TRABAJO FIN DE GRADO

Aplicación Android de un codificador de dígitos cistercienses en Python

Autores:

Luis Miguel Durán Otero

Juan Pablo Pila Naranjo

Tutor:

Alex Tolón Gimeno

Índice

1.- Análisis del problema planteado

El objetivo de este **Trabajo de Fin de Grado** es la creación de una **aplicación para Android**, siguiendo métodos poco convencionales como usar **Python** y una herramienta como lo es **Kivy/KivyMD**.

El problema de este trabajo presenta que se debe crear un algoritmo principal que será el motor de toda la app, el cual codifica los números arábigos en código de símbolos cisterciense.

La app se dividirá en tres principales partes gráficas, una primera pestaña, donde se presentará un conversor donde directamente introduciremos un número arábigo y seguidamente, un botón activará la conversión para poder visualizar en pantalla su homólogo en cisterciense.

En segundo lugar, una pestaña que muestra un contador progresivo al cual le mandaremos el inicio de la cuenta dentro de un **Textfield**, tendremos tres botones que controlarán esta actividad, tanto como iniciar la animación, pausarla y reiniciarla.

La última parte contiene una pestaña que incluye un pequeño juego, este juego nos dibuja en pantalla un número aleatorio entre el 1 y el 9999 en cisterciense, nuestro rol es intentar acertar cuál es su hermano en arábigo.

2.- Esquema cronológico de investigación seguido y justificación de las elecciones realizadas

-Toma de contacto

Tras haber realizado una profunda lectura a las directrices y enunciado del trabajo propuestos por nuestro tutor de prácticas, lo primero que pensamos es en cómo hacer una aplicación para un móvil Android utilizando el lenguaje de programación Python cuando lo normal para el desarrollo de aplicaciones Android es usar Java, Kotlin, Flutter…

-Cómo hacer una app con Python

En las ayudas que el profesor nos había proporcionado, se encontraban varios enlaces en donde podíamos encontrar alguna información y diferentes tutoriales para poder empezar a desarrollar una primera idea del proyecto.

Uno de los tutoriales del profesor era sobre una extensión o paquete de Python llamada Kivi, al parecer con esto podremos crear ventanas e incluso hemos encontrado como hacer ventanas con pestañas (tabs).

A partir de ese tutorial comenzamos a buscar por nuestra cuenta más información sobre Kivy para posteriormente comenzar a desarrollar la aplicación sobre dicho lenguaje.

¿Que es lo primero que tengo que instalar?

Lo primero fue instalar el entorno de Python con PyCharm, seguido del paquete de Kivy y experimentar un poco con él, ya que al parecer podemos separar la parte lógica de la aplicación de la parte gráfica con un archivo especial de Kivy llamado con la extensión **".kv"**. Además de poder hacerlo de forma externa se puede llamar de forma interna en el propio archivo .py.

Lo siguiente es saber cómo dibujar líneas dentro de una ventana de Python e intentar hacer los patrones que tienen los números cistercienses.

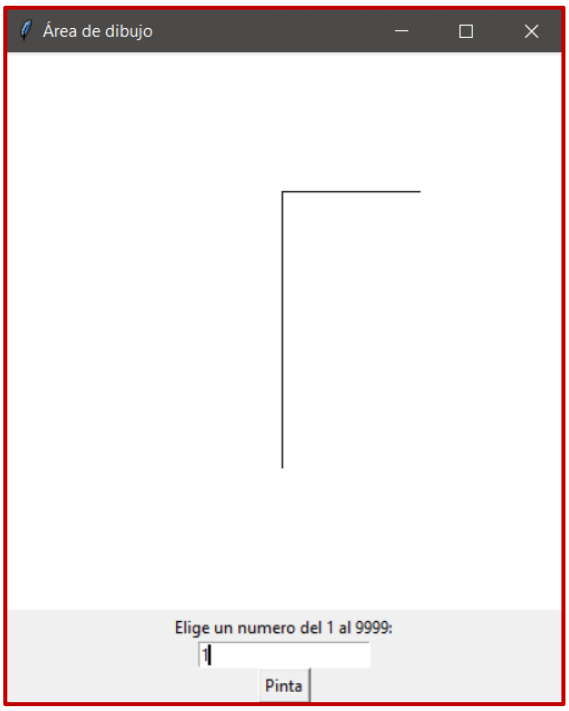
Otro paquete de Python nos permite mostrar funciones. Pero está hecho específicamente para gráficas matemáticas y lo que nosotros queremos es pintar literalmente en una pantalla.

Con tkinter podremos crear un canvas y dibujar lo que queramos sobre ese canvas, pero para ello, primero tendremos que instalarlo.

Primeros pasos dibujando

Una vez sepamos dibujar, aunque sea una línea en el canvas, el siguiente paso es intentar encontrar un sistema para recrear los números simples en cisterciense desmembrando su estructura como usualmente se representan los de siete segmentos para una calculadora o un reloj Casio.

