Desarrollo de Apps Nativas de VideoJuegos para Android

Dr. Marco Aurelio Nuño-Maganda

Universidad Politecnica de Victoria

https://github.com/mnunom-upv/Curso_Desarollo_Aplicaciones_Moviles_2023

Noviembre 2025

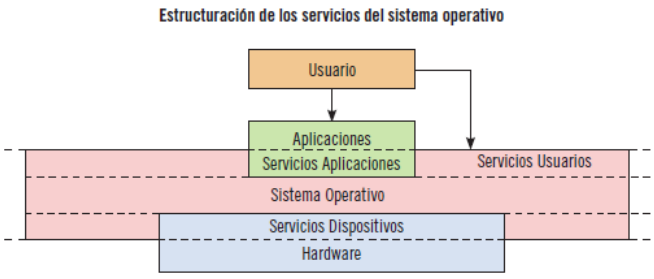
Contenido

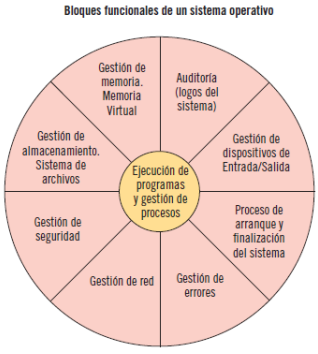
Contenido

- Es la actividad de desarrollar una aplicación específicamente para teléfonos inteligentes.
- Estas aplicaciones se encuentran preinstaladas en el teléfono o pueden ser instaladas por el usuario mediante una tienda de aplicaciones (App Store o Google Play)
- Las tareas que tradicionalmente hacíamos en la PC ahora están migrando hacia el teléfono inteligente
- Principales sistemas operativos móviles: Android, iOS,
- Enfocados principalmente en el desarrollo de aplicaciones NATIVAS.
- Lenguajes de programación: Java, Kotlin.

Sistema Operativo

Un Sistema Operativo (SO) es un programa (software) que al arrancar la computadora** se encarga de gestionar todos los recursos del sistema informático permitiendo así la comunicación entre el usuario y la computadora.





https://reader.digitalbooks.pro/content/preview/books/38230/book/OEBPS/Text/c1.html

Sistemas Operativos para PCs

Telefono Celular No-inteligente vs Telefono Celular Inteligente

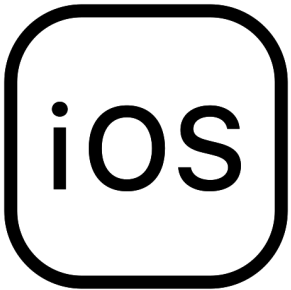
Teléfono No-inteligente

- Su funcionalidad principal era la comunicación (llamadas o mensajes) a través de la red celular (GSM)

Teléfono inteligente

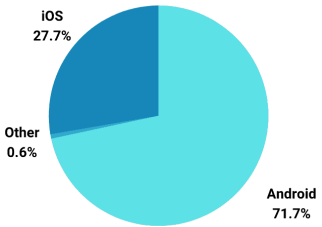
- Interfaz de entrada: Pantalla Touch (a color, de alta definición)
- Conexión a Internet: WiFi, GSM (4G o 5G)
- Comunicación con otros dispositivos: Bluetooth, NFC
- Cámaras (Frontal y Posterior)

Sistemas Operativos para Telefonos Inteligentes



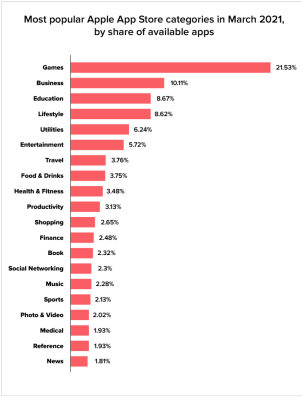
Android

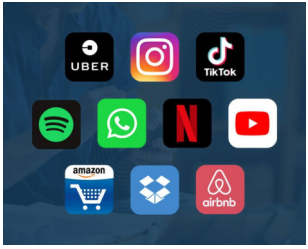
- Android es un sistema operativo móvil basado en Linux
- Principalmente orientado a dispositivos de pantalla táctil (Smartphone, tablets, smartwatches, etc)
- Fue desarrollado por Android Inc (Adquirida por Google en 2005)
- Vinculado con un grupo de empresas (HTC, Sony, Motorola, Samsung, LG, Lenovo, entre otras) para la creación de un SO común para sus dispositivos
- A la fecha (Q1 2023), los teléfonos con SO Android concentran mas del 70% del mercado global.



Aplicaciones Móviles

- Ejecutadas en el teléfono
- La entrada de datos es mediante un teclado “virtual”
- El apuntador del raton es la pantalla
- Incluyen una interfaz de usuario gráfica (GUI)
- Es posible descargar miles de éstas en nuestros dispositivos





https://www.netsolutions.com/insights/top-10-most-popular-apps-2018/

Android Studio

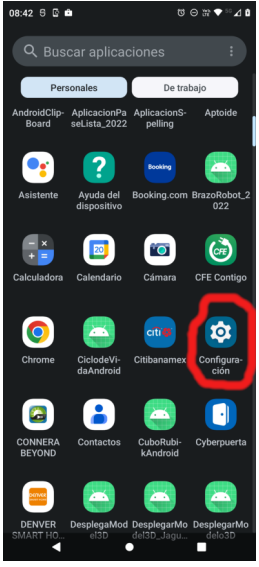
- Android Studio es un entorno oficial de desarrollo integrado (IDE) para el sistema operativo Android de Google
- La primera versión se libera en el año 2013, siendo el lenguaje de programacion Java
- En 2019, se reemplaza el lenguaje oficial de desarrollo por Kotlin, aunque Java todavía es soportado
- Es gratis, se puede descargar e instalar en cualquier computadora sin importar el sistema operativo (Windows, Linux y MacOS) `https://developer.android.com`

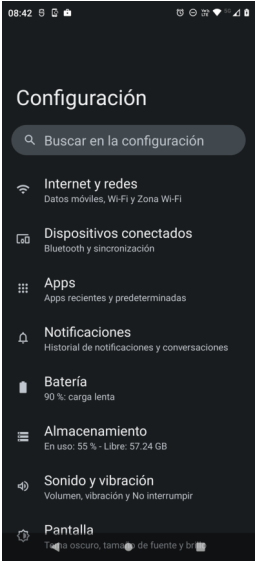
Contenido

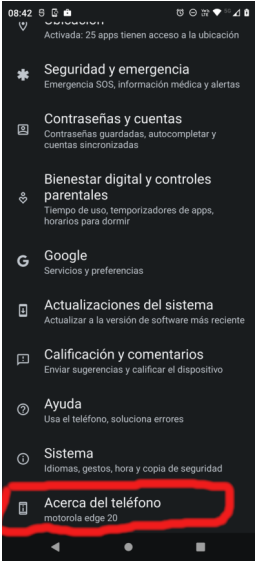
Configurar telefono inteligente en modo de desarrollador (0)

- Es necesario buscar en las opciones de configuración, en el apartado acerca del telefono ubicar el número de compilación
- Dar 5 taps (toques) sobre el número compilación. Debera aparecer un mensaje que diga que estas a X taps de ser un desarrollador
- Lo anterior habilita un nuevo menú en el apartado sistema dentro de opciones de configuración con título “opciones para desarrolladores”
- Debe habilitarse tanto “opciones para desarrolladores” como la opción “depuración USB”

