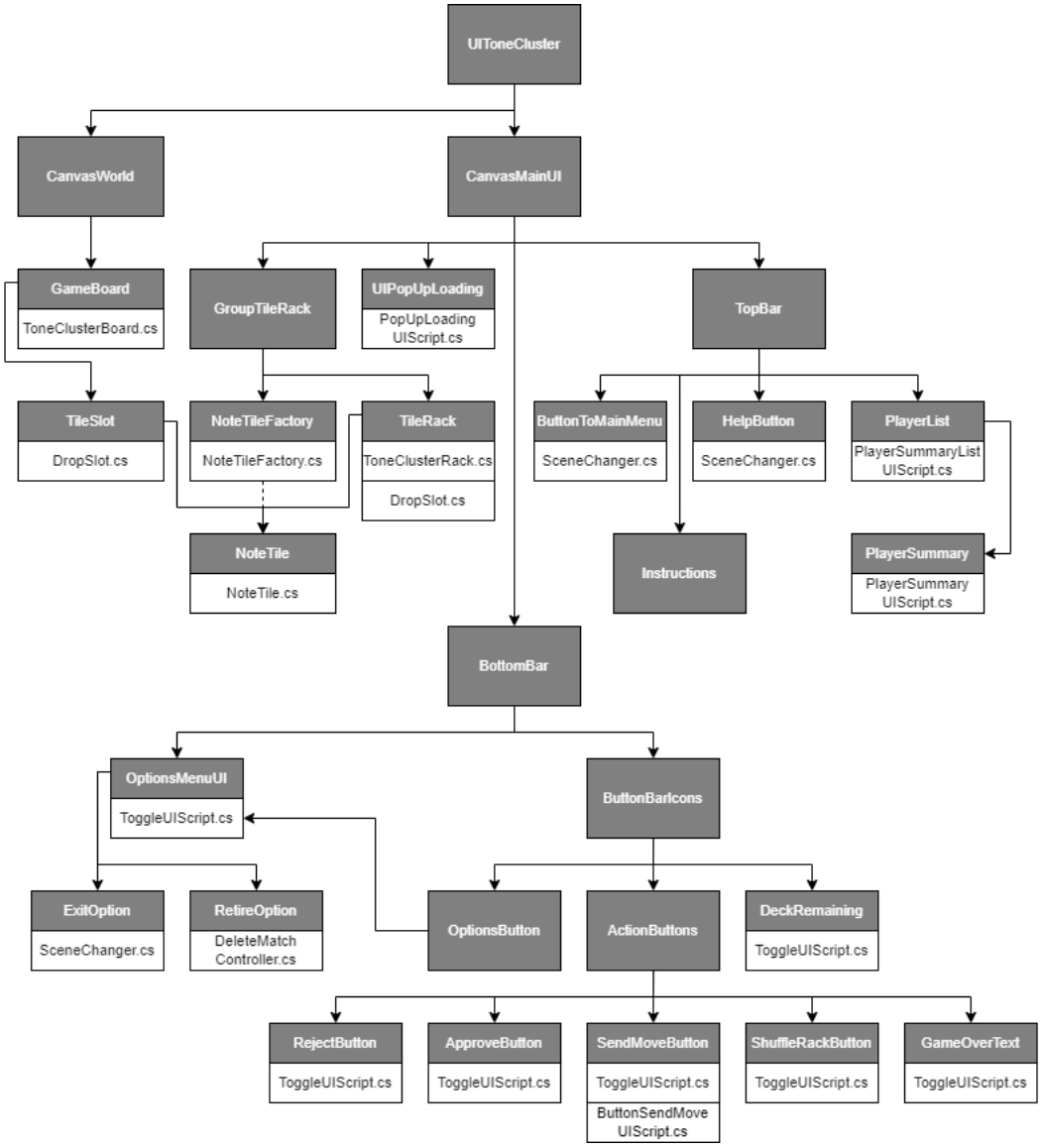


Figura 32: Diagrama de Clases. Objetos en la escena

UIToneCluster: Contiene a todas las demás clases.

CanvasWorld: Contiene un componente de Unity Canvas, con el modo de render World Space. Este canvas se usa para mostrar un area del tablero del juego, la cual el usuario puede mover o cambiar de tamaño.

CanvasMainUI: Contiene un componente de Unity Canvas, con el modo de render Screen

Space - Overlay. Este canvas muestra los elementos fijos de la interfaz de usuario, como las fichas en la mano del jugador y los puntajes de los jugadores.

GameBoard y **ToneClusterBoard.cs**: Implementan la funcionalidad necesaria para organizar en la interfaz la información del tablero de juego, los espacios donde se ponen las fichas y las que ya están puestas. Puede tomar la información del modelo y la cargarla o entregar la información de las fichas ubicadas.

DropSlot.cs: Este Script implementa la funcionalidad necesaria para que un objeto pueda ser un contenedor de fichas. Lo implementan el objeto que representa la mano del jugador y los espacios del tablero.

TileSlot: Los espacios en el tablero GameBoard donde se ubican las fichas ubicadas por los jugadores. Su script DropSlot.cs implementa la funcionalidad para recibir una ficha.

```
public class DropSlot : MonoBehaviour, IDropHandler
{
    public GameObject LastItem;
    public string ScoreModifier = "*1";

    public void OnDrop(PointerEventData eventData)
    {
        if (transform.childCount == 0)
        {
            LastItem = DragHandler.itemDragging;
            LastItem.transform.SetParent(transform);
            LastItem.transform.position = transform.position;
        }
    }
```

Figura 33: Código de DropSlot.cs que ubica una ficha en el espacio cuando se suelta encima

ToggleUIScript.cs: Permite activar y desactivar un objeto de la escena para que aparezca o desaparezca de la interfaz.

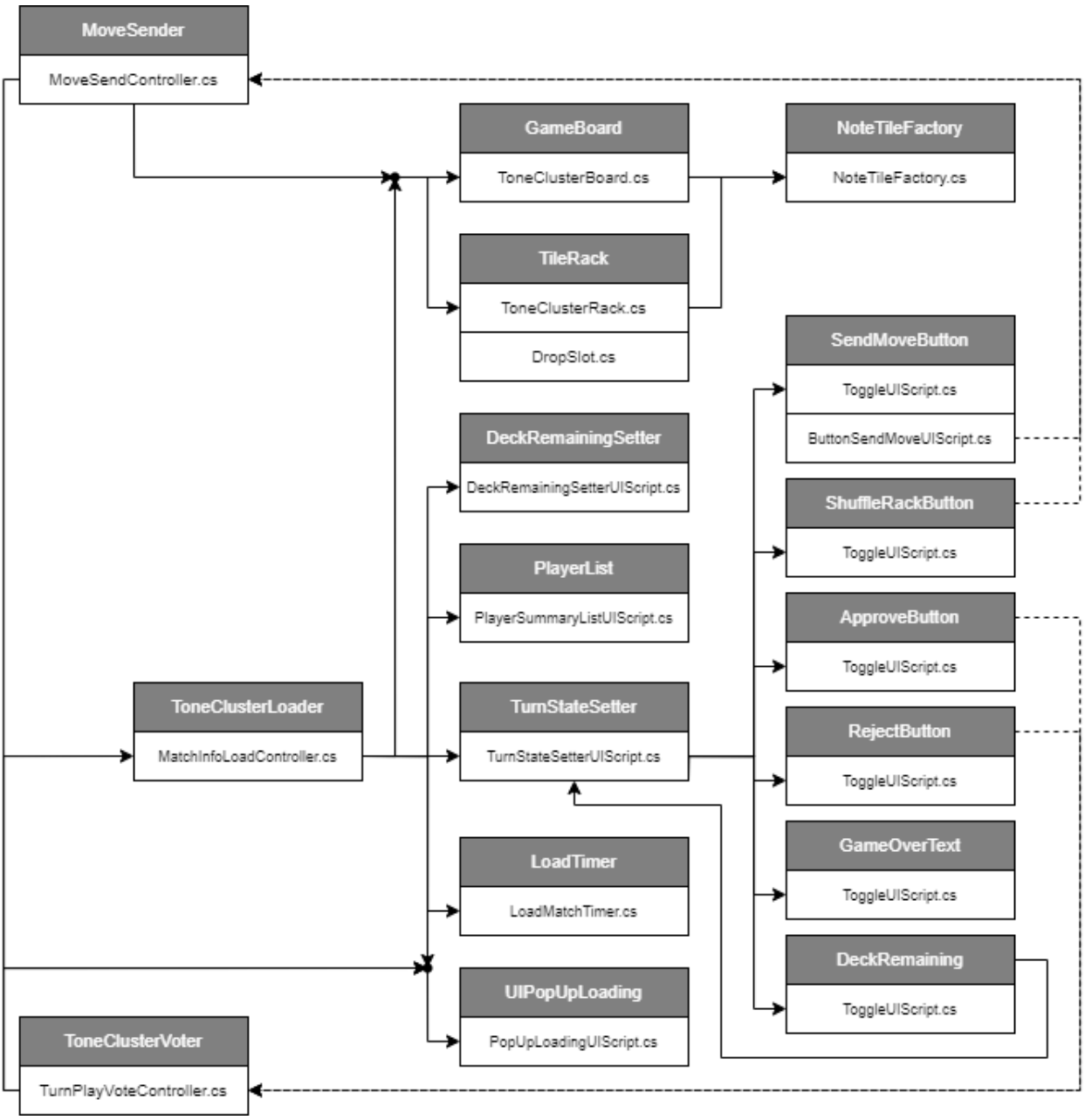


Figura 34: Diagrama de Clases. Controladores de jugadas y elementos de la interfaz.

GroupTileRack: Contiene a los objetos de la interfaz en la zona de la mano del jugador.

TileRack y ToneClusterRack.cs: Contienen las fichas que el jugador tiene en la mano. El Script puede cargar las fichas desde el modelo para que se vean en la interfaz de usuario o entregar un arreglo de las fichas ya ubicadas.

NoteTileFactory y **NoteTileFactory.cs**: Objeto que crea las fichas, necesario por como funciona la instanciación en Unity.

```
public void LoadRack()
{
    List<string> tiles = rack.GetTiles();
    foreach (string tile in tiles)
    {
        AddTile(tile);
    }
}

public void AddTile(string TileName)
{
    GameObject NewTile = tileFactory.CreateTile(TileName);
    NewTile.transform.SetParent(transform);
    NewTile.transform.localScale = new Vector3(1.0f, 1.0f, 1.0f);
}
```

Figura 35: Código de TileRack.cs para cargar las fichas en la mano del jugador

NoteTile y **NoteTile.cs**: Las fichas que pueden estar en la mano o en el tablero. Cada ficha tiene como valores su nota y su puntaje, además de un booleano que indica si la ficha está fija en su lugar, para las fichas que no son parte de la mano del jugador al cual le toca su turno.

TopBar: Contiene a los objetos de la interfaz en la zona superior de la pantalla.