Vamos a intentar dibujar el número 1 en cisterciense.



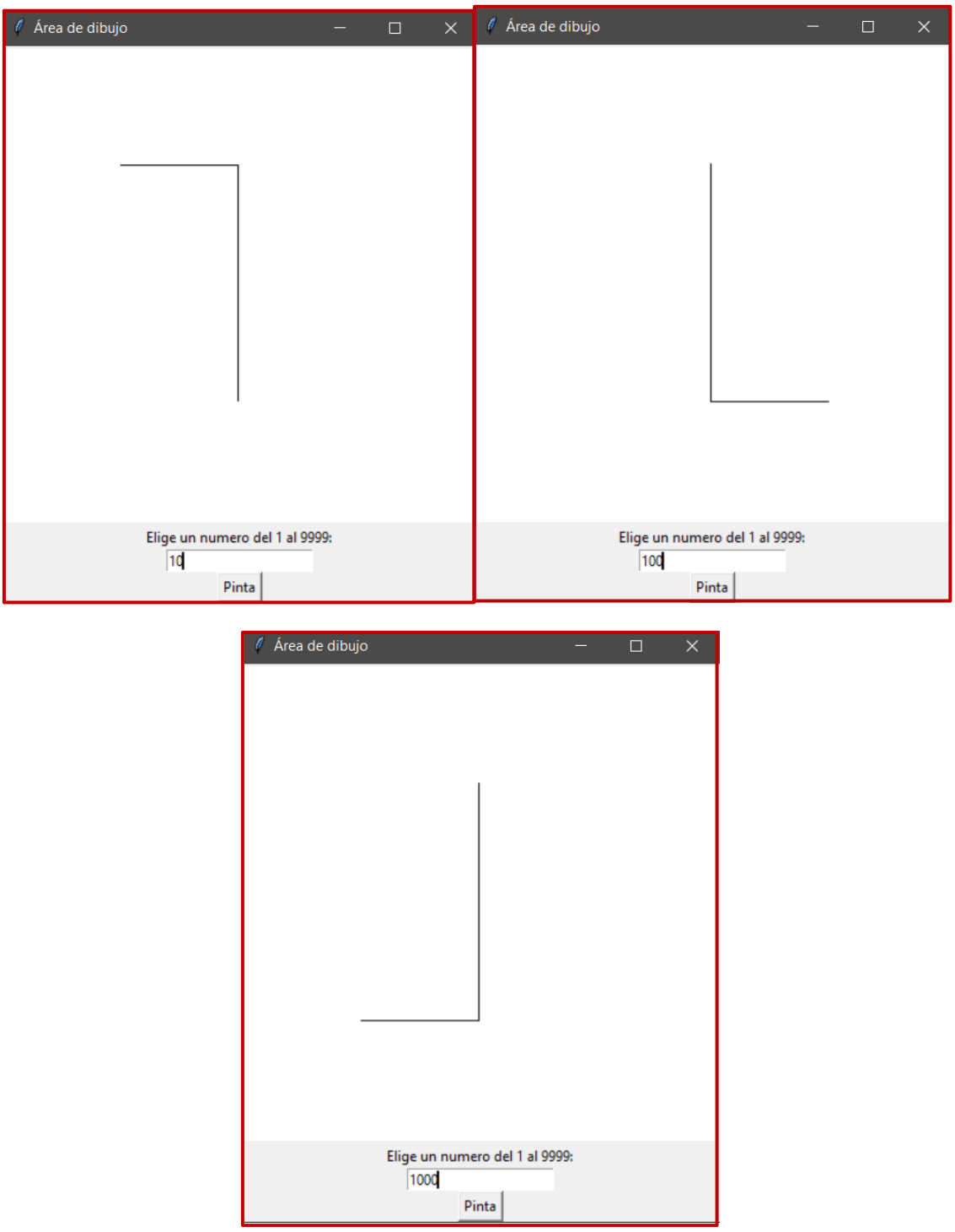
Primera confección de los símbolos

Viendo detenidamente los números cistercienses, hemos observado que en realidad son seis segmentos los que forman los números pero su múltiplo de diez tiene un atributo de rotación.

El 10 es el número 1 rotado hacia la izquierda como un espejo.

El 100 es el número 1 rotado hacia abajo.

El número 1000 es el número 10 rotado hacia abajo.



Creada la primera versión que dibuja

Una vez realizada la primera versión del dibujo de estos símbolos es la hora de empezar con Kivy/KivyMD para la realización de la interfaz gráfica y comprender su funcionamiento.