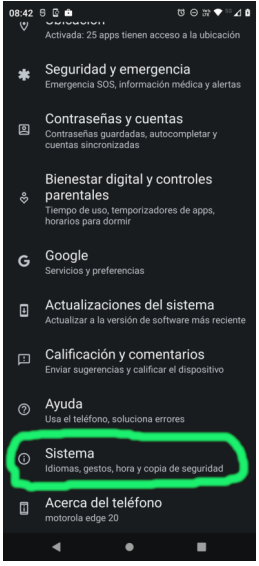
Configurar telefono inteligente en modo de desarrollador (1)

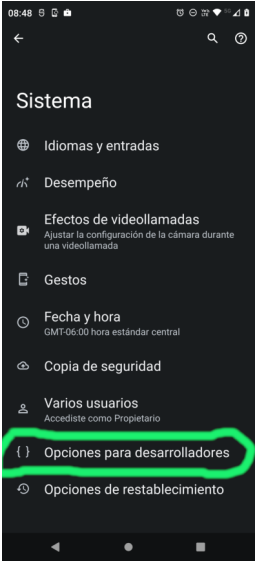


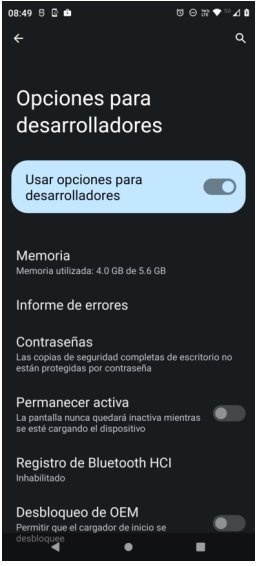


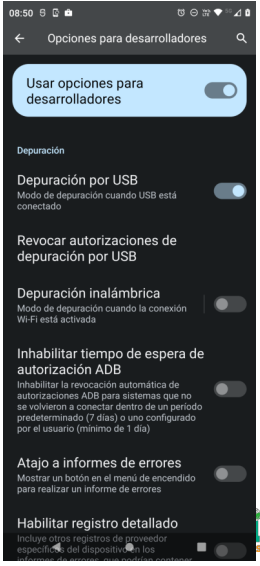


Configurar telefono inteligente en modo de desarrollador (2)







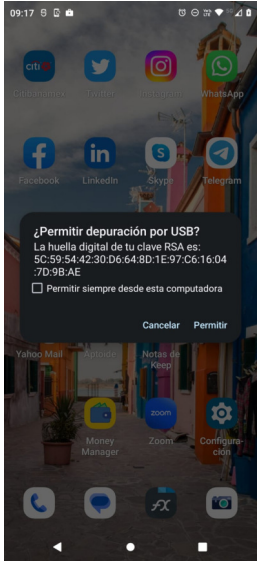


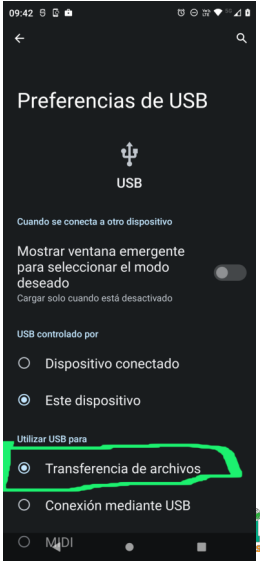
Configurar telefono inteligente en modo de desarrollador (3)

- Al habilitar la depuracion, aparece un mensaje de advertencia
- Una vez que hayas terminado tus pruebas, se recomienda deshabilitar este modo de depuración
- Para poder instalar una aplicación desarrollada con Android Studio, se debe conectar el teléfono inteligente a la computadora usando un cable USB

Conectar smartphone a la computadora (Cable USB)

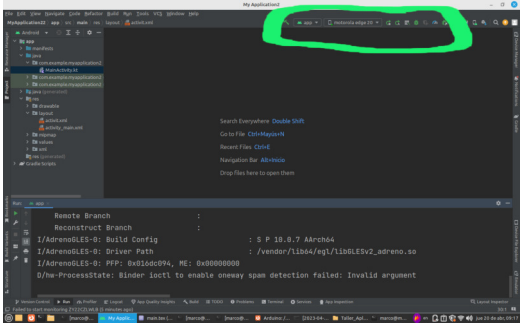
- Al conectar el dispositivo por primera vez, aparece un mensaje de confirmación en el dispositivo
- Configurar el telefono para que se conecte en modo de transferencia de archivos (por defecto, esta cargando)

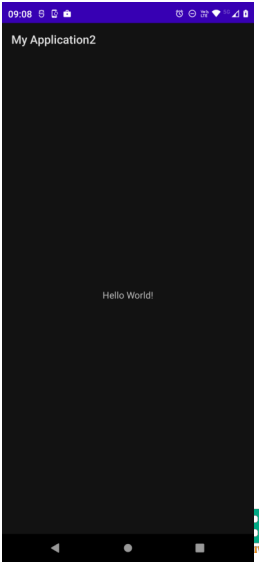




Ejecutar proyecto en telefono inteligente

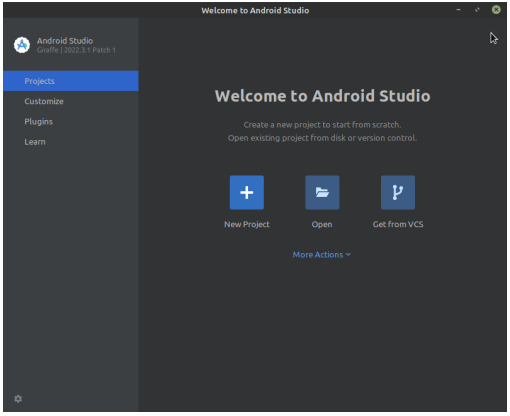
- Una vez conectado el telefono, debe aparecer el modelo en la parte superior (lado derecho) de tu proyecto de Android Studio
- Para instalar la aplicacion en el telefono, deber dar click en una flecha verde justo a un lado de nombre del telefono (sean pacientes, puede tardar un poco la primera vez)





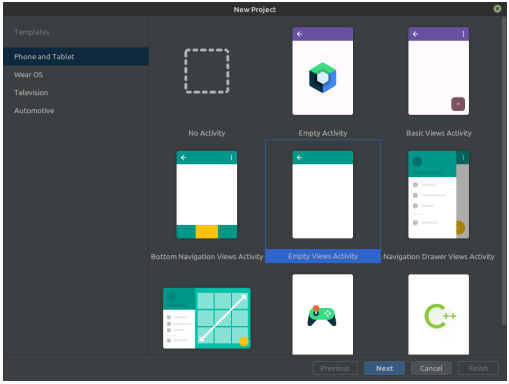
Crear un primer proyecto (1)

- Buscar el icono de la aplicacion y dar doble click
- En caso de que requiera algunas actualizaciones, ser paciente, puede tardar.
- Aparecera una ventana como la mostrada, seleccionar la opción “New project”