SceneChanger.cs: Permite cambiar de una escena a otra.

BackToMainMenu: El botón que le permite al usuario volver al menú principal, para lo cual implementa al Script SceneChanger.cs.

HelpButton: El botón que le permite al usuario ir a la interfaz de ayuda, para lo cual implementa al Script SceneChanger.cs.

Instructions: Objeto que contiene una linea de texto que explica qué acordes se deben jugar en la partida.

PlayerList y **PlayerSummaryListUIScript.cs**: El objeto en la interfaz que contiene la información de los jugadores. Por cada jugador, PlayerList crea un objeto PlayerSummary y luego destaca al jugador al cual le toca el turno.

PlayerSummary y **PlayerSummaryUIScript.cs**: El objeto en la interfaz que contiene la información de un jugador, del cual muestra su nombre y su puntaje. Además, se puede destacar en caso de que tenga la información del jugador activo.

BottomBar: Contiene a los objetos de la interfaz en la zona inferior de la pantalla.

OptionsMenuUI: Contiene los objetos que representan botones en el menú de opciones.

ExitOption: Un botón que el usuario puede apretar para volver al menú principal.

RetireOption y **DeleteMatchController.cs**: El botón que el usuario puede apretar para terminar con la partida.

ButtonBarIcons: Contiene a los objetos de la interfaz en la zona inferior de la pantalla, excluyendo al menú de opciones.

ButtonBarIcons: Contiene a los objetos de la interfaz en la zona inferior de la pantalla, excluyendo al menú de opciones.

OptionsButton: El botón que el usuario puede apretar para desplegar o replegar el menú de opciones.

DeckRemaining: Sobre este ícono se muestra la cantidad de fichas restantes en la reserva. Se puede apretar para activar o desactivar la opción de cambiar las fichas en la mano.

ActionButtons: Contiene a los objetos que representan los botones con los que el jugador puede realizar acciones.

SendMoveButton y **ButtonSendMoveUIScript.cs**: El botón que el usuario puede apretar para enviar una jugada o pasar de turno. El Script sirve para detectar si hay o no fichas ubicadas en el tablero y cambiar el mensaje del botón de manera acorde. Si hay fichas ubicadas el botón sirve enviar una jugada, y si no las hay sirve para pasar.

ApproveButton: El botón que el jugador puede apretar para aprobar el último acorde jugado.

RejectButton: El botón que el jugador puede apretar para rechazar el último acorde jugado.

ShuffleRackButton: El botón que el jugador puede apretar para cambiar de mano.

GameOverText: Texto que aparece al final de la partida para informarle al jugador que la partida ya terminó.

MoveSender y **MoveSendController.cs**: Este es un objeto que se ubica fuera de la interfaz. Puede recibir peticiones para enviar una jugada, pasar de turno o cambiar la mano, enviar estas peticiones como un mensaje a servidor y recibir la respuesta del servidor.

ToneClusterVoter y **TurnPlayVoteController.cs**: Este es un objeto que se ubica fuera de la interfaz. Puede recibir peticiones para votar a favor o en contra del último acorde jugado, enviar esta petición como un mensaje al servidor y recibir la respuesta del servidor.

ToneClusterLoader y **MatchInfoLoadController.cs**: Este es un objeto que se ubica fuera de la interfaz. Puede recibir peticiones para actualizar los objetos de la interfaz y llamar al servidor para obtener la información de la partida.

LoadTimer y **LoadMatchTimer.cs**: Este es un objeto que se ubica fuera de la interfaz. Implementa un temporizador que llama cada seis segundos al objeto ToneClusterLoader para que vuelva a obtener la información de la partida desde el servidor.

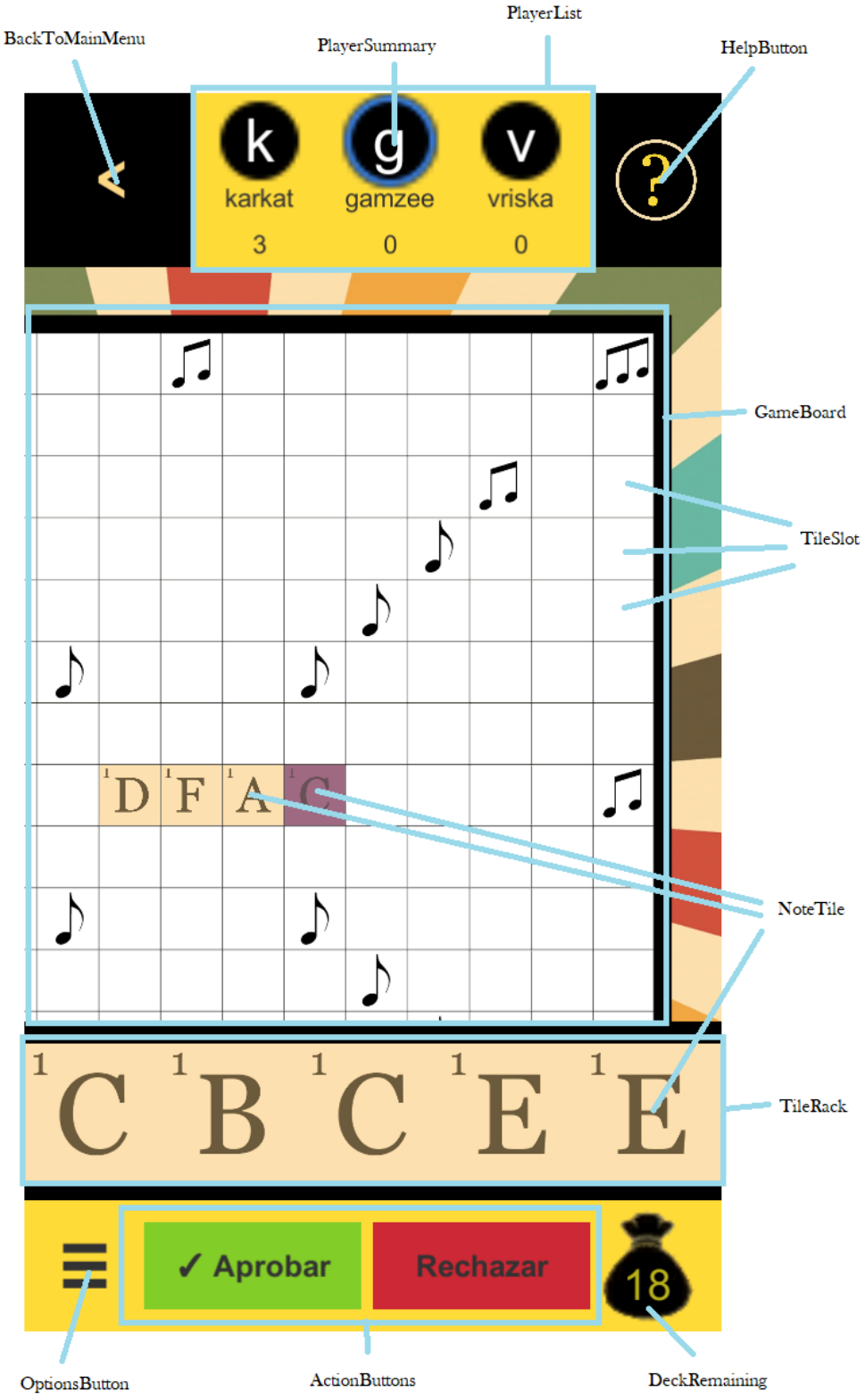


Figura 36: Objetos de Unity en la interfaz de usuario

7.5. Interfaz de Usuario

7.5.1. Navegación entre escenas

La aplicación tiene muchas interfaces. El usuario puede navegar entre ellas interactuando con cada una.

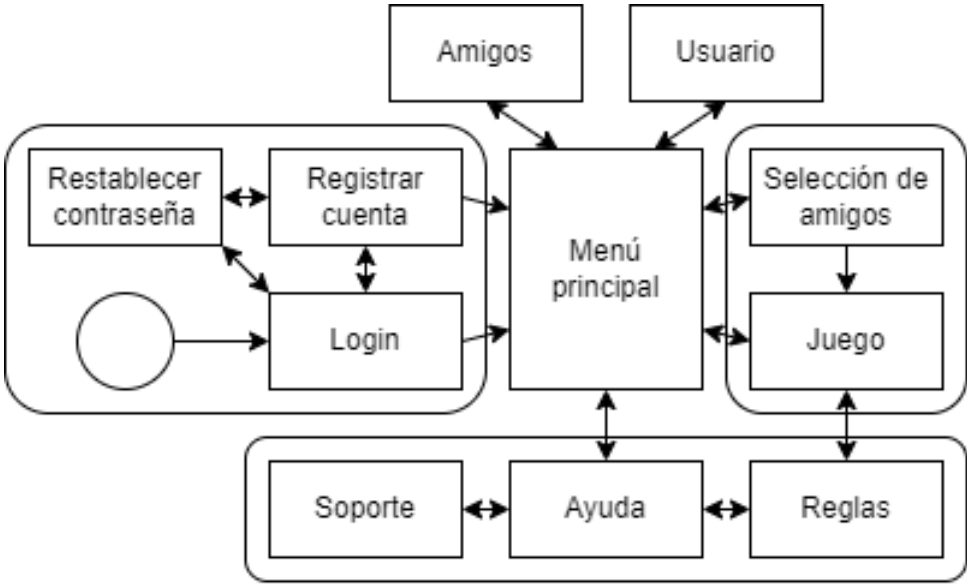


Figura 37: Mapa de Navegación

7.5.2. Escenas del juego

Interfaz de ingreso

En esta interfaz, que se muestra en la figura 38, el usuario puede ingresar a su cuenta. Esta es la primera interfaz que el usuario ve al iniciar la aplicación.

Al apretar el botón "¿No tienes una cuenta? Regístrate", la aplicación cambia a la interfaz de registro. Y al apretar el botón "¿Olvidó su contraseña?", la aplicación cambia a la interfaz de restablecimiento de contraseña. (RS 02)

Para ingresar a su cuenta, el usuario ingresa su dirección de correo electrónico en el campo de texto que dice "Correo electrónico" y su contraseña en el campo de texto que dice "Contraseña". Luego, el usuario presiona el botón "Iniciar sesión" para ingresar a su cuenta e ir al menú principal. (RS 03)

Figura 38: Interfaz de ingreso

Interfaz de registro

En esta interfaz, que se muestra en la figura 39, el usuario puede crear una nueva cuenta.

