UNIVERSITAT POLITÈCNICA DE VALÈNCIA

grado en ingeniería de tecnologías y servicios de telecomunicación

itst

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería de Telecomunicación

Aplicaciones Telemáticas

Desarrollo de una aplicación móvil

Daniel Silva Romero[(1)]
Carlos Soria Mora[(2)]
Alejandro Cano Segovia[(3)]
Hugo Beltrán Sanz[(4)]

(1) dsilrom@teleco.upv.es, (2) csormor@teleco.upv.es, (3) acanseg1@teleco.upv.es, (4) hbelsan@teleco.upv.es

Curso 2023-2024

Abstract

This project will design and implement a mobile application in Android Studio using the Java programming language. The chosen application is the tic-tac-toe game, where the user can play against the machine or against another user on the same device. It has a simple interface, which allows for easy gameplay. The machine works by means of a small pseudo-random algorithm that decides which square to mark. Also, use has been made of: intents with *bundles in opening activities*, *strings*, *drawables* and on *click buttons* (with and without the use of the on click listener interface).

1 Introducción

Se ha diseñado una aplicación móvil basada en el juego del Tres en Raya, donde dos jugadores deben marcar con un símbolo (círculo o equis) las seis casillas de un tablero tres por tres. Cada turno, un jugador marca una única casilla. El jugador gana si consigue tener tres casillas consecutivas. En caso de no ganar ningún jugador, ocurre un empate.

Esta metodología se ha implementado haciendo uso de clases para *activities* como: `MainActivity` o `JuegoActivity`; *clases* personalizadas como: `Jugador`, `Tablero` o `Juego3`; o *interfaces* como: `View.OnClickListener`, implementadas en las dos clases.

La aplicación consta de dos modos de juego:

- **JUGADOR 1 VS JUGADOR 2.** Donde dos jugadores, en el mismo dispositivo, juegan una partida por turnos.
- **JUGADOR VS MÁQUINA.** Donde el jugador juega contra una "máquina"; un algoritmo pseudo-aleatorio que determina donde marcar la casilla.

Además de otra opción, **SALIR DEL JUEGO**, que nos permite salir de la aplicación.

2 Estado del arte

Las aplicaciones de Tres en Raya en Android han evolucionado para incluir características que mejoran la experiencia del usuario y agregan complejidad al juego. Las implementaciones más comunes incluyen los modos de juego de jugador contra jugador y jugador contra máquina, con interfaces simples y funcionales.

Las aplicaciones modernas presentan una interfaz intuitiva, generalmente con botones claramente etiquetados y un diseño de tablero fácil de usar.

Los modos de juego suelen incluir opciones para jugar contra otro jugador en el mismo dispositivo o contra una inteligencia artificial (IA). La inteligencia artificial en estas aplicaciones varía en complejidad.

Por todo ello, estas serán las principales características que buscamos en la aplicación: el diseño de una interfaz simple e intuitiva, dos modos de juego y una dificultad que vendrá dada por la "máquina".

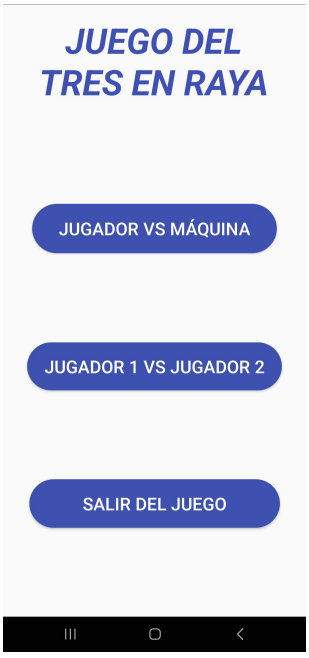

Figure 1: Pantalla principal

3 Metodología

Para la realización de este proyecto nos hemos basado [1] en los recursos web que nos proporciona el profesor Borja Molina Zea [2]. La metodología a seguir será la siguiente.