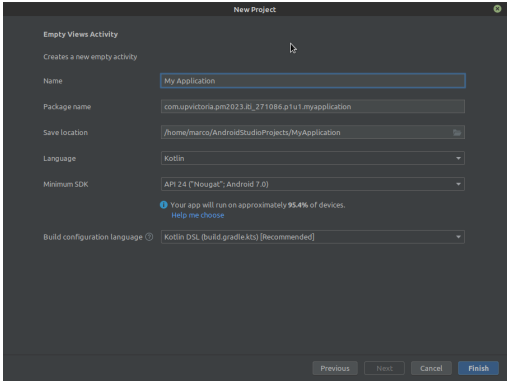
Crear un primer proyecto (2)

- Existen varios tipos de dispositivos para los que pueden crearse proyectos. Por defecto nos ubica en el primer grupo (Phone and Tablet)
- Seleccionar tipo de proyecto “Empty Views Activity” y dack click en Next



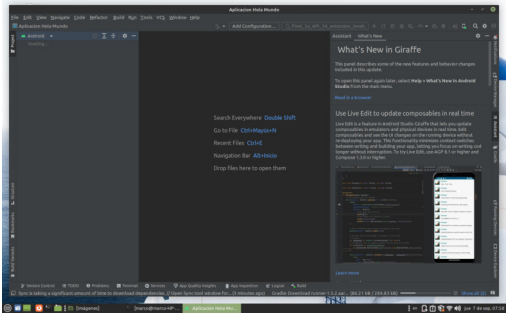
Crear un primer proyecto (3)

- Dejar los valores por defecto, solo se suguiere cambiar el nombre del proyecto
- **Debe estar seleccionado como lenguaje Kotlin**. De estar seleccionado uno diferente, cambiar
- Presionar el boton Finish



Crear un primer proyecto (4)

- La primera vez que se crea un proyecto, puede requerir la descarga de archivos necesarios. Le recomiendo ser paciente
- Ya una vez que el proyecto fue creado, aparece la ventana mostrada



Contenido

Juegos a implementar

- La implementación de juegos como **Tic-Tac-Toe** y **Pong** en este curso/tutorial tiene gran relevancia porque permite introducir de manera práctica y motivante los fundamentos esenciales del desarrollo interactivo.
- Ambos proyectos son sencillos en su lógica y mecánica, pero abarcan conceptos clave como el manejo de gráficos 2D, detección de colisiones, control de eventos táctiles, estructuras de datos, condicionales, bucles, y gestión del estado del juego.
- Además, su simplicidad favorece la comprensión del ciclo de vida de una aplicación móvil y fomenta el pensamiento lógico, la resolución de problemas y la creatividad.
- En conjunto, estos juegos sirven como laboratorios didácticos ideales para que los estudiantes consoliden sus bases en programación mientras desarrollan productos tangibles, visuales y motivadores.

Tic-Tac-Toe: Juego de Estrategia Simple

Mecánica y Tablero

- Es un juego para **dos jugadores** que se desarrolla en un tablero de 3×3 casillas.
- Un jugador marca con "equis" y el otro con "círculo".

Objetivo del Juego

- El objetivo es ser el primero en colocar tres de sus símbolos de forma consecutiva.
- Las combinaciones ganadoras pueden ser **horizontales, verticales o diagonales**.

Desarrollo de la Partida

- Los jugadores se turnan para colocar un símbolo en una casilla vacía.
- Si el tablero se llena y nadie logra la línea de tres, la partida se declara **empate** (o *gato*).

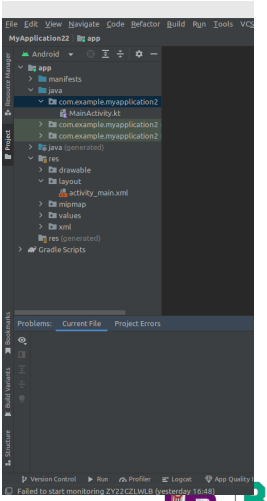
Análisis Estratégico

Se usa a menudo como una herramienta pedagógica para introducir conceptos de **teoría de juegos** y búsqueda de árboles de decisión en programación.

Carpetas del proyecto

Carpetas y archivos importantes

- *Manifest* - Por el momento es de interes
- *Java* - Código fuente - Dentro hay un archivo *MainActivity.kt*
- *Res* - Recursos (imágenes) y definición de interfaces. Dentro hay una carpeta llamada *layout*, con un archivo *activity_main.xml*



Agregar codigo a la interfaz

Apariencia

Archivo **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

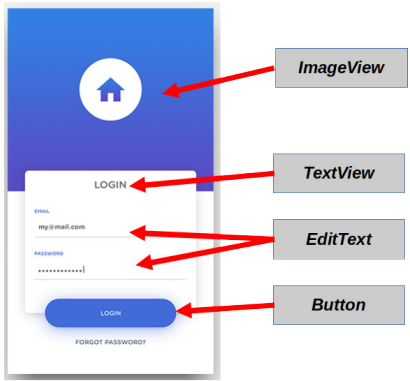
Comportamento

Archivo **MainActivity.kt**

```
package com.example.primeraplicacionandroid
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Interfaz de Usuario

- Define los controles de interfaz que la aplicacion muestra
- Estos controles pueden estar agrupados en contenedores
- El archivo **activity_main.xml** es la definicion del layout (la organizacion de dichos componentes)



Primer modificación

Archivo Original **activity_main.xml**

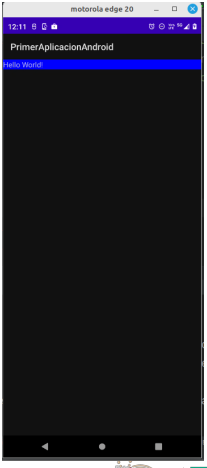
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".MainActivity">
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintEnd_toEndOf="parent"
15         app:layout_constraintStart_toStartOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17 </androidx.constraintlayout.widget.ConstraintLayout>
```

Archivo Modificado **activity_main.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      android:orientation="vertical"
9      tools:context=".MainActivity">
10     <TextView
11         android:background="#00ffff"
12         android:layout_width="match_content"
13         android:layout_height="wrap_content"
14         android:text="Hello World!" />
15 </LinearLayout>
16
17
```

Primer modificación

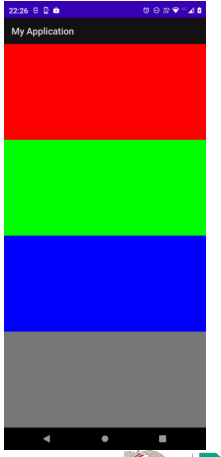
1. Reemplazar *androidx.constraintlayout.widget.ConstraintLayout* por *LinearLayout*
   - Tipo de contenedor donde los componentes se agregan secuencialmente
2. Agregar propiedad *android:orientation="vertical"* al *LinearLayout* principal
   - Los componentes seran ordenados verticalmente
3. Cambiar la propiedad *android:layout_width="wrap_content"* por *android:layout_width="match_parent"*
   - El ancho de TextView abarcar toda la pantalla
4. Agregar la propiedad *android:background="#00ffff" al textview -*
   - La cadena "#00ffff" define el color azul. Se pueden probar con otros buscando en Internet un generador de colores hexadecimal y cambiando por otros de tu preferencia



Fase 1: Dividir la pantalla en contenedores verticales

Archivo **activity_main.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent" android:layout_height="match_parent"
6      android:orientation="vertical" tools:context=".MainActivity">
7      <LinearLayout android:background="#ff0000"
8          android:layout_width="match_parent" android:orientation="horizontal"
9          android:layout_height="0dp"
10         android:layout_weight="1">
11     </LinearLayout>
12     <LinearLayout android:background="#00ff00"
13         android:layout_width="match_parent" android:orientation="horizontal"
14         android:layout_height="0dp"
15         android:layout_weight="1">
16     </LinearLayout>
17     <LinearLayout android:background="#0000ff"
18         android:layout_width="match_parent" android:orientation="horizontal"
19         android:layout_height="0dp"
20         android:layout_weight="1">
21     </LinearLayout>
22     <LinearLayout android:background="#777777"
23         android:layout_width="match_parent" android:orientation="horizontal"
24         android:layout_height="0dp"
25         android:layout_weight="1">
26     </LinearLayout>
27 </LinearLayout>
```