Al apretar el botón "¿Ya tienes una cuenta? Ingresar", la aplicación cambia a la interfaz de ingreso. Y al apretar el botón "¿Olvidó su contraseña?", la aplicación cambia a la interfaz de restablecimiento de contraseña. (RS 05)

Para registrar una cuenta nueva, el usuario ingresa su dirección de correo electrónico en el campo de texto que dice "Correo electrónico", su nuevo nombre de usuario en el campo de texto que dice "Usuario", y su nueva contraseña en los campos de texto que dicen "Contraseña" y "Confirmar contraseña". Luego el usuario presiona el botón "Crear nueva cuenta" para crear una nueva cuenta e ir al menú principal. (RS 06)

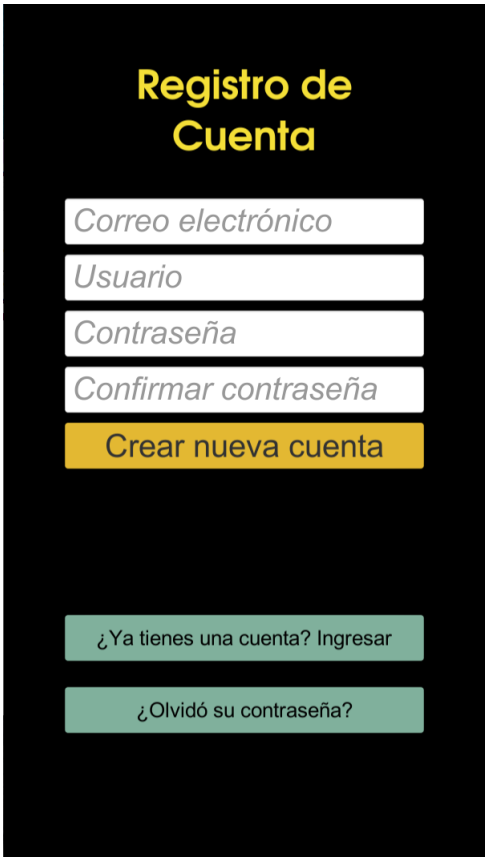


Figura 39: Interfaz de restablecimiento de contraseña

Interfaz de restablecimiento de contraseña

En esta interfaz, que se muestra en la figura 40, el usuario puede iniciar el proceso de restablecimiento de la contraseña de su cuenta.

Al apretar el botón "¿Ya tienes una cuenta? Ingresar", la aplicación cambia a la interfaz de ingreso. Y al apretar el botón "¿No tienes una cuenta? Regístrate", la aplicación cambia a la interfaz de registro. (RS 08)

Para iniciar el proceso de restablecimiento de contraseña, el usuario debe ingresar su dirección de correo electrónico en el campo de texto que dice "Correo electrónico" y apretar el botón que dice "Enviar correo". (RS 09)

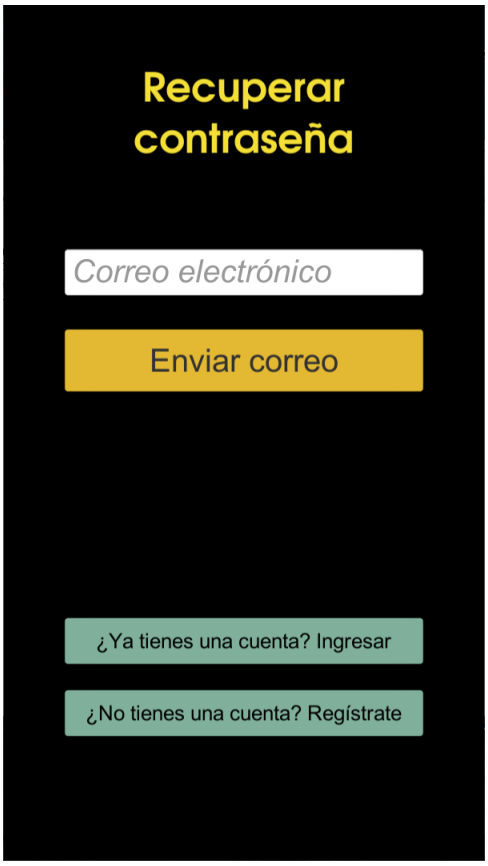


Figura 40: Interfaz de restablecimiento de contraseña

Menú principal

Desde esta interfaz, que se muestra en la figura 41, el usuario puede acceder al resto de la aplicación.

El botón con ícono de persona en la esquina superior izquierda lleva al usuario a su página de usuario. El botón con un signo de interrogación en la esquina superior derecha lleva a la página de ayuda. En la parte superior de la interfaz, se puede ver el puntaje general del jugador. Y el botón que dice "Nuevo juego" lleva al usuario a la interfaz de creación de una partida nueva. (RS 25)

Bajo el título que dice "Partidas Activas", hay una lista horizontal de botones asociados a partidas. Los botones con un ícono de un sobre están asociados a partidas a las cuales el usuario no ha aceptado ni rechazado la invitación todavía (RS 36). Los botones que tienen una letra, en cambio, están asociados a partidas a las que el usuario ya aceptó la invitación. Apretar uno lleva a la interfaz de juego, cargando la partida asociada. (RS 28)

Debajo del resto de la interfaz, se encuentra la barra de navegación. Aquí el usuario puede cambiar a la interfaz de la lista de amigos apretando el botón con el ícono de tres personas. (RS 38)

Figura 41: Menú Principal

Al apretar sobre un botón de invitación, se despliega el contenedor de UI para aceptar o rechazar la partida como se muestra en la figura 42. (RS 30) Este contiene el nombre de usuario del creador de la partida, y dos botones. Apretar el botón que dice “Si” acepta la invitación a la partida (RS 31). Mientras que el botón que dice “No” rechaza la invitación a la partida (RS 33).

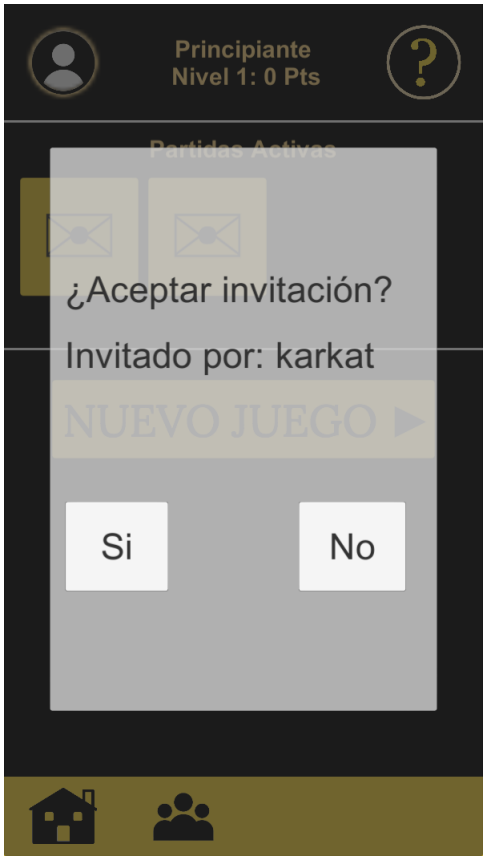


Figura 42: Invitación a una partida

Lista de amigos

En esta interfaz, que se muestra en la figura 43, el usuario puede revisar sus amigos. Además de agregar amigos nuevos y quitar amigos existentes. (RS 11)

En la esquina superior izquierda hay un botón con un ícono de una flecha para volver al menú principal.

En medio de la interfaz hay una tabla con los nombres de usuario de los amigos del usuario. Esta lista es deslizable verticalmente. Para poder ver todos los amigos aún si la cantidad de amigos se vuelve demasiado grande para verlos todos de una sola vez. (RS 13)

Para agregar un amigo, el usuario ingresa el nombre de usuario del amigo que quiera agregar al campo de texto que dice "Agregar" y aprieta el botón con el signo mas (RS 15). Para quitar un amigo, el usuario aprieta el botón con una equis que esté a la derecha del nombre de usuario del amigo que quiera quitar. (RS 17)

En la parte inferior de la interfaz se encuentra la barra de navegación. El botón con la casa lleva al usuario de vuelta al menú principal. (RS 37)

Figura 43: Lista de Amigos

Creación de partida

En esta interfaz, que se muestra en la figura 44, el usuario puede elegir amigos con los cuales jugar una nueva partida y crearla. (RS 21)

En la esquina superior izquierda hay un botón con un ícono de una flecha para volver al menú principal.

En medio de la interfaz se encuentra una lista con los amigos del usuario. Por cada amigo hay un botón gris con su nombre de usuario. Al apretar sobre un botón este se selecciona y cambia de color a verde. Se puede volver a apretar un botón para deseleccionarlo.

Al apretar el botón que dice "Continuar", se crea la partida con los amigos seleccionados. Si se han elegido más de 3 amigos, aparece un mensaje de error recordando que solo se pueden invitar hasta 3 amigos. (RS 22)

(a) Selección de amigos

(b) Error al elegir demasiados amigos

Figura 44: Crear una partida

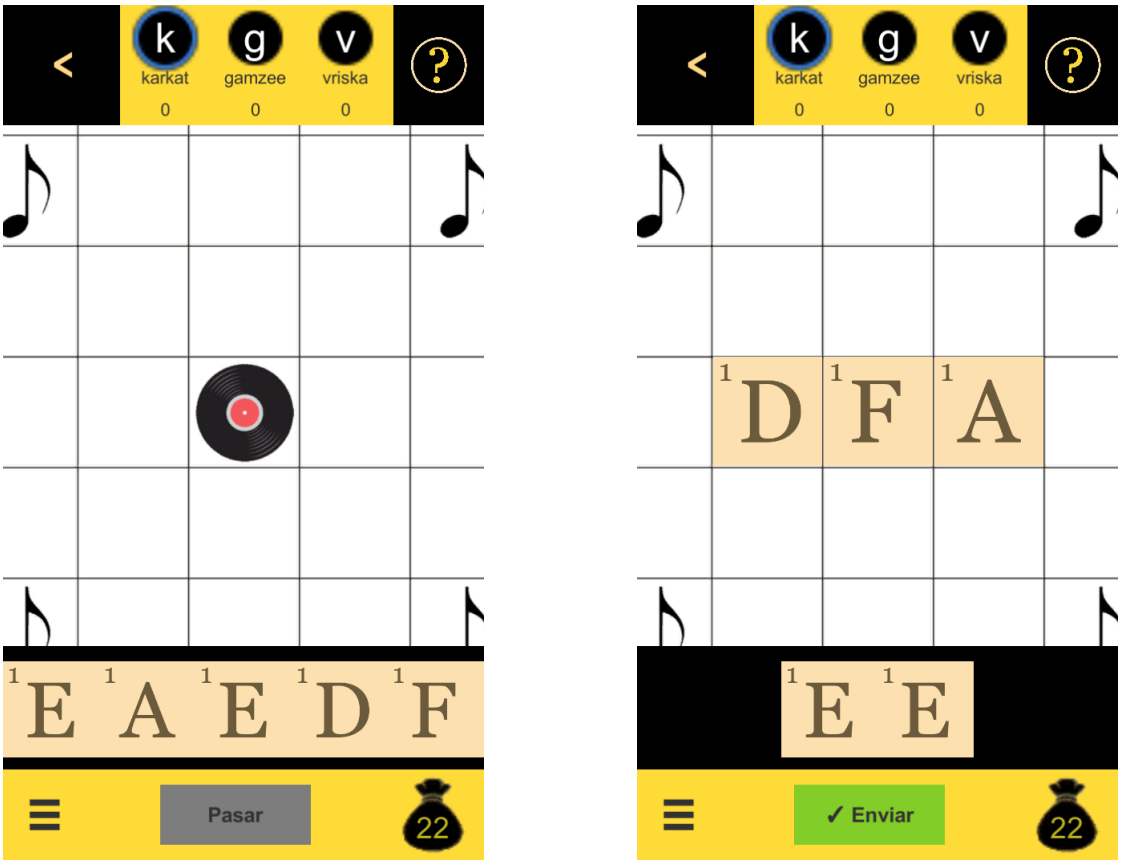
Interfaz de la partida

En esta interfaz, que se muestra en las figuras 45, 46 y 47, se juegan las partidas de Tone Cluster propiamente tal.