En primer lugar, se diseñarán dos *layouts* con *constraints layouts* entre sus *views*. De esta forma, evitamos anidar múltiples *layouts*, a cambio de tener restricciones como alineaciones, márgenes, dimensiones, etc. entre las *views*. Las propiedades de estas *views* se verán modificadas, ya sea para indicar texto o actuar de botón.

Para ello, se hará uso de la interfaz `View.OnClickListener` para los botones de la pantalla principal (Figura 1) y la pantalla del juego (Figura 2). Implementar esta interfaz permite definir qué sucede cuando el usuario hace clic en una *view*.

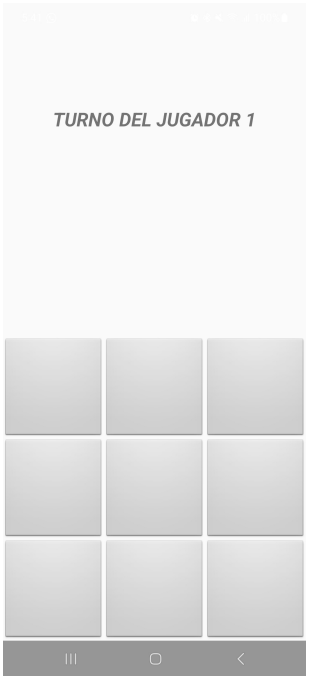

Figure 2: Pantalla del juego

Las casillas del tablero serán botones que harán uso de la propiedad `OnClick` de Android Studio. Esto permite definir un método que se llamará cuando se haga clic en un botón. Los botones del tablero llaman al método `onClick` y al método `PulsarBoton`, lo que simplifica el manejo de los clics y permite implementar la lógica del juego de manera centralizada.

También realizaremos aperturas entre *activities*, que permite cambiar a una nueva pantalla desde la actual. Esto se realiza utilizando `intent`, mensajes de comunicación que pueden solicitar una acción de otra actividad dentro de la misma aplicación. Para pasar datos entre las actividades haremos uso de un `Bundle`, en concreto, pasaremos el valor de la variable `ModoDeJuego`.

Finalmente, haremos uso de recursos personalizados como: `Drawable` para imágenes (Figura 3), `Raw` para los sonidos, `Strings` para el texto y `Theme` para los colores.

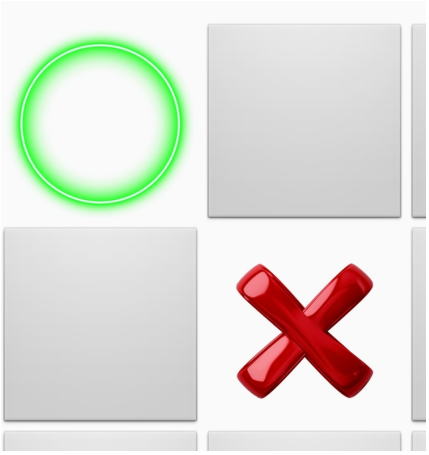

Figure 3: Imágenes usadas en el `Drawable`

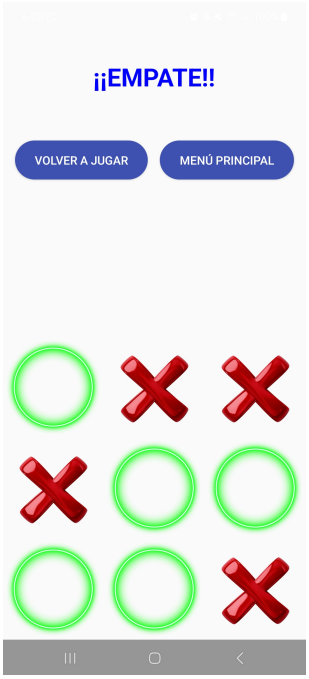

Figure 4: Tema y colores usados

4 Diseño e implementación

Una vez detallada la metodología, procedemos a diseñar la aplicación [3]. Para ello, haremos la siguiente descripción algorítmica.