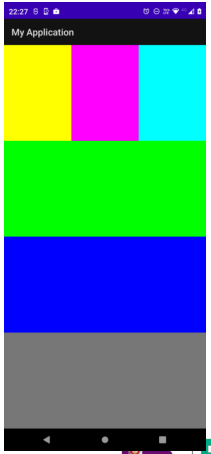


Fase 2: Agregar un contenedor horizontal dentro de uno vertical

Este fragmento de código sustituye a las lineas 7 a 10 del *activity_main.xml*

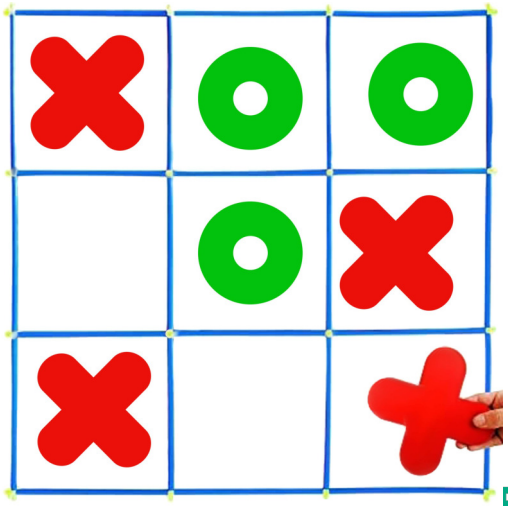
Porción del Layout que incluye tres contenedores verticales

```
1  <LinearLayout android:background="#ff0000"
2      android:layout_width="match_parent" android:orientation="horizontal"
3      android:layout_height="0dp"
4      android:layout_weight="1">
5      <LinearLayout android:background="#ffff00"
6          android:layout_width="0dp" android:orientation="vertical"
7          android:layout_height="match_parent"
8          android:layout_weight="1">
9      </LinearLayout>
10     <LinearLayout android:background="#ff00ff"
11         android:layout_width="0dp" android:orientation="vertical"
12         android:layout_height="match_parent"
13         android:layout_weight="1">
14     </LinearLayout>
15     <LinearLayout android:background="#00ffff"
16         android:layout_width="0dp"android:orientation="vertical"
17         android:layout_height="match_parent"
18         android:layout_weight="1">
19     </LinearLayout>
20 </LinearLayout>
21
```



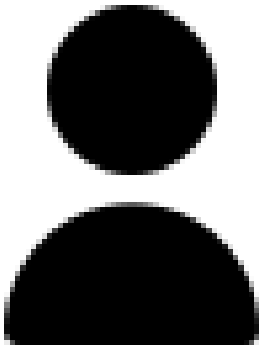
Aplicación TIC-TAC-TOE

- Juego para dos jugadores
- Cada jugador juega una de dos posibles combinaciones (circulo o tacha)
- El control debe decidir cuando un jugador gana o cuando hay empate
- Llevar el manejo de los turnos
- Requerimos un componente de interfaz que se comporte como un boton y que despliegue una imagen. Solución: *ImageButton*



Fase 3 (Parte 0): Agregar Imagenes que seran utilizadas como iconos

- Localizar la carpeta *drawable* dentro de la carpeta *res*
- De la carpeta Github, descargar las imagenes usuario.png, cruz.png y corazon.png y copiar estas imagenes en la carpeta antes mencionada

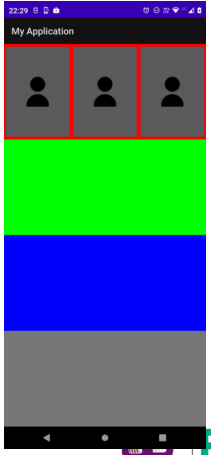


Fase 3 (Parte 1): Agregar 3 botones en orientación al contenedor horizontal al primer Contenedor

Este fragmento de código sustituye a las lineas 7 a 10 del *activity_main.xml*

Primer Layout Horizontal incluido tres *ImageButtons*

```
1   <LinearLayout android:background="#ff0000"
2       android:layout_width="match_parent"
3       android:layout_height="0dp"
4       android:layout_weight="1">
5       <ImageButton android:id="@+id/boton1_1"
6           android:src="@drawable/usuario"
7           android:layout_width="0dp"
8           android:layout_height="match_parent"
9           android:layout_weight="1"/>
10      <ImageButton android:id="@+id/boton1_2"
11          android:src="@drawable/usuario"
12          android:layout_width="0dp"
13          android:layout_height="match_parent"
14          android:layout_weight="1"/>
15      <ImageButton android:id="@+id/boton1_3"
16          android:src="@drawable/usuario"
17          android:layout_width="0dp"
18          android:layout_height="match_parent"
19          android:layout_weight="1"/>
20  </LinearLayout>
21
```



Fase 3 (Parte 2): Repetir para los otros contenedores

Este fragmento de código sustituye a las lineas 12 a 21 del *activity_main.xml*

Segundo y tercer Layouts horizontales, cada uno con tres *ImageButtons*

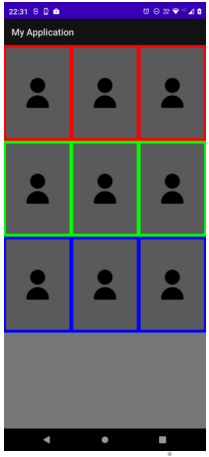
```
1   <LinearLayout android:background="#00ff00" android:layout_width="match_parent"
2       android:layout_height="0dp" android:layout_weight="1">
3       <ImageButton android:id="@+id/boton2_1"
4           android:src="@drawable/usuario" android:layout_width="0dp"
5           android:layout_height="match_parent" android:layout_weight="1"/>
6       <ImageButton android:id="@+id/boton2_2"
7           android:src="@drawable/usuario" android:layout_width="0dp"
8           android:layout_height="match_parent" android:layout_weight="1"/>
9       <ImageButton android:id="@+id/boton2_3"
10          android:src="@drawable/usuario" android:layout_width="0dp"
11          android:layout_height="match_parent" android:layout_weight="1"/>
12  </LinearLayout>
13  <LinearLayout android:background="#0000ff" android:layout_width="match_parent"
14      android:layout_height="0dp" android:layout_weight="1">
15      <ImageButton android:id="@+id/boton3_1"
16          android:src="@drawable/usuario" android:layout_width="0dp"
17          android:layout_height="match_parent" android:layout_weight="1"/>
18      <ImageButton android:id="@+id/boton3_2"
19          android:src="@drawable/usuario" android:layout_width="0dp"
20          android:layout_height="match_parent" android:layout_weight="1"/>
21      <ImageButton android:id="@+id/boton3_3"
22          android:src="@drawable/usuario" android:layout_width="0dp"
23          android:layout_height="match_parent" android:layout_weight="1"/>
24  </LinearLayout>
```