Creando el primer intento de algoritmo

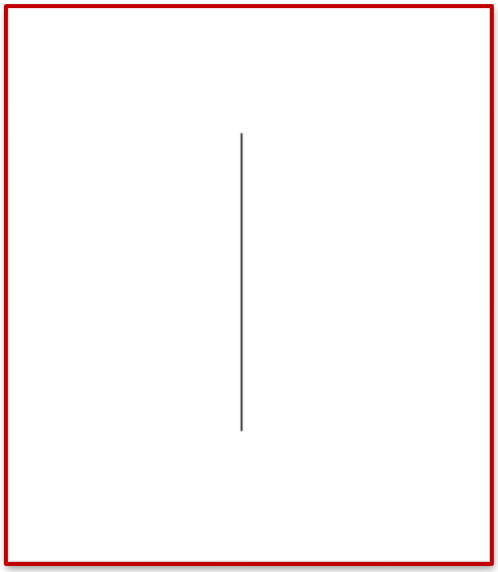
Vamos a ir dibujando todos los números del 1 al 9 y guardando las variables de las líneas que vamos dibujando y después se lo almacenaremos a una tabla donde ya deberá reconocer los números.

Primero dibujaremos los números simples sin atributos de rotación.

Por ejemplo, el segmento 1 será:

seg1 = (200, 100, 200, 300)

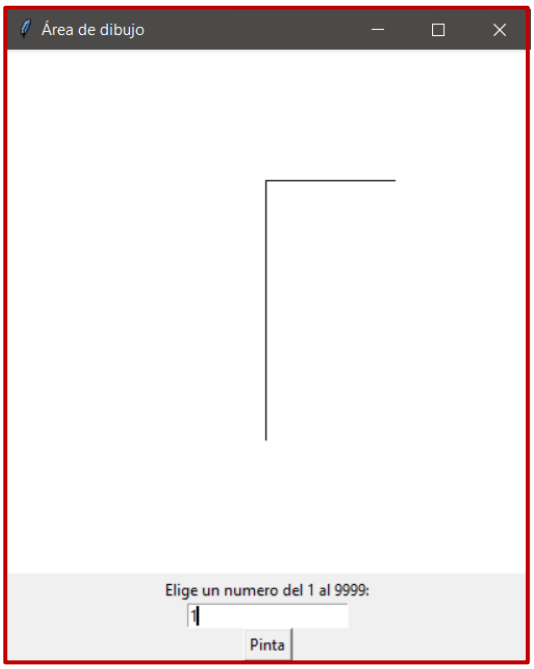
Y se corresponde con:



El número uno se corresponde con dos segmentos por lo que nos limitamos en este intento a solo dibujar el número 1 con este trozo de código:

```
canvas.create_line(200, 100, 200, 300)
canvas.create_line(200, 100, 300, 100)
```

Observamos lo siguiente al pintarlo:



Vamos a crear todos los segmentos que hay, e intentar recrear todos los números del 1 al 9.

- Realización final del conversor con tkinter (converter.py)

Ya tenemos la primera versión del algoritmo que, junto con tkinter, está capacitado para manejarse con un pequeño Textfield y un botón, podemos representar los primeros símbolos en una pantalla de escritorio desde el 1 al 9999.

- Toma de contacto con Kivy

¿Qué es Kivy?

Kivi es un framework para Python que nos permite desarrollar aplicaciones de manera rápida. Es de código abierto y multiplataforma que nos permite desarrollar aplicaciones con funcionalidades complejas, interfaz de usuarios amigable con propiedades multitáctiles, todo esto desde una herramienta intuitiva, orientada a generar prototipos de manera rápida y con diseños eficiente que ayuden a tener códigos reutilizables y de fácil despliegue.

Ha sido desarrollado utilizando Python y Cython se basa en **OpenGL ES 2** y soporta una gran cantidad de dispositivos de entradas, de igual manera, la herramienta está equipada de una extensa biblioteca de widgets que ayudan a añadir múltiples funcionalidades. Nos permite generar código base que puede ser utilizado en aplicaciones orientadas a Linux, Windows, Android e iOS.

Lenguaje KV

El lenguaje kv es el usado para dar instrucciones a Kivy de cómo va a ser la estructura gráfica de nuestra app, mediante la creación de widgets podremos encajar nuestra idea dentro de una pantalla un poco más parecida a lo que en realidad será en un teléfono Android.

¿Se usa Kivy?

Para el desarrollo de aplicaciones Android, Kivy es una excelente opción ya que nos permite crear aplicaciones móviles usando tus habilidades del lenguaje de programación Python sin tener que aprender otro lenguaje de plataforma.

Como todo lenguaje de programación tiene sus ventajas y desventajas, en este caso las ventajas de Kivy son:

-Basado en Python, que es extremadamente poderoso dada su naturaleza rica en bibliotecas.

-Escriba el código una vez y utilícelo en todos los dispositivos.

-Widgets fáciles de usar construidos con soporte multitáctil.

-Funciona mejor que las alternativas multiplataforma de HTML5.

Por otro lado encontramos las desventajas que son:

-Interfaz de usuario de aspecto no nativo.

-Tamaño de paquete más grande(porque es necesario incluir el intérprete de Python).

-Falta de apoyo de la comunidad(Kivy Community no es particularmente grande).

-Falta de buenos ejemplos y documentación.

