PROGRAMACIÓN DE SISTEMAS 24/25 Q1
Icono de la aplicación

Juego de air hockey con pantalla dividida

Autores:
José Manuel Fernández Montáns (j.m.fmontans@udc.es)
Mateo Rivela Santos (mateo.rivela@udc.es)
Hugo Mato Cancela (hugo.matoc@udc.es)
Persona de contacto: Hugo
Fecha: *A Coruña, 12 Noviembre 2024*
Versión:*1.1*
Nombre de la aplicación: *Space Rend Hockey*
Github: https://github.com/Hugoomv/TT_PS_Grupo-Q1.1_24-25

Índice

Capítulos **Página**

Cuadro 1: Tabla de versiones.

Versión	Fecha	Autor
0.1	10/10/24	Hugo
1.0	21/10/24	Hugo
1.1	10/11/24	Hugo
1.2	13/11/24	Hugo

1. Introducción

1.1. Objetivos

El objetivo principal será trabajar con una pantalla dividida con una baja latencia y gestionar los rebotes para una pelota que viaja entre dispositivos y añadir un menú.

Podemos añadir notificaciones para que se una alguien a la partida, añadir un ranking, cambiar el tipo de pelota, añadir distintas dificultades iniciales, la pelota que aumente de velocidad con cada rebote de forma indefinida, ver las personas conectadas al juego, poner un estado a cada persona en base a si está en línea o no, o si ya está en una partida.

1.2. Motivación

Intención de experimentar con traspaso de datos entre dispositivos e interacción de elementos de la aplicación con la pantalla del dispositivo.

1.3. Trabajo relacionado

Existen aplicaciones similares, como es el caso de juegos de air hockey online. Nuestra aplicación busca conectar dos dispositivos de forma remota, a diferencia de las otras aplicaciones, que apuestan por una pantalla dividida para dos jugadores. Destacar que la base del juego es muy parecida, pero sin tener que conectar pantallas.

2. Análisis de requisitos

2.1. Funcionalidades

Conectar dos teléfonos, cada uno mostraría una mitad del campo completo, de forma que los elementos en los que cada usuario se tiene que fijar pueden ser de mayor tamaño.

Los mazos con los que cada uno tiene que golpear el disco se controlarían con un solo dedo, apareciendo en la parte de la pantalla donde se mantiene este.

El disco, al ser golpeado por uno de los mazos, se mueve en una trayectoria determinada por el ángulo (y, preferiblemente, fuerza) del impacto, pudiendo ser alterada al chocar con los límites del campo, que corresponderían a los bordes del dispositivo.

2.2. Prioridades

Lo principal sería lograr la conexión entre dispositivos, pasar los datos de forma correcta y apropiada, que el disco pase y rebote para implementar el juego de air hockey. Primero, se implementará el traspaso de objetos entre pantallas y dispositivos con conexión mediante wifi-internet usando Firebase. Luego, trabajaremos con rebotes y bordes de la pantalla como paredes. Una vez realizado lo más básico, añadiremos una pantalla de menú para acceder a los ajustes y a prototipos anteriores. Por último, uniremos las funcionalidades básicas, añadiremos bloques destructibles que funcionarán como paredes.

3. Planificación inicial

3.1. Iteraciones

Aqui se definen los prototipos de nuestro proyecto y las funcionalidades principales que trabajamos en cada prototipo. Se implementarán en el orden establecido:

P1: Pasar un objeto entre dos dispositivos conectados por Wifi-internet usando Firebase.

P2: Implementar los rebotes y los bordes de la pantalla del dispositivo como paredes en un modo para un jugador.

P3: Añadimos una maza que el usuario puede manejar y que rebota con la pelota. Sigue siendo en modo para un jugador.

P4: Añadir un menú con el que acceder a la pantalla de ajustes y a los prototipos anteriores.

P5: Unión de funcionalidades de P1 y P2: un disco que rebota y se transporta entre dos pantallas al atravesar el borde superior. En ajustes también se podrá cambiar a izquierdo-derecho para tener más tiempo de reacción.

P6: Añadir bloques destructibles que funcionen como paredes.