Fase 3 (Parte 3): Al ultimo contenedor agregar un TextView

Este fragmento de código sustituye a las lineas 22 a 26 del *activity_main.xml*

Último contenedor del Layout

```
1  <LinearLayout android:background="#777777"
2      android:layout_width="match_parent"
3      android:layout_height="0dp"
4      android:layout_weight="1">
5      <TextView android:id="@+id/textView_Pizarra_Estatus"
6          android:layout_width="match_parent"
7          android:layout_height="match_parent"
8          android:textSize="18dp"
9          android:gravity="center"/>
10 </LinearLayout>
```



Codigo Completo

P1

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical" tools:context=".MainActivity">
    <LinearLayout android:background="#ff0000"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">
        <ImageButton android:id="@+id/boton1_1"
            android:src="@drawable/usuario"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
        <ImageButton android:id="@+id/boton1_2"
            android:src="@drawable/usuario"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
        <ImageButton android:id="@+id/boton1_3"
            android:src="@drawable/usuario"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"/>
    </LinearLayout>
```

Codigo Completo (2)

P2

```
1   <LinearLayout android:background="#00ff00" android:layout_width="match_parent"
2       android:layout_height="0dp" android:layout_weight="1">
3       <ImageButton android:id="@+id/boton2_1"
4           android:src="@drawable/usuario" android:layout_width="0dp"
5           android:layout_height="match_parent" android:layout_weight="1"/>
6       <ImageButton android:id="@+id/boton2_2"
7           android:src="@drawable/usuario" android:layout_width="0dp"
8           android:layout_height="match_parent" android:layout_weight="1"/>
9       <ImageButton android:id="@+id/boton2_3"
10          android:src="@drawable/usuario" android:layout_width="0dp"
11          android:layout_height="match_parent" android:layout_weight="1"/>
12  </LinearLayout>
13  <LinearLayout android:background="#0000ff" android:layout_width="match_parent"
14      android:layout_height="0dp" android:layout_weight="1">
15      <ImageButton android:id="@+id/boton3_1"
16          android:src="@drawable/usuario" android:layout_width="0dp"
17          android:layout_height="match_parent" android:layout_weight="1"/>
18      <ImageButton android:id="@+id/boton3_2"
19          android:src="@drawable/usuario" android:layout_width="0dp"
20          android:layout_height="match_parent" android:layout_weight="1"/>
21      <ImageButton android:id="@+id/boton3_3"
22          android:src="@drawable/usuario" android:layout_width="0dp"
23          android:layout_height="match_parent" android:layout_weight="1"/>
24  </LinearLayout>
```

Codigo Completo

P3

```
1
2       <LinearLayout android:background="#777777"
3           android:layout_width="match_parent"
4           android:layout_height="0dp"
5           android:layout_weight="1">
6           <TextView android:id="@+id/textView_Pizarra_Estatus"
7               android:layout_width="match_parent"
8               android:layout_height="match_parent"
9               android:textSize="18dp"
10              android:gravity="center"/>
11      </LinearLayout>
12  </LinearLayout>
```

Fase 1: Crear variables para la interfaz

MainActivity.kt

```kotlin
package com.example.myapplication

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
// imports externos que se van air agregado (*1)

class MainActivity : AppCompatActivity() {

    // En esta seccion se declaran las variables de clase (*2)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Se deben llamar los metodos que inicializar las variables
        // o la interfaz de usuario (*3)

    }

    // En esta seccion se declaran las funciones de la clase (*4)

}

// Declaracion de otras clases (*5)
```

Concepto de Matriz

Matriz

- Colección ordenada de valores, conformada por filas y columnas
- Los elementos pueden ser identificados por la tupla (Fila, Columna), de manera análoga a un estante con fila (entepaño) o columna
- Por convención, la primera fila o columna tiene el indice cero

Fase 1: Crear variables para la interfaz

(*1)

```
1  import android.widget.ImageButton
2  import android.widget.TextView
3  import android.widget.Toast
4  import android.view.View
```

(*2)

```
1  lateinit var B1_1 : ImageButton
2  lateinit var B1_2 : ImageButton
3  lateinit var B1_3 : ImageButton
4  lateinit var B2_1 : ImageButton
5  lateinit var B2_2 : ImageButton
6  lateinit var B2_3 : ImageButton
7  lateinit var B3_1 : ImageButton
8  lateinit var B3_2 : ImageButton
9  lateinit var B3_3 : ImageButton
10 lateinit var TextViewPizarra : TextView
11 var Turno : Int = 1   // 1-Turno Cruz, 2-Turno Corazon
12 var numRows : Int = 3
13 var numCols : Int = 3
14 lateinit var matrix: Array<Array<CeldaGato?>>
15
16 var ConteoCruces : Int = 0  // # de Celdas Marcadas Cruz
17 var ConteoCorazones : Int = 0  // de Celdas Marcadas Corazon
18
```

(*3)

```
1
2  InicializarBotonesConIds()
3  AsignarListenerABotones()
4  CrearMatrizEstatus ()
5  ReiniciarControles()
6
```

(*4)

```
1  fun InicializarBotonesConIds () {
2      TextViewPizarra = findViewById(R.id.textView_Pizarra_Estatus)
3      B1_1 = findViewById (R.id.boton1_1)
4      B1_2 = findViewById (R.id.boton1_2)
5      B1_3 = findViewById (R.id.boton1_3)
6      B2_1 = findViewById (R.id.boton2_1)
7      B2_2 = findViewById (R.id.boton2_2)
8      B2_3 = findViewById (R.id.boton2_3)
9      B3_1 = findViewById (R.id.boton3_1)
10     B3_2 = findViewById (R.id.boton3_2)
11     B3_3 = findViewById (R.id.boton3_3)
12 }
```

Fase 1: Agregar un Listener

(*4)

```
fun AsignarListenerABotones () {
    B1_1.setOnClickListener (btnListener)
    B1_2.setOnClickListener (btnListener)
    B1_3.setOnClickListener (btnListener)
    B2_1.setOnClickListener (btnListener)
    B2_2.setOnClickListener (btnListener)
    B2_3.setOnClickListener (btnListener)
    B3_1.setOnClickListener (btnListener)
    B3_2.setOnClickListener (btnListener)
    B3_3.setOnClickListener (btnListener)
}
```

(*4)

```
    val btnListener = View.OnClickListener {
        val (fila, columna) = ObtieneFilaColumna (it)
        Toast.makeText(applicationContext,
            "PresionasteBoton ["+fila+","+columna+"]"+
                matrix[fila][columna]!!.deviceStatus,
            Toast.LENGTH_SHORT).show()
        TextViewPizarra.text = "Alguna informacion util"
    }
```

(*4)