Usaremos dos *clases* para las *activities*, se tratan de `MainActivity` y `JuegoActivity`. En la primera, diseñaremos la pantalla del menú principal (Figura 1). Mientras que en la segunda, diseñaremos la pantalla con el tablero del juego (Figura 2).

También se incluyen *clases* como `Jugador`, `Tablero` o `Juego3`. Inicialmente, la aplicación se diseñó sin hacer uso de estas clases, posteriormente, se hizo un intento de adecuarlo a una programación orientada a objetos. `Jugador` crea dos objetos de la clase jugador (uno por cada jugador). `Tablero` representa un *array* de nueve elementos. Y `Juego3` crearía un objeto `juego3raya`, donde se le pasarían de parámetros los objetos `jugador1` y `jugador2`. E internamente se crea un objeto `Tablero`. Por diversos motivos, principalmente la falta de tiempo, no se ha podido finalizar este tipo de diseño.

Además del método `onClick`, implementamos el método `iniciarPartida`: se encarga de iniciar una nueva actividad `JuegoActivity` desde la actividad actual, pasando un valor de configuración (modo de juego) a la nueva actividad mediante un `Intent` y un `Bundle`. Esto asegura que el modo de juego seleccionado en la actividad actual se pase a la nueva actividad para que pueda ser utilizado allí.

En la clase `JuegoActivity`, los *views* utilizados son: `TextView` (en `TextoGanador` y `TextoTurnos`) y `Button` (en `buttonRestart y buttonMenu`).

También inlcuiremos otros métodos:

- `PulsarBoton`: maneja los eventos de clic en los botones del tablero de un juego de Tres en Raya. Se encarga de actualizar el estado del tablero, cambiar los turnos, y comprobar el estado del juego después de cada movimiento. Este método puede entenderse de manera más visual en el diagrama de la Figura 5.

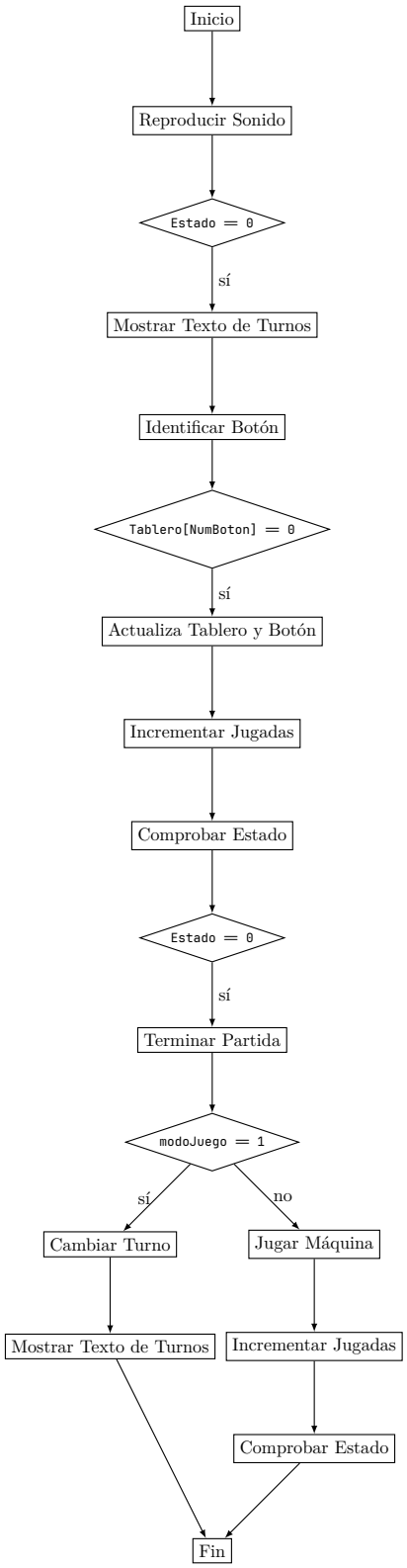

Figure 5: Diagrama de flujo del método `PulsarBoton`

- `Maquina`: es el algoritmo pseudo-aleatorio de la "máquina", este consiste en colocar la marca de la máquina en el centro del tablero si está disponible. Si el centro está ocupado, elige una esquina aleatoria que esté vacía. Si las esquinas están ocupadas, elige una posición aleatoria en el tablero que esté vacía. Finalmente, actualiza la interfaz de usuario y el estado del tablero con la posición elegida. En el diagrama de flujo de la Figura 6 podemos ver su funcionamiento.