P7: Producto final

3.2. Responsabilidades

Físicas: Mateo

Conexión y Visualización: José Manuel

Visualización y Entregable: Hugo

3.3. Hitos

- Crear usuarios en la BD.
- Login y logout de usuarios.
- Enviar datos entre dos dispositivos usando Firebase.
- Establecer una conexión de baja latencia entre ambos dispositivos.
- Implementar rebotes.
- Hacer que los bordes de la pantalla funcionen como paredes.
- Añadir bloques destructibles que funcionen como paredes.
- Crear un menú con varios ajustes.
- Crear una partida añadiendo a un jugador mediante invitación.
- Mostrar un ranking con el número de victorias o puntos.
- Enseñar el perfil de otros jugadores

Entregables: Cada prototipo será una entrega.

3.4. Incidencias

No tenemos conocimientos sobre gráficos o físicas por lo que tendremos que aprender y probar todo lo relacionado a eso. Además, para paliar esas carencias, tenemos pensado usar LibGDX [1] y KryoNet [2] para la codificación general de juego. Y usaremos las librerías Box2d [3], Tween Engine [4] para las físicas y las animaciones respectivamente. Para establecer conexión se hará uso de Firebase [5]. En lo relacionado a Android, tomaremos como referencia lo enseñado en clase y la documentación oficial [6]. Por último, en lo relacionado a los gráficos y apartado más visual, Canva [7].

En la primera iteración, probaremos la conexión entre dispositivos y que el objeto se pasa correctamente entre ellos. En la segunda, en un prototipo distinto, el objeto debe rebotar contra los bordes de la pantalla de un dispositivo para que funcionen como paredes En la tercera, implementamos la maza, que debe responder correctamente al input del usuario, y nos aseguramos que la pelota rebote correctamente contra esta. En la cuarta, el menú nos debe permitir cambiar entre varios prototipos En la quinta, uniremos las funcionalidades de las dos primeras iteraciones y nos aseguraremos de que funcionan correctamente ambas al mismo tiempo En la sexta, añadiremos bloques que funcionen como paredes. La pelota debe rebotar contra estos como lo hace con los bordes de la pantalla. En la última, comprobaremos que, tras haber implementado todo, funciona correctamente.

En caso de que alguien enferme repartiremos el trabajo asignado a esa persona entre los demás integrantes del grupo.

4. Diseño

4.1. Arquitectura

Se hará uso de una arquitectura centralizada cliente servidor. La distribución de trabajo se hará entre el servidor (Firebase) y el cliente (el usuario de nuestra app). El servidor almacena la información de los usuarios, y los mensajes que envían. Guardará datos como si el usuario está conectado o no, su id (único para cada uno), nombre, correo y contraseña. El cliente revisa el usuario actual, con el que estamos logueados, en el servidor y tiene un listener que escucha los cambios, como son los mensajes recibidos por otros usuarios.

4.2. Persistencia

Respecto a qué información almacenaremos, ambos jugadores siempre tendrán los datos de la posición, ángulo y fuerza de la pelota. Además, cada uno guardará dónde está situada su propia maza en caso de que esté siendo usada. El servidor guardará las coordenadas de la pelota, que los usuarios obtendrán de forma constante.
Mientras el usuario está conectado, podrá recibir datos, invitaciones o mensajes y se mostrará como online. Una vez que esté fuera de línea, no será posible que reciba nada de lo anterior. Cuando vuelva a conectarse, se actualizará y cargará los mensajes que recibió mientras estaba offline.

4.3. Vista

En lo que respecta al diseño de la aplicación, tendremos una única actividad a priori en la que, mediante fragmentos, iremos añadiendo la dinámica. El manejo del juego en sí se hará mediante un view personalizado, mostrando todos los elementos y actualizando sus respectivas posiciones cada cierto tiempo.
En la primera pantalla el usuario se podrá loguear o registrar. Introducirá su email y su contraseña. Una vez dentro, en la pantalla principal se mostrarán varios botones:

- Jugar: Permitirá iniciar el juego en modo solitario o en línea.
- Ajustes: Mostrará un menú con varias opciones que modificar, como cambiar la orientación de la pantalla.
- Invitar: Abrirá un desplegable para poder iniciar una partida con otro usuario.