```
fun CrearMatrizEstatus() {
    matrix = Array(numRows) { row ->
        Array(numCols) { col ->
            CeldaGato(row, col)
        }
    }
}
```

(*4)

```
fun ObtieneFilaColumna (it : View):  Pair<Int, Int> {
    var fila = 0;
    var columna = 0
    when (it.id) {
        R.id.boton1_1 -> { fila = 1;  columna = 1 }
        R.id.boton1_2 -> { fila = 1;  columna = 2 }
        R.id.boton1_3 -> { fila = 1;  columna = 3 }
        R.id.boton2_1 -> { fila = 2;  columna = 1 }
        R.id.boton2_2 -> { fila = 2;  columna = 2 }
        R.id.boton2_3 -> { fila = 2;  columna = 3 }
        R.id.boton3_1 -> { fila = 3;  columna = 1 }
        R.id.boton3_2 -> { fila = 3;  columna = 2 }
        R.id.boton3_3 -> { fila = 3;  columna = 3 }
        else -> {          fila = -1; columna = -1 }
    }
    return Pair(fila-1, columna-1)
}
```

Fase 1: Reiniciar Controles a su Estado Inicial (1)

(*4)

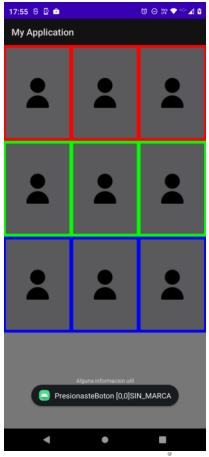
```
fun ReiniciarControles() {
        for (i in 0..numRows - 1) {
            for (j in 0..numCols - 1) {
                matrix[i][j]!!.deviceStatus = "SIN_MARCA"
            }
        }
        B1_1.setImageResource(R.drawable.usuario)
        B1_2.setImageResource(R.drawable.usuario)
        B1_3.setImageResource(R.drawable.usuario)
        B2_1.setImageResource(R.drawable.usuario)
        B2_2.setImageResource(R.drawable.usuario)
        B2_3.setImageResource(R.drawable.usuario)
        B3_1.setImageResource(R.drawable.usuario)
        B3_2.setImageResource(R.drawable.usuario)
        B3_3.setImageResource(R.drawable.usuario)
        Turno = 1  // 1 - Turno Cruz, 2 - Turno Corazon
        ConteoCruces = 0  // Numero de Celdas MArcadas con Cruz
        ConteoCorazones = 0  // Numero de Celdas Marcadas con Gato

        TextViewPizarra.text = "Inicio del Juego: " +
                "\n Conteo Cruces: " + ConteoCruces +
                "\n Contro Corazones: " + ConteoCorazones
    }
```

Fase 1: Agregar un Listener

(*5)

```
1  class CeldaGato (val row: Int, val col: Int) {
2      var deviceStatus = "SIN_MARCA"
3      constructor(row: Int, col: Int, statusCode: Int)
4              : this(row, col) {
5          deviceStatus = when (statusCode) {
6              0 -> "TACHA"
7              1 -> "CORAZON"
8              else -> "SIN_MARCA"
9          }
10     }
11 }
12
```

- Hasta este punto, la aplicación responde con un mensaje en cada celda seleccionada, con el numero de fila y columna en la cual se ubica



Fase 2: Cambiar Estado de la Casilla (1)

(*4)

```
1  fun CambiarEstadoCasilla(b: ImageButton, fila: Int, columna: Int) {
2      if (matrix[fila][columna]!!.deviceStatus.equals("SIN_MARCA")) {
3          if (Turno == 1) {
4              b.setImageResource(R.drawable.cruz)
5              matrix[fila][columna]!!.deviceStatus = "CRUZ"
6              ConteoCruces += 1
7          } else {
8              b.setImageResource(R.drawable.corazon)
9              matrix[fila][columna]!!.deviceStatus = "CORAZON"
10             ConteoCorazones += 1
11         }
12         if (Turno == 1) Turno = 2 else Turno = 1
13         ActualizarEstatusTablero(fila, columna)
14     }
15 }
16
```

(*4)

```
1  fun ActualizarEstatusTablero(fila: Int, columna: Int) {
2      TextViewPizarra.text = "Ultimo Turno: " +
3              matrix[fila][columna]!!.deviceStatus +
4              "\n Conteo Cruces: " + ConteoCruces +
5              "\n Contro Corazones: " + ConteoCorazones
6  }
```

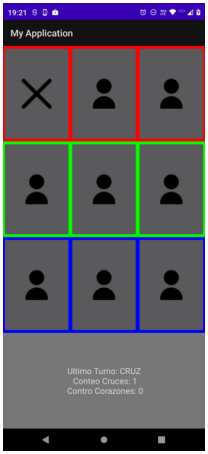
Fase 2: Cambiar Estado de la Casilla

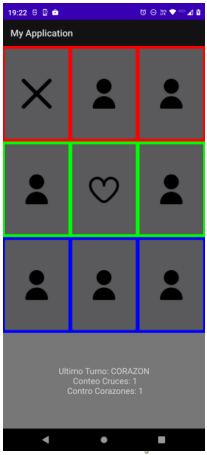
(*1)

```
1  import android.view.View
2  import android.widget.ImageButton
3  import android.widget.TextView
4  import android.widget.Toast
5  import android.widget.Toast
6  import androidx.appcompat.app.AlertDialog
```

(*4) - Reemplazar

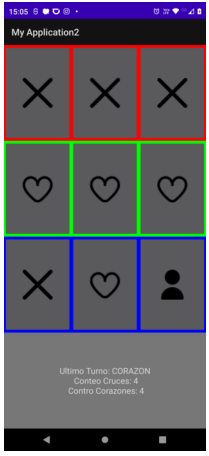
```
1  val btnListener = View.OnClickListener {
2      val (fila, columna) = ObtieneFilaColumna (it)
3      //Toast.makeText(applicationContext,
4      //    "PresionasteBoton ["+fila+","+columna+"]"+
5      //        matrix[fila][columna]!!.deviceStatus,
6      //    Toast.LENGTH_SHORT).show()
7      //TextViewPizarra.text = "Alguna informacion util"
8
9      val b: ImageButton = findViewById(it.id)
10     CambiarEstadoCasilla (b, fila, columna)
11 }
12
```

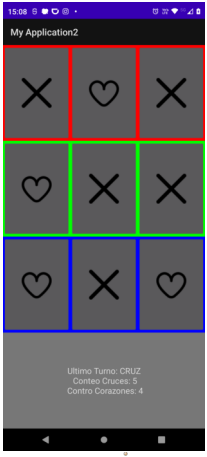




Problemas pendientes de resolver

1. No detecta al ganador
2. Permite continuar el juego aun habiendo ganado uno (o dos)
3. No detecta el empate





Fase 3: Checar Ganador (1)

(*4)

```
fun MostrarAlertDialog(C: String) {
    val builder = AlertDialog.Builder(this)
    builder.setTitle("Juego Finalizado")
    builder.setMessage(C)
    builder.setCancelable(false)
    builder.setPositiveButton("Reiniciar") { dialog, which ->
        Toast.makeText(applicationContext, android.R.string.yes, Toast.LENGTH_SHORT).show()
        ReiniciarControles()
    }
    builder.setNegativeButton("Salir") { dialog, which ->
        Toast.makeText(applicationContext, android.R.string.no, Toast.LENGTH_SHORT).show()
        finish()
    }
    builder.show()
}
```

Fase 3: Checar Ganador (2)

(*4)