En la esquina superior izquierda hay un botón con un ícono de una flecha para volver al menú principal. En la esquina superior derecha hay un botón que lleva a la interfaz con las reglas del juego (RS 38).

En la parte superior de la interfaz hay una lista horizontal con los jugadores de la partida. Por cada jugador aparece su puntaje y su nombre de usuario (RS 41). El jugador al cual le toca el turno aparece destacado con un círculo azul al rededor de la inicial de su nombre (RS 60).

En medio de la interfaz se encuentra el tablero del juego (RS 49). El jugador puede arrastrar el tablero para mover el área del tablero mostrada (RS 52), o arrastrar dos dedos, acercándolos o alejándolos, para cambiar el nivel de zoom sobre el tablero (RS 53). El tablero está compuesto por casillas, las cuales pueden estar vacías o contener una ficha (RS 45). Además, las casillas pueden ser casillas de premio (RS 79). En cuyo caso pueden tener una corchea (RS 81), dos corcheas (RS 82) o tres corcheas (RS 83).



(a) Pasar de turno

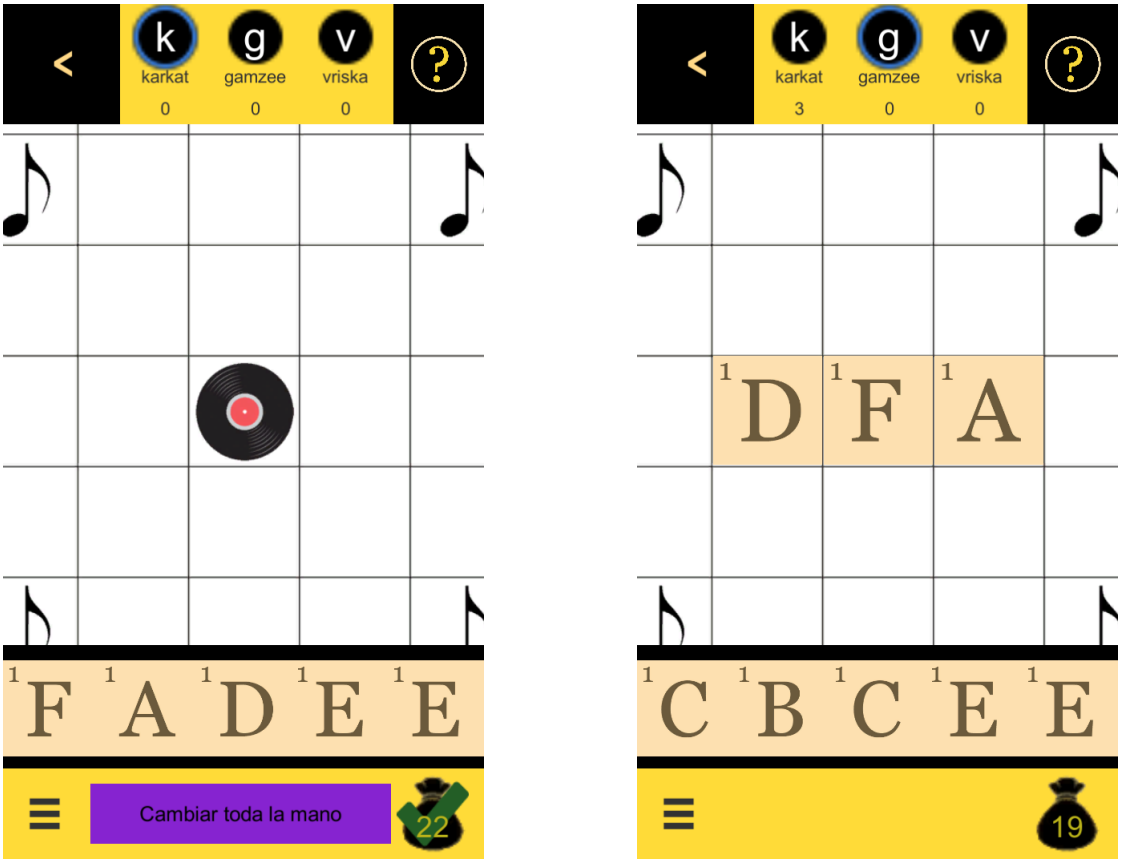
(b) Enviar una jugada

Figura 45: Interfaz de la partida 1/3

Debajo del tablero se encuentra la mano del jugador (RS 54). La cual puede contener fichas (RS 45) que pueden arrastrarse de la mano al tablero y viceversa (RS 57). No se pueden arrastrar las fichas del tablero que hayan sido puestas en otras jugadas.

En la esquina inferior izquierda de la interfaz se encuentra un botón con un ícono de sándwich que despliega un menú, como se muestra en la figura 47b. Desde este menú se puede volver al menú principal (RS 43). Mientras que en la esquina inferior derecha se encuentra un botón con un ícono de bolsa que tiene escrito encima la cantidad de fichas que quedan en el mazo (RS 48).

En el centro de la barra inferior de la interfaz hay un espacio para los botones que el jugador puede apretar para realizar jugadas.



(a) Cambiar las fichas de la mano

(b) Turno de otro jugador

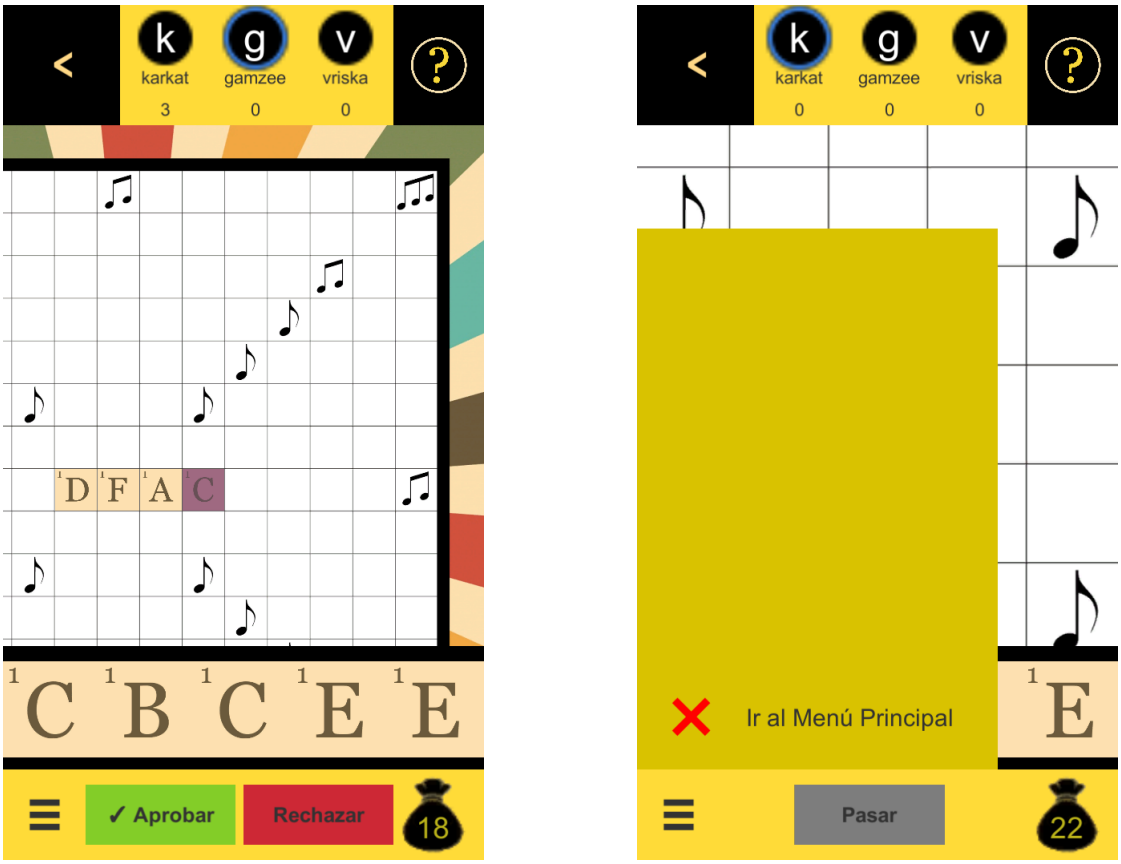
Figura 46: Interfaz de la partida 2/3

Si al jugador le toca su turno y todas sus fichas están en su mano, el espacio inferior lo ocupa un botón que tiene escrito “Pasar”, como se muestra en la figura 45a, que el jugador puede apretar para pasar de turno (RS 67). Si el jugador tiene fichas ubicadas sobre el tablero, el espacio lo ocupa un botón que dice “Enviar”, como se muestra en la figura 45b, que el jugador puede apretar para enviar la jugada con el acorde armado en el tablero (RS 63).

En vez de pasar o enviar una jugada, el jugador puede cambiar las fichas de su mano apretando el botón de la bolsa y luego el botón que dice “Cambiar todas las fichas de la mano” como se muestra en la figura 46 (RS 65). El jugador puede volver a apretar el botón de la bolsa para deshacer haberlo apretado

Si la partida está en la fase de votación del turno de otro jugador, y el jugador no ha votado, entonces el espacio inferior lo ocupan dos botones, como se muestra en la figura 47a. Estos son: Un botón que dice “Aprobar” que el jugador puede apretar para aprobar la jugada hecha (RS 71), y un botón que dice “Rechazar” que el jugador puede apretar para rechazar la jugada hecha (RS 73).

Si no aplica ninguno de los casos anteriores, el espacio inferior de la interfaz pasa a estar vacío, como se ve en la figura 46b.



(a) Votar sobre una jugada

(b) Menú desplegable

Figura 47: Interfaz de la partida 3/3

Menú de ayuda

Desde esta interfaz, que se ve en la figura 48, el jugador puede acceder a interfaces asociadas a darle información al jugador.

En la esquina superior izquierda de esta interfaz hay un botón con un ícono de flecha que el usuario puede apretar para volver al menú principal. En medio de la interfaz hay un botón con ícono de libro que dice “Reglas del juego” que el usuario puede apretar para ir a la interfaz de ayuda, y un botón con ícono de persona que dice “Contacto con soporte” que el usuario puede apretar para ir a la interfaz de contacto con soporte. (RS 90)

Además, al fondo de la interfaz se encuentra la barra de navegación que el usuario puede usar para ir al menú principal o a la interfaz de lista de amigos.