-Alternativas mejores y más ricas en la comunidad disponibles si solo se enfoca en dispositivos móviles multiplataforma, es decir, React Native.

¿Kivy es mejor que Android Studio?

Kivy se basa en Python, mientras que Android Studio es principalmente Java, con compatibilidad reciente con C++.

Para un principiante, sería mejor ir con Kivy, ya que Python es relativamente más fácil que Java y es más fácil de entender y construir.

Además, a nivel de principiante, el soporte multiplataforma es algo de los que preocuparse al principio.

- Primeros intentos con Kivy y KivyMD

Los primeros intentos de desarrollo con Kivy en general están basados en la comprensión de este lenguaje ya que es algo nuevo para este equipo. Para ello hemos realizado una profunda lectura y comprensión de la documentación probando algunos de sus Widgets.

Se intenta sacar una primera ventana en el escritorio y probar botones y algunos Widgets que aparecen en la documentación oficial para así poder ir viendo la capacidad y posibilidades gráficas de esta nueva herramienta.

Estructura de la app

En una primera idea decidimos hacerlo con una estructura por pestañas, llamadas Tabs, anidadas dentro de un MDToolBox que a su vez pertenece a un Layout que le indiquemos a Kivy.

Gracias al lenguaje Kivy podremos acceder a los elementos de su código, escrito únicamente como cadenas de texto y siendo llamadas para ser leídas de manera parecida a como CSS funcionaba con HTML.

Creando las llamadas Clases podremos acceder a ciertos elementos o Widgets de nuestra aplicación para poder ser modificados a nuestro antojo.

La primera pestaña

La pestaña del conversor y la manera en la que debía dibujar los símbolos fue un poco más costosa de lo imaginado, igualmente su resultado ha sido relativamente exitoso, ya que ahora queda la primera prueba en la que veremos si esto funciona en formato apk.

La segunda pestaña

Una vez realizada la primera pestaña y habernos liberado del primer bache que nos presentaba el dibujar los símbolos, solo nos queda averiguar cómo fabricar una especie de reloj que haga avanzar los números a la par que se dibuja su hermano en cisterciense.

Este reloj (o segundero) será representado por un label y permitirá una cuenta progresiva desde 0 hasta el 9999.

El objetivo es poder elegir el inicio y el fin de esta cuenta progresiva, y a su vez poder controlarla por los botones “START”, “STOP” y “REINICIAR”, cuya función es la de su mismo nombre.

El resultado de momento es exitoso en cuanto a la cuenta progresiva sin límite desde 0.

Siguiente punto, marcar el inicio y el fin.

La tercera pestaña

Esta tercera pestaña remata la aplicación añadiendo un pequeño juego en el que se nos representa un símbolo aleatorio que se refiere a un número entre el 0 y el 9999, nuestro papel es intentar acertar el número correcto.

El diseño es simple y tendremos dos botones los cuales nos permiten pedir otro símbolo y corregir nuestra opción.

El resultado ha sido satisfactorio.

Primer intento de la apk

Siguiendo algunos tutoriales recomendados por el profesor y facilitador del enunciado del proyecto, llegamos a un punto muerto el cual

mediante cualquier método que se mencionaba en la web no era posible extraer una apk que no dejase de funcionar en el momento de ejecutarla en un móvil Android.

Después de un par de días de investigación se llegó a una solución bastante factible la cual permite transformar nuestra app en apk desde la página de **Google Colab**.

Colab mediante unos comandos de Linux nos hacía casi todo el trabajo, solo había que meter nuestro **main.py** en una carpeta de esta misma página y ejecutar una serie de comandos, generar un archivo de configuración llamado **buildozer.spec**, el cual contenía información para poder compilar nuestra app, tal como su **icono**, su **nombre** e incluso las **extensiones** o **plugins** que hemos usado dentro de ella.

La solución contaba que dentro de este apartado de las extensiones y plugins, debíamos incluir, aparte de Kivy (más la concreción de su versión usada) también debíamos incluir Kivymd y la versión que se había utilizado y esto debería valer para que el buildozer empiece a hacer su magia y transformar nuestra apk.

Cosa que no fue suficiente ya que nos faltaba una pequeña pieza, que era incluir la extensión de pillow, que al parecer permite la renderización de imágenes de estas dos extensiones que predominan mayoritariamente en nuestro código y a su vez permite la compatibilidad con los smartphones.

Después de esto, llegamos al primer prototipo de nuestra app, que después de unos ajustes gráficos y redimensionar la pantalla pudimos gozar de ella por primera vez en una pantalla de un dispositivo Android.

Errores encontrados (el canvas de Kivy)

El principal problema era enlazar el evento del botón que disparaba la acción de pintar en pantalla.

Instalación de entorno Ubuntu (algunos comandos de shell)

El primer intento se basaba en un tutorial facilitado por el tutor escolar, donde la protagonista de este tutorial instalaba un entorno Ubuntu como primer paso esencial para nuestra conversión de **.py a .apk.**

Adb no reconoce ningún dispositivo.(Solución)

Otro de los pasos era instalar el Android Debug Bridge, cosa que no fue posible ya que no reconoce nuestro dispositivo en modo depuración USB para este tipo de operaciones.