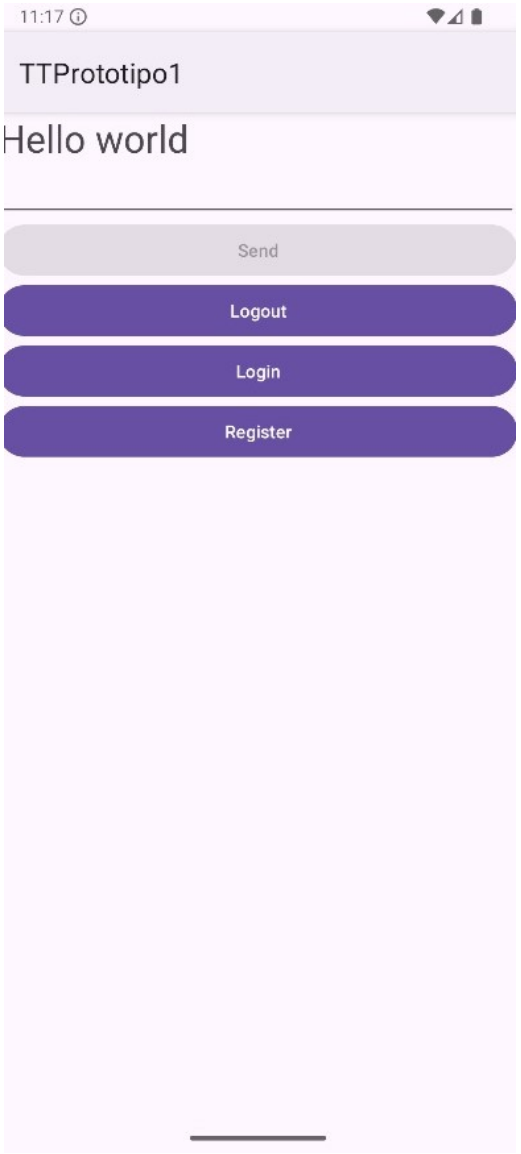

Figura 1: Figura 1

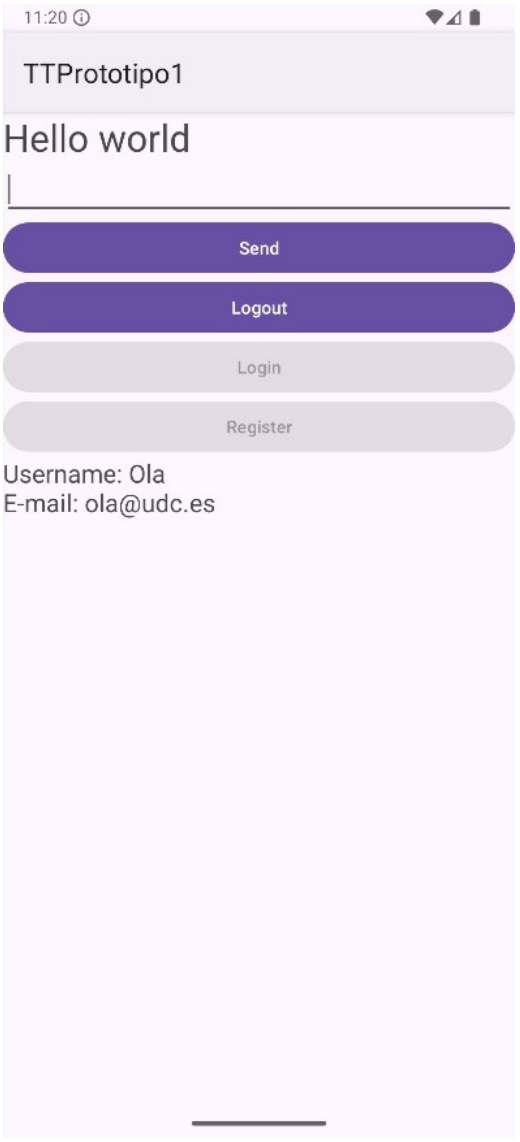

Figura 2: Pantalla una vez logueado

Figura 3: Pantallas de la primera iteración

4.4. Comunicaciones

La comunicación en este juego es una parte fundamental. Para ello, buscamos un sistema de baja latencia. Para esta conexión, se contará con un servidor central al que cada cliente se conectará por internet (Firebase), de esta forma, se gana en rango de uso (rango global). Existe un claro problema y es que el servidor del que se va a hacer uso es gratuito y, por lo tanto, la latencia va a ser bastante alta, pero hemos llegado a la conclusión de que es no es crítico para este proyecto porque, como cada usuario en una partida solo va a ver su pantalla a priori. En el caso de que ambos jugadores estén juntos físicamente, la latencia sería notable.