Figura 48: Menú de ayuda

Reglas del juego

En esta interfaz, que se ve en la figura 49, el usuario puede ver las reglas del juego.

En la esquina superior izquierda de esta interfaz hay un botón con un ícono de flecha que el usuario puede apretar para volver a la interfaz de donde llegó (ayuda o juego). Por cada página de las reglas hay un contenedor de UI en el centro de la interfaz con texto e imágenes que explican las reglas del juego. Y en la parte inferior de la interfaz hay dos botones con flechas que le permiten al usuario navegar entre las páginas de reglas del juego. (RS 91)

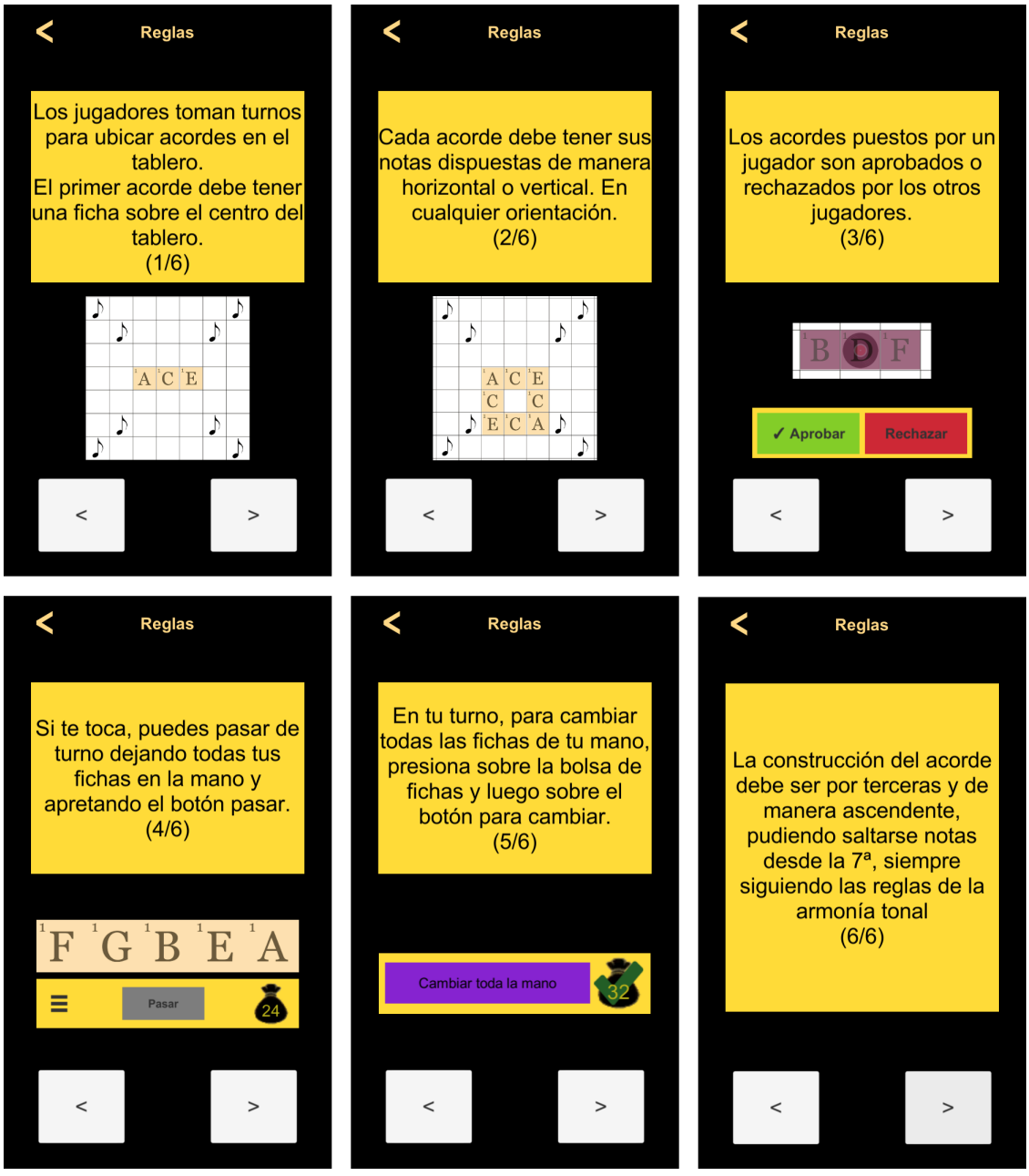


Figura 49: Reglas del juego

Contacto con soporte

En esta interfaz, que se ve en la figura 50, el usuario puede escribir y enviar mensajes al equipo de soporte del juego. (RS 92)

En el cuerpo de la interfaz hay dos campos en los cuales el usuario puede ingresar texto. En el campo que dice “Resumen”, el usuario ingresa el resumen del mensaje que quiere enviar. Mientras que en el campo que dice “Mensaje”, el usuario puede ingresar un mensaje mas largo. Una vez que el usuario haya escrito su mensaje, puede enviarlo presionando el botón que dice “Enviar”. (RS 93)

Al fondo de la interfaz se encuentra la barra de navegación. El usuario puede usarla para ir al menú principal o a la interfaz de lista de amigos.

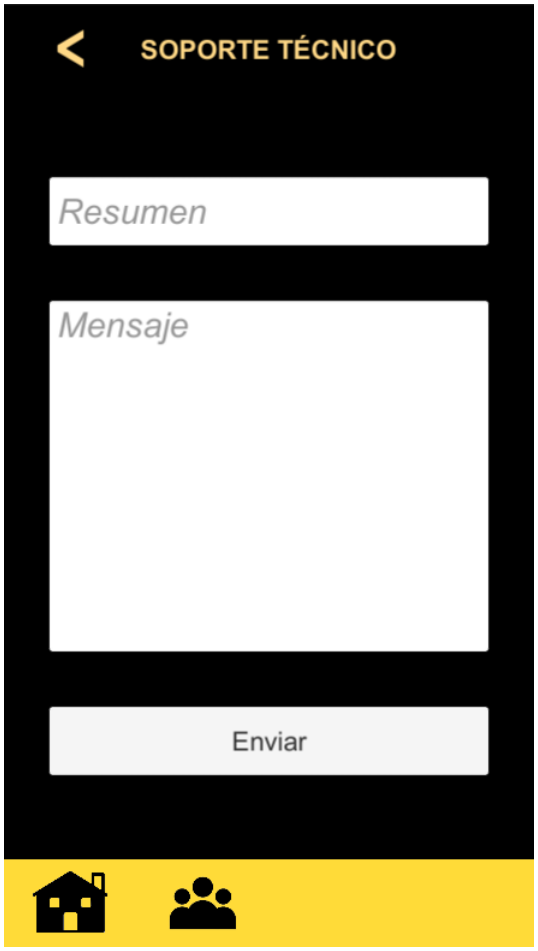


Figura 50: Contacto con soporte

8. Evaluación de Usabilidad

Para evaluar el juego se hicieron dos pruebas con grupos de usuarios. Por un lado, se hizo una prueba de usabilidad con usuarios comunes. Por otro lado, se hizo una evaluación heurística con usuarios expertos.

Para la evaluación, se usaron dos de los cuestionarios de usabilidad construidos por el Profesor Jaime Sánchez [36] [37]. Estos cuestionarios se aplicaron después de que los jugadores probasen el juego para obtener que impresión les causó.

Gracias a que el juego es en linea, la evaluación se pudo realizar a distancia.

8.1. Proceso de Evaluación

En el proceso de evaluación se realizaron los siguientes pasos

1. Selección de los instrumentos.

Para medir la usabilidad de la aplicación con usuarios se usaron cuestionarios provistos por el profesor. Los cuales son ampliamente utilizados en la literatura.

2. Selección de los usuarios evaluadores.

Las circunstancias de pandemia que hubieron durante el proceso de la memoria complicaron el acceso a usuarios para las evaluaciones.

Los usuarios expertos para la evaluación heurística fueron elegidos por el profesor. Los usuarios expertos son profesores universitarios de música, que enseñan armonía musical a alumnos de universidad y que tienen conocimiento y experiencia con aplicaciones digitales en área de música.

Los usuarios para la evaluación de usabilidad fueron elegidos por la diseñadora del juego y el profesor guía de esta memoria. La profesora enseña música a nivel universitario y tiene acceso a estudiantes.

3. Envío de los instrumentos con las instrucciones y link de acceso a la aplicación.

Para los usuarios expertos, se envió un link para descargar el videojuego y un link para responder una encuesta en google forms, la cual incluye las instrucciones de qué se debe realizar en la evaluación.

Para la evaluación de usabilidad con usuarios finales, se le envió la aplicación del videojuego a los estudiantes.

4. Recepción de los instrumentos aplicados y respondidos.

El memorista recibió los resultados de las encuestas.

5. Análisis de las respuestas de los usuarios evaluadores de la usabilidad.

Los resultados de los análisis se ponderaron para ver si los usuarios aceptaron el videojuego.

8.2. Evaluación de Usabilidad con Usuarios

El primer cuestionario es el Cuestionario de Usabilidad de Videojuegos, el cual tiene por objetivo evaluar la usabilidad del videojuego con usuarios finales [36], basado en Nielsen [23]. Este está compuesto por las siguientes preguntas:

1. El videojuego es fácil de navegar/interactuar
2. Es fácil encontrar en las interfaces del videojuego la información deseada
3. Las interfaces son claramente identificadas
4. Las interfaces funcionan correctamente
5. El uso de las imágenes es aceptable
6. El uso del color es aceptable
7. El diseño general del videojuego es apropiado
8. El videojuego es fácil de aprender
9. El videojuego es fácil de recordar
10. El videojuego evita que cometa errores en la interacción
11. El videojuego es interactivo
12. Las interfaces del videojuego son entendibles
13. Las interfaces del videojuego son usables
14. La organización de la información del videojuego es apropiada
15. El contenido del videojuego es relevante
16. La interfaz del videojuego es placentera
17. El videojuego tiene todas las funcionalidades esperadas
18. El videojuego tiene todas las capacidades esperadas
19. ¿Cómo califica globalmente el videojuego analizado?

Hubo un problema en el proceso de la evaluación causado por un bug que no le permitió a los usuario interactuar con la interfaz principal del juego. Este bug se corrigió y los usuarios que no pudieron jugar el juego la primera vez pudieron hacerlo en la segunda. Este problema pudo haber cambiado su opinión sobre el juego en el área de evitar errores.

8.2.1. Resultados

El cuestionario fue aplicado a 10 usuarios. Los cuales fueron 3 aficionados, 5 estudiantes de música y 2 profesionales. Los resultados se describen a continuación en las figuras 51 y 52.