Se usó Google colab para sacar la apk. (Solución)

Rastreando un poco internet se encontró otro método que mencionaba la herramienta online de Google Colab.

En un foro, cuyo enlace está en el apartado de recursos, pudimos encontrar unas instrucciones sobre este método que supuestamente introduciendo unos comandos de Linux y un par de ajustes, nuestra app saldría del horno en formato .apk.

Esto no funcionó debido a que nos faltaba incluir en los ajustes la herramienta Pillow.

3.- Recursos consultados correctamente reseñados y priorizados

He aquí los enlaces visitados en el transcurso del desarrollo de la aplicación:

-Simple Python App with Kivy - Step by Step GUI Tutorial: https://www.youtube.com/watch?v=YDp73WjNISc

-Binary to Decimal - Convert Numbers and Fractions Like a Pro!:

https://www.youtube.com/watch?v=4IFq7_s9NEA&t=3s

-Un primer vistazo a la codificación cisterciense: https://www.costadelsolfm.org/wp-content/uploads/2020/11/Numeros-cistercienses-4.jpg

-Cómo se conforman los números:

https://ichef.bbci.co.uk/news/640/cpsprodpb/0D00/production/_115482330_numeros_cister_ejemplos-nc.png

-Kivy y Android Studio:

https://ulmerstudios.com/tips-and-recommendations/can-i-use-kivy-with-android-studio/

-Una demo muy ilustrativa: https://youtu.be/ah3JeHAfM0M

-Otra demo con enfoque de GUI: https://youtu.be/gimdyxWwVYQ

-Python for Mobile App Development - What's the Catch?: https://www.mobileappdaily.com/python-for-mobile-apps-development

-Build a Mobile Application With the Kivy Python Framework: https://realpython.com/mobile-app-kivy-python/

-Hacer un contador con botones : https://www.geeksforgeeks.org/python-create-a-stopwatch-using-clock-object-in-kivy-using-kv-file/

-Uso del Toolbar:

https://kivymd.readthedocs.io/en/latest/components/toolbar/index.html#bottom

-Otro tutorial usado: https://avionmission.github.io/blog/convert-py-to-apk-using-python-and-buildozer/

-Página del tutorial con los comandos: https://colab.research.google.com/gist/kaustubhgupta/0d06ea84760f65888a2488bac9922c25/kivyapp-to-apk.ipynb#scrollTo=Gjp7v6bTnd5V

-Como convertir de .py a .apk : https://kivy.org/doc/stable/guide/packaging-android.html

4.- Presentación del conjunto de códigos que conforman el proyecto que recoge la solución

Estructura de la app:

-Widget Padre

```
<MainWidget@MDBoxLayout>:
    orientation: "vertical"
    MDToolbar:
        title: "La orden del Císter"
        md_bg_color: app.theme_cls.primary_dark
    MDTabs:
        text_color_normal: 1, 1, 1, 1
                text_color_active: 1, 209/255, 60/255, 1
                tab_indicator_anim: 'True'
                anim_threshold: 0.5
```

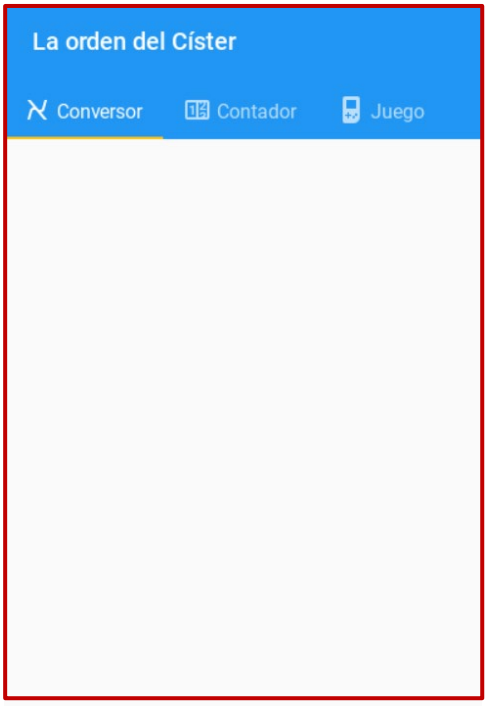
La estructura principal de conforma de un **MainWidget**, o en su defecto un **Widget padre**, el cual recoge un **MDToolbar** y un **MDTabs** que nos permite insertar las pestañas **(Tabs).**

-Estructura en Tabs (Parte gráfica)

```
MDTabs:
        text_color_normal: 1, 1, 1, 1
                text_color_active: 1, 209/255, 60/255, 1
                tab_indicator_anim: 'True'
                anim_threshold: 0.5

        Tab:
            id: tabConversor
            title: "Conversor"
            icon: "abjad-hebrew"

        Tab:
            id: tabContador
            title: "Contador"
            icon: "counter"

        Tab:
            id: tabJuego
            title: "Juego"
            icon: "nintendo-game-boy"
            text_icon_color: .2, .2, .2, 1
```

Esta es la estructura que seguirá la app, tenemos tres **tabs** que representan a cada **pestaña**, con sus propiedades id, title (título), e icon (icono).