```
1  fun ChecarGanadorPorFilas(): Triple<Boolean, Int, String> {
2      // Recorrido por filas
3      var Pivote: String = ""
4      for (i in 0..numRows - 1) {
5          Pivote = matrix[i][0]!!.deviceStatus
6          for (j in 1..numCols - 1) {
7              if (matrix[i][j]!!.deviceStatus.equals(Pivote))
8                  Pivote = matrix[i][j]!!.deviceStatus
9              else
10                 Pivote = "NO"
11         }
12         if (Pivote.equals("CRUZ") || Pivote.equals("CORAZON")) {
13             return Triple(true, i, Pivote)
14         }
15     }
16     return Triple(false, -1, "No");
17 }
```

Fase 3: Checar Ganador (3)

(*4)

```
1  fun ChecarGanadorPorColumnas(): Triple<Boolean, Int, String> {
2      // Recorrido por columnas
3      var Pivote: String = ""
4      for (i in 0..numCols - 1) {
5          Pivote = matrix[0][i]!!.deviceStatus
6          for (j in 1..numRows - 1) {
7              if (matrix[j][i]!!.deviceStatus.equals(Pivote))
8                  Pivote = matrix[j][i]!!.deviceStatus
9              else
10                 Pivote = "No"
11         }
12         if (Pivote.equals("CRUZ") || Pivote.equals("CORAZON")) {
13             return Triple(true, i, Pivote)
14         }
15     }
16     return Triple(false, -1, "NO");
17 }
```

Fase 3: Checar Ganador (4)

(*4)

```
1  fun ChecarGanadorPorDiagonales(): Triple<Boolean, Int, String> {
2      var Pivote: String
3      Pivote = matrix[0][0]!!.deviceStatus
4      for (i in 1..numRows - 1) {
5          if (matrix[i][i]!!.deviceStatus.equals(Pivote))
6              Pivote = matrix[i][i]!!.deviceStatus
7          else
8              Pivote = "NO"
9      }
10     if (Pivote.equals("CRUZ") || Pivote.equals("CORAZON")) {
11         return Triple(true, 0, Pivote)
12     }
13     var Col = 2
14     Pivote = matrix[0][Col]!!.deviceStatus
15     for (i in 1..numRows - 1) {
16         Col -= 1
17         if (matrix[i][Col]!!.deviceStatus.equals(Pivote))
18             Pivote = matrix[i][Col]!!.deviceStatus
19         else
20             Pivote = "NO"
21     }
22     if (Pivote.equals("CRUZ") || Pivote.equals("CORAZON")) {
23         return Triple(true, 1, Pivote)
24     }
25     return Triple(false, -1, "NO");
26 }
```

Fase 3: Checar Ganador (5)

(*4)

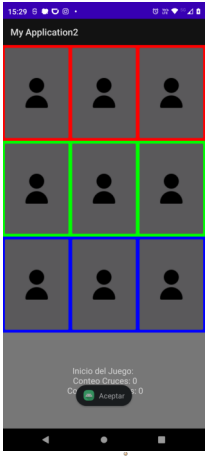
```
1  fun ChecarAlgunGanadorFin(fila: Int, columna: Int): Boolean {
2      if ((ConteoCorazones > 2) || (ConteoCruces > 2)) {          // Checa si hay fila ganadora
3          var (Ganador, Indice, Jugador) = ChecarGanadorPorFilas()
4          if (Ganador) { // Hay una fila ganadora
5              TextViewPizarra.text = "Gano: " + Jugador + " Fila: " + Indice
6              MostrarAlertDialog("Ganador: " + Jugador)
7              return (true)
8          } else { // Checa si hay columna ganadora
9              var (Ganador2, Indice2, Jugador2) = ChecarGanadorPorColumnas()
10             if (Ganador2) { // Hay una columna ganadora
11                 MostrarAlertDialog("Ganador: " + Jugador2)
12                 TextViewPizarra.text = "Gano: " + Jugador2 + " Columna: " + Indice2
13                 return true
14             } else {
15                 var (Ganador3, Indice3, Jugador3) = ChecarGanadorPorDiagonales()
16                 if (Ganador3) { // Ganador por alguna columna
17                     MostrarAlertDialog("Ganador: " + Jugador3)
18                     TextViewPizarra.text = "Gano: " + Jugador3 + " Diagonal: " + Indice3
19                     return true
20                 } else
21                     return (false)
22             }
23         }
24     } else {
25         ActualizarEstatusTablero(fila, columna)
26     }
27     return false
28 }
```

Fase 3: Checar Ganador (6)

(*4) - Actualizar

```
1  val btnListener = View.OnClickListener {
2      val (fila, columna) = ObtieneFilaColumna (it)
3      //Toast.makeText(applicationContext,
4      //     "PresionasteBoton ["+fila+","+columna+"]"+
5      //         matrix[fila][columna]!!.deviceStatus,
6      //     Toast.LENGTH_SHORT).show()
7      //TextViewPizarra.text = "Alguna informacion util"
8
9      val b: ImageButton = findViewById(it.id)
10     CambiarEstadoCasilla (b, fila, columna)
11     ChecarAlgunGanadorFin(fila, columna)
12 }
13
```

¿Qué problema falta por resolverse?



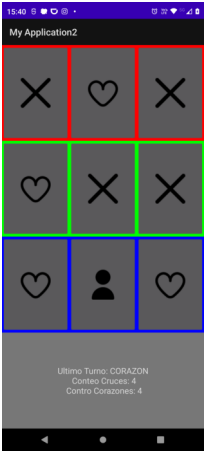
Fase 4: Checar Empate

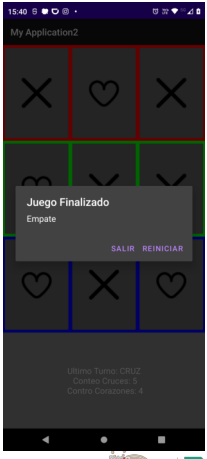
(*4)

```
fun ChecarEmpate() {
    // Numero de Celdas Marcadas con Gato
    if (ConteoCruces + ConteoCorazones == 9) {
        MostrarAlertDialog("Empate")
    }
}
```

(*4) - Actualizar