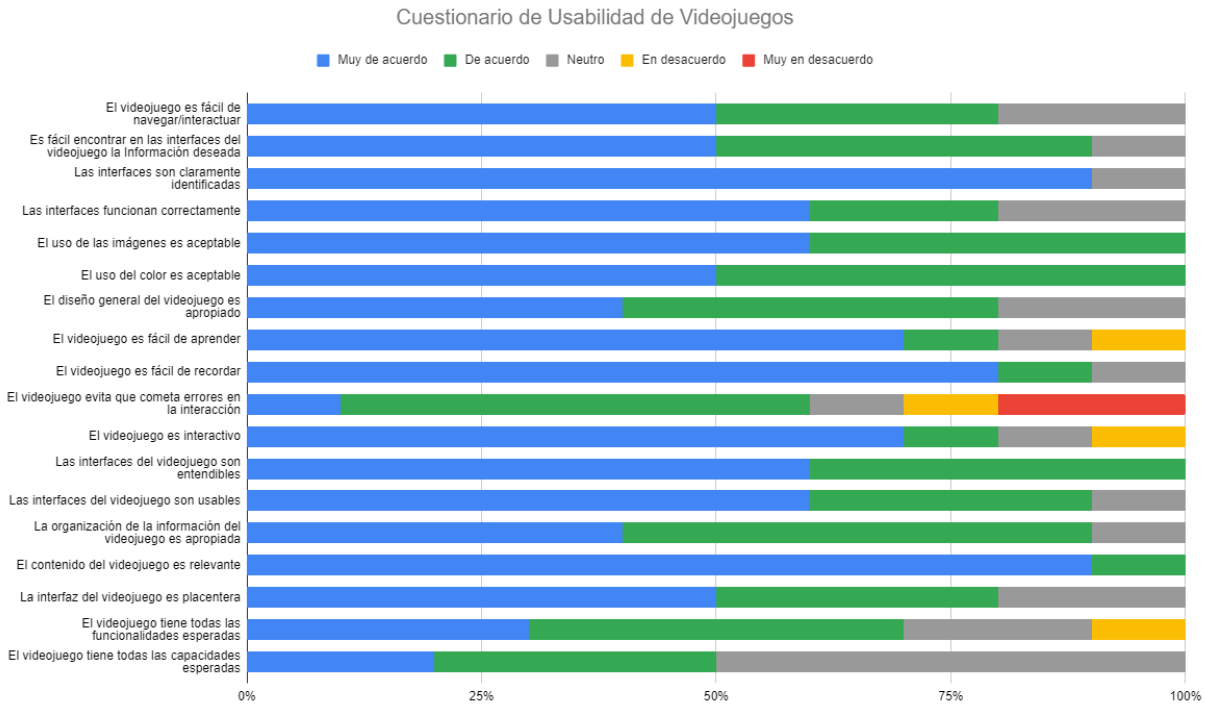


Figura 51: Resultados del Cuestionario de Usabilidad

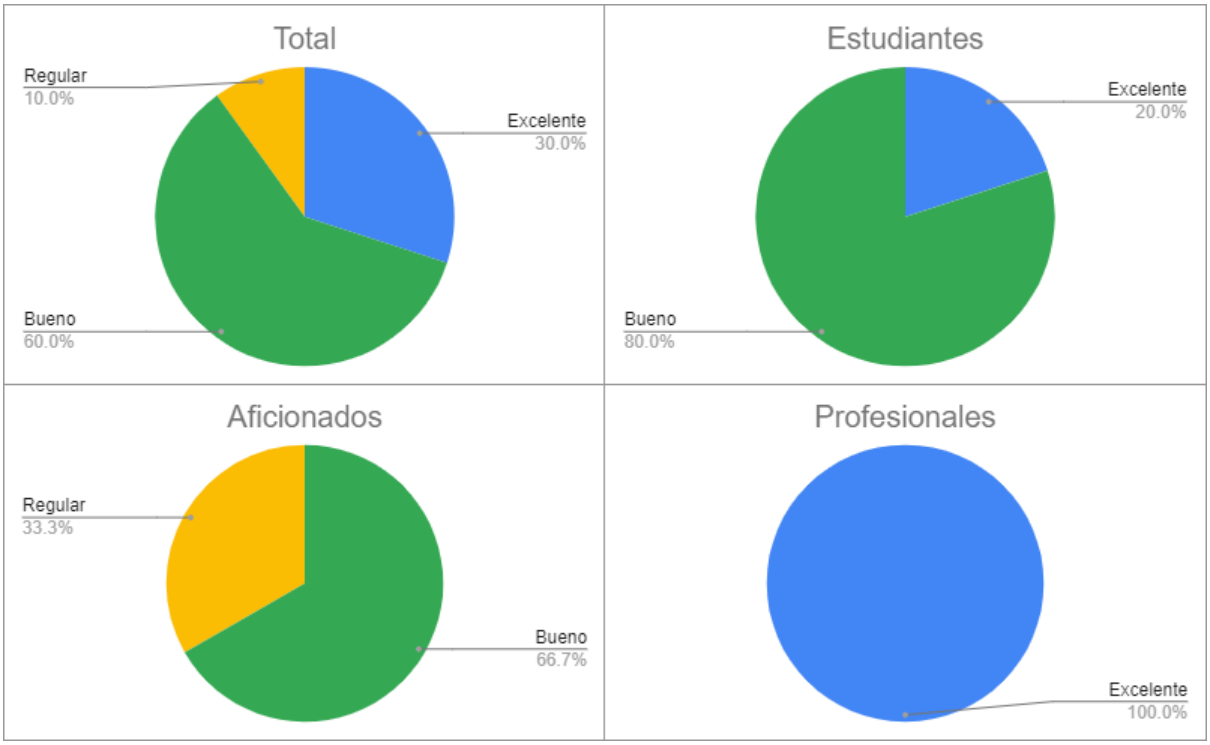


Figura 52: Respuestas a ¿Cómo calificaría globalmente el videojuego analizado?

Lo que los usuarios pusieron en “Justifique su calificación”:

Creo que se pueden añadir algunas cosas que en verdad son detalles que no son estrictamente necesarios, pero que mejorarían la experiencia de juego

A veces el juego no abre las partidas, se queda pegado al tratar de entrar. Creo que podría haber un tutorial con todas las instrucciones y cómo se juega. Aparte de eso el juego funciona muy bien y su interfaz es muy clara. Muy entretenido

Me parece que podrían mejorar ciertos aspectos para hacerlo más fácil de entender, sobre todo a la hora de errar en una jugada. Mensajes instantáneos, colores de alerta, etc, serían un buen elemento a agregar.

No había escuchado ni conocía de algún juego de teoría musical, por lo cual lo encuentro muy original y muy buena idea para estudiantes y profesionales. Creo que puede servir para hacer más digeribles y didácticos ciertos procesos de aprendizaje y también para agilizar y refrescar contenidos teóricos. ¡Está genial!

Creo que la organización de la interfaz es correcta, sin embargo tuve problemas de interacción con los botones al momento de crear una cuenta. No respondían siempre. Lo mismo cuando agregué a un amigo. Me parece que el feedback visual o de sonido de las acciones que se realizan puede mejorar. Estimo que el juego en sí se beneficiaría de mayor feedback cuando el jugador opuesto acepta o rechaza una jugada, como por ejemplo la posibilidad de enviar la justificación del rechazo de una jugada o directamente tener un chat, pues es necesario discutir ciertos movimientos. Por lo demás, la estructura y la navegación es sencilla, así como las mecánicas de juego y su ejecución.

El videojuego es efectivo y usable, sin embargo carece de interactividad pues es imposible explicar al oponente por qué se rechaza una jugada. Por otra parte sería ideal que las reglas tonales utilizadas estuvieran codificadas de tal manera de que el juego indicara si es o no correcta una jugada. Imagino que se considera integrar funciones que permitan resolver estas problemáticas.

Falta feedback para saber en qué uno se equivoca. Si bien creo que es un plus la posibilidad de competir con un amigo, esto puede hacer que el juego se estanque rápidamente. Además, no queda claro porqué es el otro jugador quien acepta o rechaza la jugada. Eso debería ser realizado por la inteligencia interna del juego. Agregaría la posibilidad de jugar solo, y un tutorial.

Tuve unos problemas técnicos a la hora de intentar jugar con alguien más, pero fuera de eso, la idea y puesta en acción está muy buena.

8.3. Evaluación Heurística con Expertos

El cuestionario usado para esta prueba es de Evaluación Heurística [37]. Para este tipo de evaluación solo es necesaria una cantidad pequeña de usuarios expertos [23]. Este cuestionario tiene por objetivo evaluar la usabilidad de videojuegos basado en heurísticas de diseño de interfaces, principalmente basado en los estudios de Nielsen [23] y Schneiderman [45] [46]. Contiene preguntas asociadas a 13 factores y una categoría adicional:

1. Visibilidad del estado del videojuego
2. Relación entre el videojuego y el mundo real
3. Control del jugador y libertad
4. Consistencia y estándares
5. Prevención de errores
6. Reconocer en lugar de recordar
7. Flexibilidad y eficiencia de uso
8. Estética y diseño minimalista
9. Reconocimiento, diagnóstico y recuperación de errores
10. Ayuda y documentación
11. Tratamiento del contenido
12. Velocidad y medios
13. Interactividad
14. ¿Como califica globalmente el videojuego analizado?

8.4. Resultados de la Evaluación Heurística

Para probar la usabilidad de la aplicación, se puso a prueba con tres usuarios expertos. Los resultados de la evaluación están descritos en las figuras 53, 54 y 55.

1. Visibilidad del estado del sistema	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
1.1. El videojuego muestra claramente dónde se encuentra el jugador	1	2				
1.2. Los lugares/secciones posibles de explorar están claramente señalados	2	1				
2. Relación entre sistema y mundo real	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
2.1. El lenguaje es claro	2	1				
2.2. Los conceptos utilizados son entendibles	1	2				
2.3. Las palabras son de significado conocido	1	2				
2.4. Los íconos generan significado	2	1				
3. Control del jugador y libertad	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
3.1. Es fácil regresar a la interfaz inmediatamente anterior	2	1				
3.2. Es fácil volver a la interfaz principal desde cualquier lugar	2	1				
3.3. Provee elementos de interfaces propios para volver o dar paso a otro lugar	2	1				
3.4. El videojuego es soportado por dispositivos de hardware móvil y PC	1		1			1
4. Consistencia y estándares	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
4.1. Existe coherencia entre el nombre de una interfaz y el lugar al que apunta	3					
4.2. Todos los links de las interfaces tienen contenido	2	1				
4.3. Existe coherencia entre el nombre de una interfaz y su contenido	3					
4.4. Sólo existe una forma que lo lleve a una misma interfaz	2	1				
5. Prevención de errores	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
5.1. Existen mensajes que prevengan posibles errores			3			
5.2. Es posible prever posibles errores		1	1	1		
5.3. El videojuego no induce a cometer errores		1	2			
6. Reconocer en lugar de recordar	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
6.1. Los íconos son facilmente reconocibles	1	2				
6.2. Los links pueden identificarse claramente	2	1				
6.3. Es posible reconocer dónde se encuentra el jugador	2	1				

Figura 53: Evaluación heurística. Preguntas 1 a 6.