Este es el resultado gráfico de nuestro código Kivy:



-Tab Conversor

```
Tab:
    id: tabConversor
    title: "Conversor"
    icon: "abjad-hebrew"
    text_icon_color: .2, .2, .2, 1

    MDBoxLayout:
        orientation: "vertical"
        spacing: 370

        MDTextField:
            id: txt1

            hint_text: "Introduce un número entre el 1 y el 9999"
            pos_hint: {"center_x": .99, "center_y": .5}
            max_text_length: 4
            hint_text_color_normal: 0, 0, 0, 1
            font_size:"16dp"
        MDBoxLayout:
            orientation: "horizontal"
            id: lobtn
```

Se define un Textfield donde introduciremos el número válido para su consiguiente conversión.

Resultado del código:

-Tab Contador

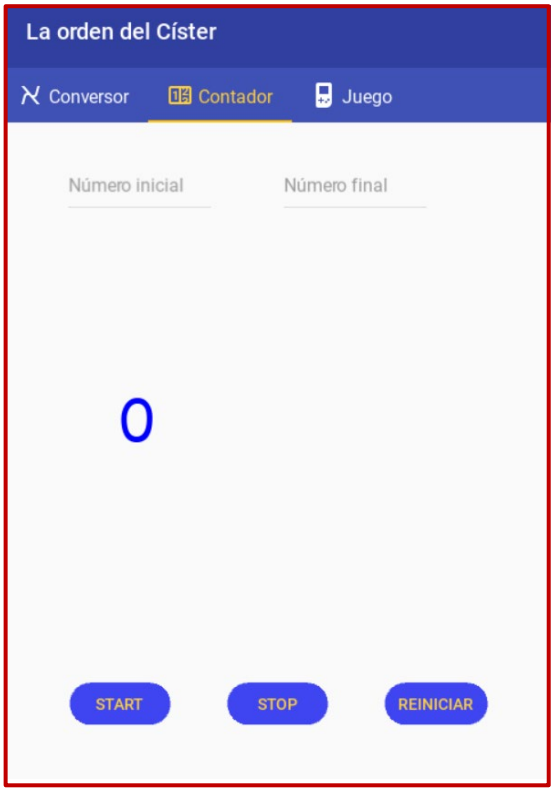
```
Tab:
    id: tabContador
    title: "Contador"
    icon: "counter"
```

* Esta pestaña está dividida en dos partes fundamentales, debido a la complejidad del lenguaje para nuestro equipo, la primera es la ya mostrada que es prácticamente el padre o contenedor del widget que se muestra a continuación:

```
<RelojWidget>:
    FloatLayout:
        orientation: 'vertical'

    FloatLayout:
        orientation: 'vertical'
        Label:
            id: lblArabigo
            text: str(round(root.number))
            text_size: self.size
            pos_hint: {"center_x": .4, "center_y": 1}
            color: (0, 0, 1, 1)
            font_size: 50
            font_name: "Comic"
        BoxLayout:
            orientation: 'horizontal'
            spacing: 50
            pos_hint: {"center_x": 0, "center_y": .6}
            MDFillRoundFlatButton:
                text: 'START'
                on_press: root.start()
                text_color: (1, 209 / 255, 60 / 255, 1)  # Azul
                line_color:(1, 209 / 255, 60 / 255, 1) # Amarillo
                md_bg_color:(56/255, 62/255, 241/255, 0.81) # Azul
                font_style: 'Button'

            MDFillRoundFlatButton:
                text: 'STOP'
                on_press: root.stop()
                text_color: (1, 209 / 255, 60 / 255, 1)  # Azul
                line_color:(1, 209 / 255, 60 / 255, 1) # Amarillo
                md_bg_color:(56/255, 62/255, 241/255, 0.81) # Azul
                font_style: 'Button'

            MDFillRoundFlatButton:
                text: 'REINICIAR'
                on_press: root.number = 0
                text_color: (1, 209 / 255, 60 / 255, 1)  # Azul
                line_color:(1, 209 / 255, 60 / 255, 1) # Amarillo
                md_bg_color:(56/255, 62/255, 241/255, 0.81) # Azul
                font_style: 'Button'
```

Se encuentran definidos en este código tres botones: **START**, **STOP** Y **REINICIAR**, para manejar la animación del contador en paralelo.