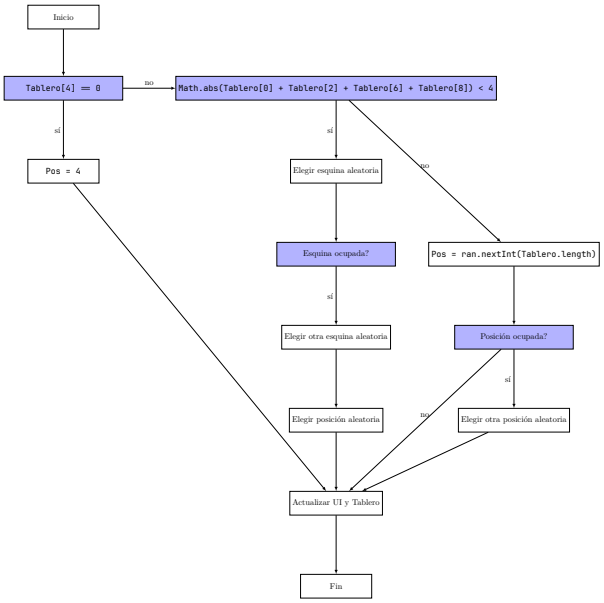

Figure 6: Diagrama de flujo del método `Maquina`

- `TerminarPartida`: actualiza la interfaz al finalizar una partida de un juego, mostrando el texto y el ícono adecuados según el resultado. Inicializa la ficha de victoria con un ícono predeterminado. Si el estado del juego es 1 (gana el jugador 1), muestra el texto del ganador, oculta los turnos, reproduce el sonido de victoria, actualiza el texto y el color del ganador, y muestra los botones de reinicio y menú. Si el estado es -1 (gana el jugador 2), según el modo de juego, reproduce el sonido de victoria del jugador 2 o el de derrota, actualiza el texto del ganador o perdedor, el color del texto, muestra los botones de reinicio y menú, y establece la ficha de victoria en rojo. Si el estado es 2 (empate), muestra el texto de empate, reproduce el sonido de empate, actualiza el texto y el color del empate, y muestra los botones de reinicio y menú. Finalmente, recorre las posiciones ganadoras y actualiza el fondo de los botones correspondientes con la ficha de victoria.

- `ComprobarEstado`: verifica el estado del juego para determinar si hay una combinación ganadora o un empate. Inicializa `NuevoEstado` a 0 y luego chequea varias combinaciones posibles en el tablero. Si la suma absoluta de cualquier combinación de tres casillas (horizontal, vertical o diagonal) es igual a 3, establece `NuevoEstado` con el valor de `Turno` y asigna `PosicionGanadora` con los índices correspondientes de las casillas ganadoras. Si no hay combinaciones ganadoras y se han realizado 9 jugadas, establece `NuevoEstado` a 2 indicando un empate. Finalmente, retorna `NuevoEstado` con el resultado del chequeo.

- `resetGame`: reinicia el estado del juego a sus valores iniciales. Establece el tablero a una matriz de ceros, reinicia el número de jugadas a 0, el turno al jugador 1, y la posición ganadora a valores no válidos. Oculta el texto del ganador y muestra el texto de los turnos, actualizándolo para indicar que es el turno del jugador 1. También oculta los botones de reinicio y menú. Si hay un estado de victoria o empate, detiene y reinicia los sonidos correspondientes (`P1win`, `P2win`, `Loser`, `Empate`). Luego, restablece el estado a 0. Finalmente, recorre los botones del tablero y les devuelve su apariencia predeterminada.