7. Flexibilidad y eficiencia de uso	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
7.1. Las interfaces son de fácil acceso	2	1				
7.2. Las interfaces permiten una adaptación del jugador	2				1	
7.3. Las interfaces favorecen la continuidad en el juego	1	2				
8. Estética y diseño minimalista	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
8.1. La información del videojuego es relevante	3					
8.2. El contenido está bien clasificado	1	2				
8.3. El contenido está correctamente organizado	1	2				
8.4. El contenido está bien distribuido en el videojuego	1	2				
9. Reconocimiento, diagnóstico y recuperación de errores	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
9.1. Es fácil reconocer cuándo ocurre un error		1	1	1		
9.2. Después que ocurre un error es fácil volver a la interfaz de origen		1	2			
9.3. Cuando ocurre un error existen mecanismos para solucionarlos			2	1		
10. Ayuda y documentación	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
10.1. Existe algún tipo de ayuda o indicación en el videouego	2	1				
10.2. Cuando existe ayuda, ésta es específica	2	1				
10.3. La ayuda está asequible	2	1				
11. Tratamiento del contenido	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
11.1. El contenido se adecua a la realidad social y cultural del jugador	1		1	1		
11.2. El contenido del videojuego constituye un valor agregado en relación al mismo contenido en otro medio	2	1				
11.3. Existe opción de realizar consultas al o los autores o foros relacionados con el videojuego	1	1				1
11.4. Es posible ampliar la información accediendo a otras interfaces relacionadas con el tema	1		2			

Figura 54: Evaluación heurística. Preguntas 7 a 11.

12. Velocidad y medios	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
12.1. Existe posibilidad de acceder al contenido del videojuego en formato texto			2		1	
12.2. Los medios utilizados (imágenes, video, sonido) demoran en exceso el desarrollo del videojuego					3	
12.3. La calidad técnica de videos, imágenes y sonido es aceptable	1	1			1	
12.4. Los medios utilizados refuerzan el aprendizaje del funcionamiento del videojuego	1	2				
12.5. Los elementos multimedia son apropiados para el contenido expuesto	1	2				
13. Interactividad	Muy de acuerdo	De acuerdo	Neutro	En desacuerdo	Muy en desacuerdo	No aplica
13.1. El videojuego es interactivo	2	1				
13.2. Se involucra al usuario en tareas y problemas para el aprendizaje del videojuego	2	1				
13.3. Las interfaces generan visibilidad del jugador	1	2				
13.4. Las interfaces generan mapping del jugador	2	1				
13.5. Las interfaces generan affordances del jugador	1	2				
13.6. Las interfaces generan feedback para el jugador	1	2				
13.7. Las interfaces se ajustan al modelo mental del jugador	2	1				
¿Cómo califica globalmente el videojuego analizado?	Excelente	Bueno	Neutro	Regular	Deficiente	
14.1. Visibilidad del estado del videojuego	1	2				
14.2. Relación entre sistema y mundo real	1	2				
14.3. Control del jugador y libertad	2	1				
14.4. Consistencia y estándares	1	1	1			
14.5. Prevención de errores			3			
14.6. Reconocer en lugar de recordar	1	1	1			
14.7. Flexibilidad y eficiencia de uso	1	2				
14.8. Estética y diseño minimalista	2	1				
14.9. Reconocimiento, diagnóstico y recuperación de errores		1	2			
14.10. Ayuda y documentación	1	2				
14.11. Tratamiento del contenido	1	2				
14.12. Velocidad y medios	1	2				
14.13. Interactividad	2	1				

Figura 55: Evaluación heurística. Preguntas 12 a 14.

Unico comentario escrito en “Comentarios adicionales”:

> Excelente prototipo. Algunos comentarios:
> - pienso que es el sistema debiese dar más feedback en general. Por ejemplo, cuando un jugador rechace correctamente una jugada
> - pienso que el sistema debe no permitir, y sugerir al intentar hacer una jugada no válida
> - ¿El juego termina cuando se acaban las piezas? ¿cómo se calcula el puntaje?
> - se podría sumar el sonido de acorde construido, y no solo de la nota sola
> - podría haber una partida contra “el cpu” estilo tutorial
> - que se pudiese hacer zoom (“in” y “out”) del tablero fue genial
> Exito!

8.5. Conclusiones de la evaluación

En la evaluación de usabilidad con usuarios finales, de un total de 14 preguntas, en 11 de ellas se obtuvo por lo menos un 75 % de aprobación. Con esto, se considera que el resultado de la evaluación de usabilidad fue positivo.

De las tres preguntas que recibieron menos de un 75 % de aprobación, dos de ellas fueron “el videojuego de todas las funcionalidades esperadas” y el “videojuego tiene todas las capacidades esperadas”. Estas dos respuestas tienen en común que los usuarios esperaban más funcionalidades en la aplicación.

La otra pregunta que recibió menos de un 75 % de aprobación fue “el videojuego evita que cometa errores en la interacción”. Hubo un problema en la evaluación causado por un bug que tuvo su origen en la diferencia técnica entre el celular usado para el testeo interno durante el desarrollo y los celulares de los usuarios, pues el celular usado para el testeo interno tenía más capacidad técnica.

Este bug causó que no se pudiera interactuar con la interfaz para jugar partidas, lo que llevó a que se tuviera que repetir la evaluación con los usuarios una vez que el bug estuviese corregido. Este percance pudo haber afectado el resultado de la evaluación.

Por otra parte, la aplicación tiene una gran cantidad de servicios que dependen del internet, lo cual también pudo haber contribuido a que hubiesen errores.

Otras observaciones comunes son que debería haber mas retroalimentación para el usuario, que se puede mejorar el aspecto visual y que debería haber un tutorial o un modo de un solo jugador.

Cabe destacar que algunos elementos que los usuarios querían ver en la aplicación, como un chat de mensajes, estaban complementados en el diseño original anterior a esta memoria de la aplicación. Pero no se pudieron incluir por falta de tiempo.

9. Discusión Final

Durante este trabajo de título se rediseñó, implementó y evaluó la aplicación de videojuego Tone Cluster para celular. La cual es capaz de manejar partidas en linea.

El rediseño fue simple, pero pertinente a los objetivos del juego. En el futuro se podría realizar un rediseño mas grande, como agregar un modo con límite de tiempo.

El desarrollo de esta aplicación fue desafiante por su gran cantidad de funcionalidades dependientes de la comunicación en línea. Lo cual hizo que la aplicación tuviera una gran escala y complejidad. Por lo cual se tuvo que tomar mas tiempo para implementar la aplicación y reducir su escala para incluir solo lo más importante para su componente central, que es el juego en línea entre usuarios.

El uso en el lado del cliente de Unity tuvo la ventaja de acelerar el desarrollo en áreas como la creación de la interfaz de usuario, la interactividad sobre las fichas y la conexión con el servidor, pero también hubo que considerar el peso en recursos para los dispositivos de los usuarios.

El uso en el lado del servidor de PlayFab también aceleró el desarrollo al no tener que crear todas las interacciones de creación de cuenta, validación y organización de información comunes en los videojuegos como la lista de amigos y las invitaciones a partidas, pero había poca documentación clara, así que se tuvo que experimentar bastante para alcanzar el desarrollo obtenido. En particular, la forma en que se guardó la información de las partidas aplicando los grupos de PlayFab tuvo que descubrirse.

El resultado de la aplicación se considera positivo por el resultado de las evaluaciones y el interés que generó entre los usuarios, siendo el videojuego aceptado entre los usuarios.

Se aprendió que durante las pruebas internas se deben utilizar dispositivos de gama baja y alta para evitar que los errores de rendimiento se descubran en la evaluación con usuarios.

Falta refinar la implementación para que los usuarios no caigan en errores evitables. También, se requiere incluir mas funcionalidad para que la experiencia de juego cumpla plenamente con el diseño original. Esta funcionalidad incluye los niveles de dificultad, el chat de texto y la verificación dentro de las partidas de las jugadas realizadas. Además, se pueden incluir ideas propuestas por los usuarios, como un tutorial o un modo de un solo jugador.

En el futuro será posible robustecer la aplicación para que evitar errores y también se podrá incluir las herramientas de las que se prescindió, como el piano y las funciones harmónicas. Así también, se puede mejorar el feedback de las interfaces para darle una mejor experiencia al usuario. Finalmente, se estima conveniente realizar una evaluación de usabilidad más representativa y completa, con una muestra más grande y con tareas más específicas y detalladas.

BIBLIOGRAFÍA

[1] Medel Sierralta, D. (2021) Propuesta de Prototipo de Videojuego Online para el Aprendizajede la Armonía Musical. Tesis de Magister, Universidad de Chile Facultad de Ciencias Sociales Escuela de Postgrado, Santiago, Chile

[2] Nettles, B. (1987) Harmony 1. New York: Berklee College of Music

[3] Savage, J. (2007) Reconstructing music education through ICT. Research in Education. 78. 65-77

[4] Denis, G. & Jouvelot, P. (2005). Motivation-driven educational game design: Applying best practices to music education. 462-465. 10.1145/1178477.1178581

[5] Anderson, T. & Dron, J. (2011). Three Generations of Distance Education Pedagogy. International Review of Research in Open and Distance Learning 12.10.19173/irrodl.v12i3.890.