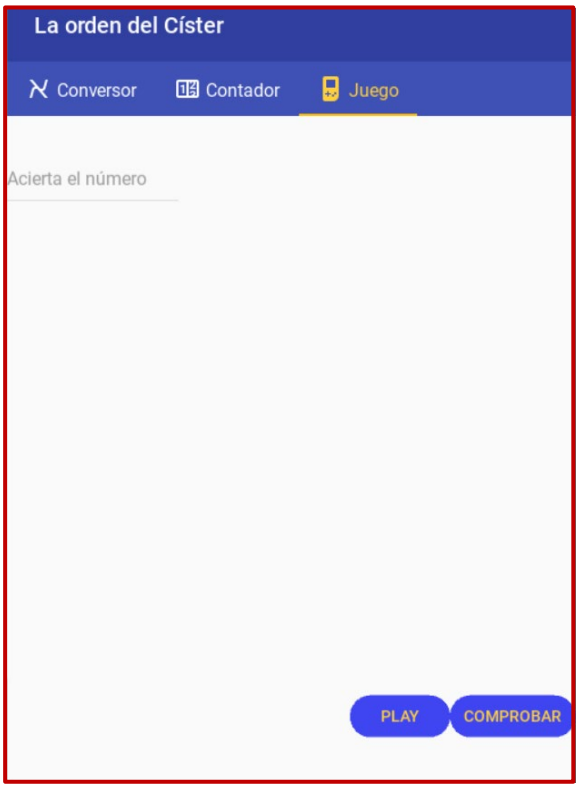
-Resultado del código:



-Tab Juego

```
Tab:
    title: "Juego"
    id: tabJuego
    icon: "nintendo-game-boy"
    text_icon_color: .2, .2, .2, 1

    MDTextField:
        id: respuesta
        pos_hint: {"center_x": -2, "center_y": .9}
        max_text_length: 4
        hint_text_color_normal: 0, 0, 0, 1
        font_size:"16dp"
        hint_text: "Acierta el número"
```

-Resultado del código:



- Parte lógica

```
# width = 500
# height = 700
# dimensiones del movil
width = 1080
height = 2250

Window.size = (width, height)

altura = height / 6
```

En esta primera captura de código declaramos las dimensiones de la ventana de la app empezando por su tamaño para escritorio (500x700), el tamaño ideal para smartphone de 1080x2250 y hemos definido una variable “altura” para ajustar la posición del dibujo.

```
x200 = width / 2
x300 = width / 1.34
y100 = height / 6 + altura
y150 = height / 4 + altura
y300 = height / 2 + altura

seg1 = (x200, y100, x200, y300)
seg2 = (x200, y100, x300, y100)
seg3 = (x200, y150, x300, y150)
seg4 = (x200, y100, x300, y150)
seg5 = (x200, y150, x300, y100)
seg6 = (x300, y100, x300, y150)
```

-Definimos las coordenadas siguiendo el mismo patrón que cuando hicimos la versión de tkinter, pero redimensionando las proporciones al gusto y la necesidad de la pantalla donde se representará.

```
# Registramos todos los números
num1 = [1, 1, 0, 0, 0, 0]
num2 = [1, 0, 1, 0, 0, 0]
num3 = [1, 0, 0, 1, 0, 0]
num4 = [1, 0, 0, 0, 1, 0]
num5 = [1, 1, 0, 0, 1, 0]
num6 = [1, 0, 0, 0, 0, 1]
num7 = [1, 1, 0, 0, 0, 1]
num8 = [1, 0, 1, 0, 0, 1]
num9 = [1, 1, 1, 0, 0, 1]
```

-Aquí hemos declarado los números según el segmento en específico que se va a pintar, por ejemplo, el número uno solo pintara el segmento 1 y el 2; el número dos, el segmento 1 y el 3, y así con todos los números emulando el cómo se declaran para ser representados con el patrón de 7 segmentos.

```
numeros = (num1, num2, num3, num4, num5, num6, num7, num8, num9)
segmentos = (seg1, seg2, seg3, seg4, seg5, seg6)
```

-Iniciamos dos listas, una para los números del 1 al 9 y otra para los segmentos, se irán llamando según la app lo requiera.

```
class Aplicacion(MDApp):
    painter = CanvasWidget()
    painter2 = CanvasWidget2()
    painter3 = CanvasWidget3()
    eligeCanvas = 0
    numeroJuego = 0
```

-Esta es la cabecera del método principal donde se describirá el esqueleto de la aplicación, esta es la clase padre "Aplicacion", descrita como MDApp, necesario para su ejecución como aplicación de KivyMD.
-Los painter serán los canvas que nos ayudarán a pintar en cada pestaña de la aplicación, **eligeCanvas** nos ayudara a que la lógica de la app sepa en cuál de las pestañas dibujar, su valor oscila entre el 1, 2,y 3 según donde ejecutemos la acción de pintar.