- `goToMenu`: cambia la actividad actual a la actividad principal (`MainActivity`) y asegura que cualquier sonido de victoria o empate se detenga y reinicie antes de finalizar la actividad actual. Primero, crea un `Intent` para iniciar `MainActivity` y lo inicia con `startActivity(intent)`. Luego, detiene y reinicia los sonidos correspondientes según el estado del juego (`Estado`). Si el estado es 1 (victoria del jugador 1) o -1 (victoria del jugador 2), pausa y reinicia `P1win`. Si el estado es -1 y el modo de juego es 1, pausa y reinicia `P2win`, de lo contrario pausa y reinicia `Loser`. Si el estado es 2 (empate), pausa y reinicia `Empate`. Finalmente, llama a `finish()` para cerrar la actividad actual.

Queda por determinar las interfaces que se implementan en la aplicación. Por un lado, tenemos que `MainActivity` implementa la interfaz `View.OnClickListener`, de esta form, se implementa a su vez el método `onClick`. Su función es capturar el clic en los botones de la pantalla principal. Por otro `JuegoActivity` también implementa la interfaz `View.OnClickListener`, implementando a su vez el método `onClick`. La función es exactamente la misma, capturar el clic de los botones de la pantalla, en este caso, la pantalla del juego. De esta forma, podemos ir a la pantalla del juego o a la pantalla principal y viceversa.

Finalmente, se ha implementado un botón para salir de la aplicación utilizando una condición que verifica si el ID del elemento coincide con `R.id.exit_game`. Si es así, se llama al método `finishAffinity()`, que cierra la actividad actual y todas las actividades relacionadas, finalizando completamente la aplicación.

Como últimos detalles se ha añadido sonido al hacer clic en los botones y en las casillas. Además, los botones tienen una animación en la cual el botón se expande y se contrae, de forma que el jugador sabe que lo ha pulsado. Y por último, se le indica al jugador con una pista de audio cuando ha ganado, empatado o perdido.

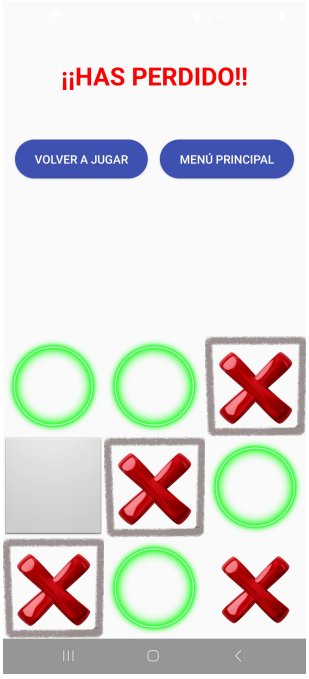

Figure 7: Partida perdida

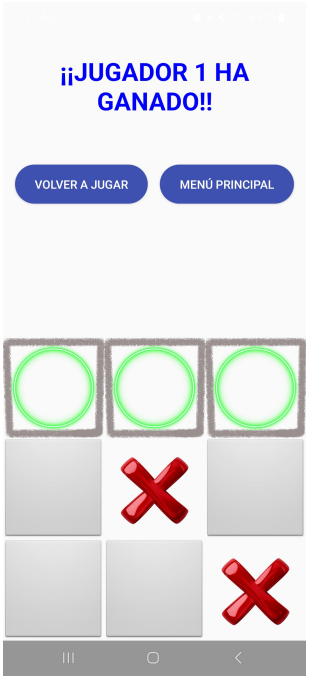

Figure 8: Partida ganada

References

[1] YouTube: "Profe de Informática Borja Molina, Programación para Android"

[2] Borja Molina Zea, Profesor de FP de infórmatica: DAW, DAM, ASIR y SMR

[3] Developers, Guías para desarrolladores.