```
val btnListener = View.OnClickListener {
    val (fila, columna) = ObtieneFilaColumna (it)
    //Toast.makeText(applicationContext,
    //    "PresionasteBoton ["+fila+","+columna+"]"+
    //        matrix[fila][columna]!!.deviceStatus,
    //    Toast.LENGTH_SHORT).show()
    //TextViewPizarra.text = "Alguna informacion util"

    val b: ImageButton = findViewById(it.id)
    CambiarEstadoCasilla (b, fila, columna)
    if (!ChecarAlgunGanadorFin(fila, columna))
        ChecarEmpate()
}
```





Pong: El Nacimiento de la Industria

Creador y Compañía

- Desarrollado para la empresa **Atari** en 1972, convirtiéndose en el **primer videojuego de éxito comercial masivo**.
- El concepto se inspira en el deporte del tenis de mesa (ping pong).

Revolución Doméstica (1975)

- Atari lanzó una versión casera, *Home Pong*, vendida exclusivamente a través de las tiendas Sears durante la temporada navideña de 1975.
- Esto introdujo los videojuegos en el **hogar** y provocó el nacimiento del mercado de consolas domésticas.

Legado Técnico y Cultural

- A pesar de su simplicidad gráfica (dos paletas y una pelota), sentó las bases para mecánicas de juego fundamentales y estableció a los videojuegos como una **forma de entretenimiento viable y lucrativa**.

Aplicación Pong

1. **Configuración de Pantalla Completa**
   - Se asegura una experiencia inmersiva eliminando la barra de título (`FEATURE_NO_TITLE`) y la barra de acción (`supportActionBar.hide()`), y estableciendo el modo de **pantalla completa** (`FLAG_FULLSCREEN`).
2. **Motor de Juego Principal (`GameView`)**
   - La lógica del juego (dibujo, física, bucle de actualización) reside en una vista personalizada (`GameView`), la cual se añade dinámicamente al contenedor (`FrameLayout`) de la actividad.
3. **Comunicación y Actualización de Puntaje**
   - Se establece un **Listener** (`ScoreUpdateListener`) en la `GameView` para recibir los nuevos puntajes.
   - La actualización de los `TextView` de los jugadores se realiza de forma segura en el **Hilo Principal** de Android (`runOnUiThread`) para evitar fallos en la interfaz de usuario.
4. **Manejo del Ciclo de Vida del Juego**
   - La actividad maneja el estado del juego mediante los métodos `onPause()` y `onResume()`, llamando a `gameView.pauseGame()` y `gameView.resumeGame()` respectivamente. Esto es esencial para **conservar recursos** al cambiar de aplicación.

El Corazón del Juego: Clase `GameView`

La clase `GameView` es la superficie de dibujo principal y el motor de lógica de la aplicación.

1. **Gestión de Hilos (Threading)**
   - Contiene un **GameThread** que ejecuta el bucle de juego para **actualizar la lógica** (`update()`) y (`draw()`) en un ciclo constante.
   - Esto asegura que las animaciones se ejecuten fluidamente sin bloquear el Hilo Principal (UI).
2. **Implementación de la Lógica del Juego**
   - Maneja la posición y el movimiento de los objetos (pelotas, paletas, etc.) y la detección de colisiones.
   - Es responsable de incrementar los contadores de puntaje tras eventos clave, como anotar un punto.
3. **Control de Estado y Comunicación**
   - Utiliza la interfaz `ScoreUpdateListener` para notificar a la actividad sobre los cambios de puntaje, separando la lógica del juego de la actualización de la UI.

Fase 1: Dividir la pantalla en contenedores verticales

Archivo **activity_main.xml**

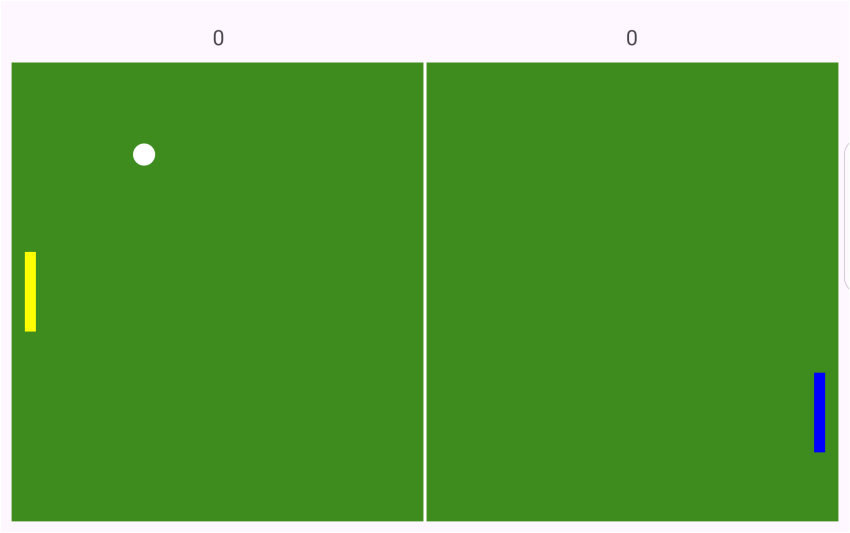
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"  android:layout_height="match_parent"
4      android:orientation="vertical" android:padding="16dp">
5      <LinearLayout  android:layout_width="match_parent"
6          android:layout_weight="1"  android:orientation="horizontal"
7          android:layout_height="0dp" >
8          <TextView
9              android:id="@+id/scorePlayer1"  android:gravity="center"
10             android:layout_weight="1"  android:layout_width="0dp"
11             android:layout_height="match_parent" android:textSize="32sp"
12             android:text="0"  android:padding="8dp"/>
13         <TextView
14             android:id="@+id/scorePlayer2"  android:gravity="center"
15             android:layout_weight="1" android:layout_width="0dp"
16             android:layout_height="match_parent" android:textSize="32sp"
17             android:text="0"  android:padding="8dp"/>
18     </LinearLayout>
19     <FrameLayout
20         android:id="@+id/gameContainer" android:layout_width="match_parent"
21         android:layout_weight="9"  android:layout_height="0dp"
22         android:layout_below="@id/scorePlayer1"/>
23 </LinearLayout>
```

Fase 2: Definir el comportamiento de la aplicación

Archivo **MainActivity.kt**

```
    // Declaración de variables usando 'lateinit' para inicialización no nula en onCreate
    private lateinit var gameView: GameView
    private lateinit var scorePlayer1: TextView
    private lateinit var scorePlayer2: TextView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        supportActionBar?.hide()
        window.setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN)
        setContentView(R.layout.activity_main)
        scorePlayer1 = findViewById(R.id.scorePlayer1)
        scorePlayer2 = findViewById(R.id.scorePlayer2)
        val gameContainer: FrameLayout = findViewById(R.id.gameContainer)
        gameView = GameView(this)
        gameContainer.addView(gameView)
        gameView.setScoreUpdateListener(object : GameView.ScoreUpdateListener {
            override fun onScoreUpdate(player1Score: Int, player2Score: Int) {
                runOnUiThread {
                    scorePlayer1.text = player1Score.toString() // Convierte Int a String
                    scorePlayer2.text = player2Score.toString() // Convierte Int a String
                }
            }
        })

    }
..
```

Comportamiento de la aplicación



Contenido

Conclusiones

- En este taller describimos conceptos fundamentales de programación, computadoras y sistemas operativos
- Configuramos un teléfono inteligente en modo de depuración
- Creaamos una aplicación vacía en Android Studio, instalar y ejecutar tal aplicación en el teléfono inteligente
- Comprendimos los archivos principales de la aplicación
- Modificarmos la interfaz para entender el concepto de contenedor
- Modificarmos la interfaz para implementar el juego de TIC-TAC-TOE
- Modificarmos el comportamiento de la aplicación agregando variables, métodos y clases
- Repetimos el ejercicio para el juego del Pong.

Agradecimiento

Presentación y código fuente disponible en el siguiente enlace:
https://github.com/mnunom-upv/Curso_Desarollo_Aplicaciones_Moviles_2025
Comentarios o Dudas: mnunom@upv.edu.mx