4.5. Sensores

Se hace uso de la pantalla táctil, con este se implementará el movimiento del mazo en cuestión. Es un dispositivo que detecta y responde al contacto físico o la presión sobre su superficie, en este caso no se hará uso de la detección de la presión. Para la implementación del rebote se puede tener o no en cuenta la velocidad con la que se desplaza en contacto con el panel táctil pero no es algo que vayamos a implementar en nuestro caso desde a priori.

4.6. Trabajo en background

Se enviarán los datos de la posición y ángulo de la pelota, cada X tiempo. Se usa Firebase para trabajar con una comunicación y latencia en tiempo real. Para controlar la velocidad de actualización de la pantalla y evitar sobrecargar la CPU, usaremos un servicio que recibirá las coordenadas del servidor. Con cada nueva información, se modificará la posición de la pelota en la pantalla.

5. Arquitectura Propuesta

Ahora mismo es una aplicación monolítica y cambiará a CLEAN Arquitecture. En esta primera iteración hay una actividad principal que incluye todo, user y UsersAdapter para el ListView que muestra los usuarios conectados.

6. Diseño e implementación de la 1ª iteración

Se ha creado una aplicación con una MainActivity, UsersAdapters y User. Partimos de un diseño monolítico, que en un futuro pasará a ser CLEAN Arquitecture. En la MainActivity está la mayor parte del código, como es la UI. La interfaz incluye en la parte superior un TextView que mostrará los mensajes que recibamos

de otros jugadores. Después, un EditText donde escribir el mensaje que queremos enviar a otros jugadores. Más abajo, están cuatro botones. Por último, un TextView con nuestro nombre de usuario y el email. Este último TextView no mostrará nada si el usuario no está logueado. Además, se ha añadido toast para informar al usuario si hay problemas al iniciar sesión, conectarse... A continuación se explica el funcionamiento de los botones:

- Send: muestra un desplegable con los usuarios conectados a los que podemos enviar un mensaje. Deshabilitado si el usuario no ha iniciado sesión.
- Logout: permite cerrar sesión.
- Login: iniciar sesión. Muestra AlertDialog para introducir los datos, correo y contraseña. Además, aparece un botón para cambiar a register. Deshabilitado si el usuario ya ha iniciado sesión.
- Register: registrar usuario. Muestra un AlertDialog similar al de login y también tiene el botón para alternar a login. Deshabilitado si el usuario ya ha iniciado sesión.

UsersAdapters y Users es usado para la implementación del ListView que muestra los usuarios conectados (tras pulsar Send). La app se conecta a Firebase, donde se han creado varios usuarios para comprobar que funciona correctamente, con su login y logout, y se ha implementado el envío de mensajes, lo que representaría a los primeros hitos. Finalmente, también se ha inicializado la base de datos haciendo uso del json.

7. 1^{ras} pruebas

Se han usado dos emuladores en la misma máquina para comprobar el funcionamiento de la aplicación. Se han comprobado el registro de usuarios, el login, el logout, la creación de datos en la base de datos en tiempo real y el paso de mensajes usando Firebase. Adicionalmente, se han usado varios equipos distintos, y se han comunicado varias apps de emuladores y equipos diferentes.

8. Bibliografía

Referencias

[1] libGDX Team, "libgdx: Cross-platform game development framework." libGDX Official Website: https://libgdx.com/, 2024.

[2] Software, E., "Kryonet: A java network library based on kryo." GitHub Repository: https://github.com/EsotericSoftware/kryonet, 2014.

[3] libGDX Team, "Box2d: A 2d physics engine for games integrated with libgdx." libGDX Documentation: https://libgdx.com/wiki/graphics/2d/box2d, 2024.

[4] Ribon, A., "Universal tween engine: An open-source java tween engine." GitHub Repository: https://github.com/AurelienRibon/universal-tween-engine, 2015.

[5] Google, "Firebase documentation." Firebase Documentation: https://firebase.google.com/docs?hl=es-419, 2023.

[6] Google, "Android developers: Guía oficial para desarrolladores." Sitio web oficial de Android Developers: https://developer.android.com/?hl=es-419.

[7] Canva, "Canva: Herramienta de diseño gráfico en línea." Canva offical website: https://www.canva.com/es_es/.