[6] Bogost, I. (2004). 1 Asynchronous Multiplay Futures for Casual Multiplayer Experience

[7] Fu, F.-L., Su, R.-C. & Yu, S.-C. (2009). EGameFlow: A scale to measure learners' enjoyment of e-learning games. Computers & Education. 52. 101-112. 10.1016/j.compedu.2008.07.004

[8] Yáñez-Gómez, R., Font, J.L., Cascado-Caballero, D. et al. (2019) Heuristic usability evaluation on games: a modular approach. Multimed Tools Appl 78, 4937–4964. https://doi.org/10.1007/s11042-018-6593-1

[9] Sánchez, J., Mendoza, C. & Salinas, A. (2009). Mobile serious games for collaborative problem solving. Studies in health technology and informatics. 144. 193-7. 10.3233/978-1-60750-017-9-193

[10] Portowitz, A., Peppler, K. A., & Downton, M. (2014). In Harmony: A technology-based music education model to enhance musical understanding and general learning skills. International Journal of Music Education, 32(2), 242–260. https://doi.org/10.1177/0255761413517056

[11] Portowitz, A., Lichtenstein, O. & Egorov, L. (2015). "Teach, Learn, Evaluate"(TLE): A Computer-based Music-learning Tool Designed to Foster Musical Understanding, General Learning Skills, and Crosscultural Understanding (Poster)

[12] Bamberger, J. S. & Hernandez, A. (2002). Tuneblocks. https://tuneblocks.com/

[13] Holland, S. (1994). Learning about harmony with harmony space: an overview. Tech. Rep. STAN-M-88, Stanford University Department of Music

[14] Mandanici, M. & Roda, A. (2014). The "Harmonic Walk": an Interactive Educational Environment to Discover Musical Chords

[15] Ruthmann, S. A. (2013). Musedlab. [en línea] <https://musedlab.org/>[consulta: 06 de octubre 2022]

[16] NYU Music Experience Design Lab (MusEDLab). (s.f.). MathScienceMusic. [en línea] <https://mathsciencemusic.org/>[consulta: 06 de octubre 2022]

[17] Google. (2018). CHROME MUSIC LAB. [en línea] <https://musiclab.chromeexperiments.com/>[consulta: 06 de octubre 2022]

[18] Enriquez, J. G. & Casas, S. I. (2013) Usabilidad en aplicaciones móviles. Informes Científicos - Técnicos UNPA, [S.l.], v. 5, n. 2, p. 25-47, june 2014. ISSN 1852-4516. Disponible en: http://ict.unpa.edu.ar/journal/index.php/ICTUNPA/article/view/ICTUNPA-62-2013

[19] Al Fatta, H., Maksom, Z. & Zakaria, M.H. (2018) Systematic literature review on usability evaluation model of educational games : playability, pedagogy, and mobility aspects 1. J. Theor. Appl. Inf. Technol. 31(14)

[20] Hernández Gallardo, S. C. (2007). El constructivismo social como apoyo en el aprendizaje en línea. Apertura, 7(7),46-62. ISSN: 1665-6180. Disponible en: https://www.redalyc.org/articulo.oa?id=68800705

[21] Jain, A. (2018). A Beginner's Guide to Rapid Prototyping. [en línea] <https://www.freecodecamp.org/news/a-beginners-guide-to-rapid-prototyping-71e8722c17df/>[consulta: 06 de octubre 2022]

[22] Microsoft. (2020). PlayFab Client REST API. [en línea] <https://docs.microsoft.com/en-us/rest/api/playfab/client/>[consulta: 06 de octubre 2022]

[23] Nielsen, J. (1993). Usability Engineering, Morgan Kaufmann, ISBN 9780125184069

[24] Sommerville, I. (2015). Software Engineering 10th edition, página 134

[25] Aihara, D. S. (2009). Study About the Relationship Between the Model-View-Controller Pattern and Usability. <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A833593&dswid=-1790>

[26] Sommerville, I. (2019). Software Engineering 10th edition Web sections, Requirements traceability. <iansommerville.com/software-engineering-book/web/traceability/>

[27] Unity Technologies. (s.f.). Unity. [en línea] <https://unity.com/>[consulta: 06 de octubre 2022]

[28] Epic Games, Inc. (s.f.). [en línea] <https://www.unrealengine.com/en-US>[consulta: 06 de octubre 2022]

[29] Tristem, B. (2022). Unity vs. Unreal: Which Game Engine is Best For You?. [en línea] <https://blog.udemy.com/unity-vs-unreal-which-game-engine-is-best-

for-you/>[consulta: 06 de octubre 2022]

[30] Batschinski, G. (s.f.). What is BaaS – Backend-as-a-Service?. [en línea] <https://blog.back4app.com/backend-as-a-service-baas/>[consulta: 07 de octubre 2022]

[31] Ferencz, E. (2021). AWS, AZURE or GCP? Huge comparison of cloud providers for the gaming industry. [en línea] <https://www.revolgy.com/insights/blog/aws-azure-or-gcp-huge-comparison-of-cloud-providers-for-the-gaming-industry>[consulta: 07 de octubre 2022]

[32] Clark, J. (s.f.). Backend Platform for Games. [en línea]. <https://blog.back4app.com/backend-platform-games/>[consulta: 07 de octubre 2022]

[33] Batschinski, G. (s.f.). Top Mobile Game Backend Providers. [en línea]. <https://blog.back4app.com/mobile-game-backend/>[consulta: 07 de octubre 2022]

[34] Microsoft. (2022). Quickstart: PlayFab Client library for C# in Unity. [en línea]. <https://docs.microsoft.com/en-us/gaming/playfab/sdks/unity3d/quickstart>[consulta: 07 de octubre 2022]

[35] Microsoft. (2022). What is PlayFab?. [en línea] <https://docs.microsoft.com/en-us/gaming/playfab/what-is-playfab>[consulta: 07 de octubre 2022]

[36] Sánchez, J. (2016). Evaluación de Usabilidad de Videojuegos, Cuestionario de Usuario Final. Departamento de Ciencias de la Computación. Universidad de Chile

[37] Sánchez, J. (2020). Evaluación de Usabilidad de Videojuegos, Cuestionario de Evaluación Heurística. Departamento de Ciencias de la Computación. Universidad de Chile

[38] Microsoft. (2020). Entity groups. [en línea] <https://learn.microsoft.com/en-us/gaming/playfab/features/social/groups/quickstart>[consulta: 07/10/2022]

[39] Microsoft. (2020). Using Shared Group Data. [en línea] <https://learn.microsoft.com/en-us/gaming/playfab/features/social/groups/using-shared-group-data>[consulta: 07/10/2022]

[40] Microsoft. (s.f.). Shared Group Data. [en línea] <https://learn.microsoft.com/en-us/rest/api/playfab/server/shared-group-data?view=playfab-rest>[consulta: 07/10/2022]

[41] Microsoft. (2022). CloudScript. [en línea] <https://learn.microsoft.com/en-us/gaming/playfab/features/automation/cloudscript/>[consulta: 07/10/2022]

[42] Microsoft. (2022). Actions and rules quickstart. [en línea] <https://learn.microsoft.com/en-us/gaming/playfab/features/automation/actions-rules/quickstart>[consulta: 07/10/2022]

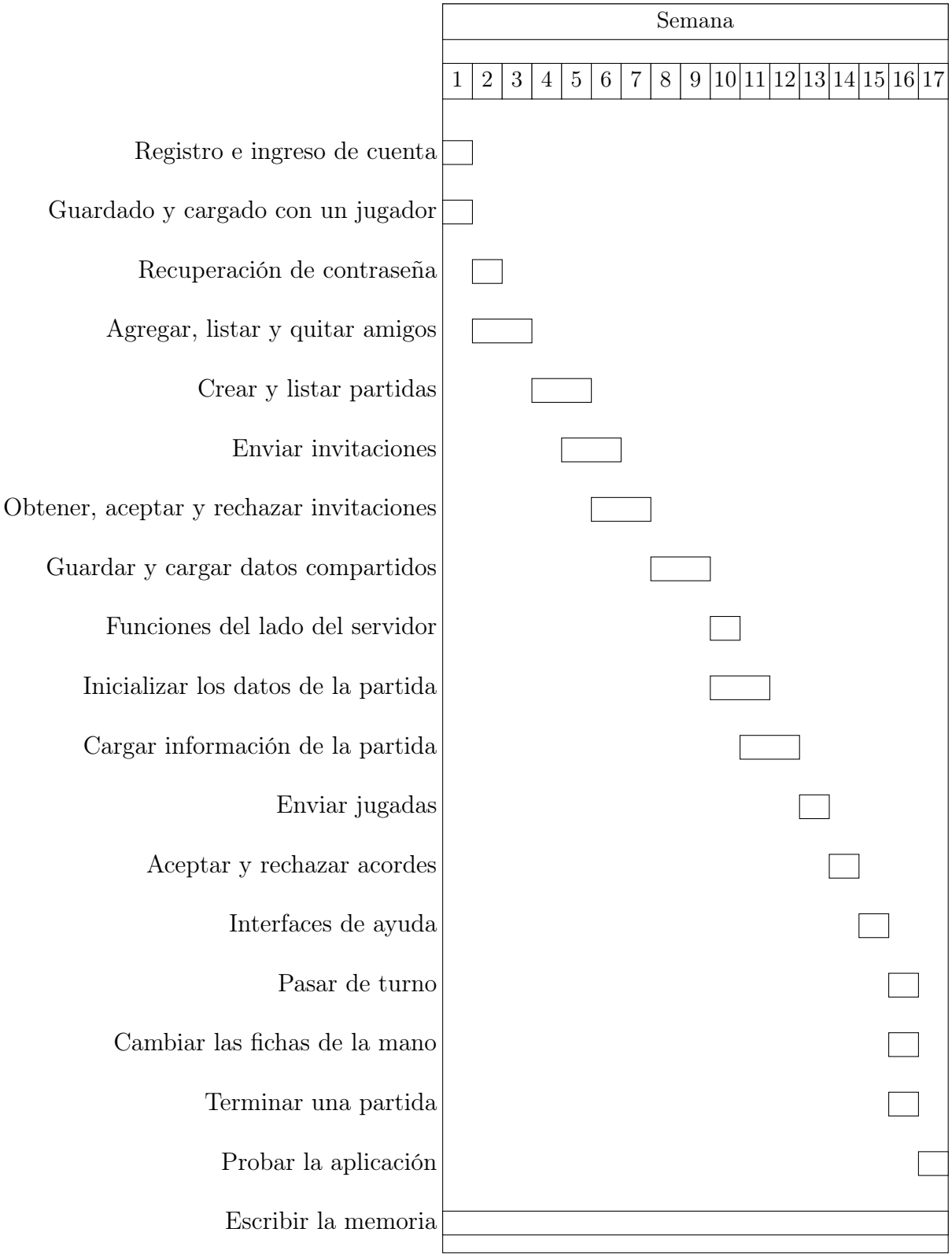
[43] Microsoft. (2020). Encrypted logins. [en línea] <https://learn.microsoft.com/en-us/gaming/playfab/gamemanager/encrypted-logins>[consulta: 11/10/2022]

[44] Microsoft. (s.f.). Authentication - Login With Email Address. [en línea] <https://learn.microsoft.com/en-us/rest/api/playfab/client/authentication/login-with-email-address?view=playfab-rest>[consulta: 11/10/2022]

[45] Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., & Elmqvist, N. (2016). Designing the User Interface: Strategies for Effective Human-Computer Interaction: Sixth Edition, Pearson

[46] Wong, E. (2020). Shneiderman's Eight Golden Rules Will Help You Design Better Interfaces. <https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces>[consulta: 13/10/2022]

Anexo


Semana
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Registro e ingreso de cuenta
Guardado y cargado con un jugador
Recuperación de contraseña
Agregar, listar y quitar amigos
Crear y listar partidas
Enviar invitaciones
Obtener, aceptar y rechazar invitaciones
Guardar y cargar datos compartidos
Funciones del lado del servidor
Inicializar los datos de la partida
Cargar información de la partida
Enviar jugadas
Aceptar y rechazar acordes
Interfaces de ayuda
Pasar de turno
Cambiar las fichas de la mano
Terminar una partida
Probar la aplicación
Escribir la memoria