-Por último, se encuentra declarado el numerojuego, que no es más que el valor inicial de un número que después será aleatorio entre el 1 y el 9999 para la acción del juego.

```
def build(self):
    self.theme_cls.primary_palette = "Indigo"

    parent = Factory.MainWidget()

    # desde aqui se llama a la propiedad id de los widgets para declarar los tabs
    tab1 = parent.ids["tabConversor"]
    tab2 = parent.ids["tabContador"]
    tab3 = parent.ids["tabJuego"]
```

-Esta es la función build de la clase principal, donde definimos el tema de color para la App, que será "Indigo", después el Widget contenedor de los demás, "parent" y por último definimos de manera sólida en Python los tabs de la app.

```
# desde aqui podemos controlar el estilo de los botones y sus propiedades
pintabtn = MDFillRoundFlatButton(text='Pinta',
                                 text_color=(1, 209 / 255, 60 / 255, 1),  # Azul
                                 line_color=(1, 209 / 255, 60 / 255, 1),  # Amarillo
                                 md_bg_color=(56 / 255, 62 / 255, 241 / 255, 0.81),  # Azul
                                 font_style='Button',
                                 pos_hint={"center_x": .10, "center_y": .1})
```

-Aquí se encuentran definidas las propiedades del botón que permite el pintado de símbolos en la primera pestaña, así como el texto, color de texto, color de fondo, posición, etc.

```
playbtn = MDFillRoundFlatButton(text='Play',
                                text_color=(1, 209 / 255, 60 / 255, 1),  # Azul
                                line_color=(1, 209 / 255, 60 / 255, 1),  # Amarillo
                                md_bg_color=(56 / 255, 62 / 255, 241 / 255, 0.81),  # Azul
                                font_style='Button',
                                pos_hint={"center_x": .10, "center_y": .1})

checkbtn = MDFillRoundFlatButton(text='Comprobar',
                                 text_color=(1, 209 / 255, 60 / 255, 1),  # Azul
                                 line_color=(1, 209 / 255, 60 / 255, 1),  # Amarillo
                                 md_bg_color=(56 / 255, 62 / 255, 241 / 255, 0.81),  # Azul
                                 font_style='Button',
                                 pos_hint={"center_x": .10, "center_y": .1})
```

-De la misma manera para los botones de la pestaña del juego, uno para empezar a jugar y pintar un número cualquiera "**playbtn**" y otro para comprobar nuestra respuesta, "**checkbtn**"

```
# desde aqui asignamos el evento al botón
pintabtn.bind(on_release=self.pinta)
playbtn.bind(on_release=self.numrandom)
checkbtn.bind(on_release=self.compruebaRespuesta)
```

-Seguidamente asignamos los eventos correspondientes a los botones.

```
# aqui se añade el boton al tab (widget padre)
tab1.add_widget(pintabtn)

# añadimos el canvas al tab1
tab1.add_widget(self.painter)
```

-Añadimos el botón de pintar y el widget canvas al **tab1.**

```
tab3.add_widget(self.painter3)
tab3.add_widget(playbtn)
tab3.add_widget(checkbtn)
```

-Igualmente para los widgets que se alojan en el **tab3**, la pestaña del juego.

```
# le ponemos nombre al Widget reloj
self.reloj = RelojWidget()

tab2.add_widget(self.reloj)

tab2.add_widget(self.painter2)
```

-Aquí vamos a crear el widget "reloj" que será el que se encargará de la animación de la pestaña del contador.

-Se agrega al **tab2**, a la par que lo hacemos con el canvas correspondiente.

```
return parent
```

-Este es el final de la clase principal donde se devuelve el "**parent**" o widget padre, así como lo que es la ejecución de la aplicación.

```
# metodo que hace posible que se dibuje en el tab1
    def pinta(self, obj):
        self.eligeCanvas = 1
        self.painter.canvas.clear()
        numero = self.root.ids.txt1.text
        self.calculaCifras(numero)
```

-Pasamos a la lógica de pintado y este es el método encargado del canvas del tab1, se elige el canvas 1, limpiamos el canvas para volver a dibujar luego en un "folio en blanco", extraemos el número del textfield y se lo enviamos a "**calculaCifras**" para su posterior procesado.

```
# metodo que hace posible que se dibuje en el tab2
def pinta2(self, numerotab2):
    self.eligeCanvas = 2
    self.painter2.canvas.clear()
    self.calculaCifras(numerotab2)
```

-El método "**pinta2**" es prácticamente como el anterior pero para el **tab2.**

```
# metodo que hace posible que se dibuje en el tab3 y genera un numero aleatorio para el juego
def numrandom(self, obj):
    self.numeroJuego = random.randint(1, 9999)
    self.eligeCanvas = 3
    self.painter3.canvas.clear()
    print(self.numeroJuego)
    self.calculaCifras(self.numeroJuego)
```

-Aquí seleccionamos un número aleatorio, ya no se recoge de ningún textfield, y se le pasa a **calculaCifras**.

```
# con este metodo podemos comprobar la respuesta del jugador
def compruebaRespuesta(self, obj):
    resp = int(self.root.ids.respuesta.text)
    numeroCorrecto = self.numeroJuego

    if resp == numeroCorrecto:
        toast("Acierto!!")
    else:
        toast("Fallaste!! - El número correcto era:" + str(numeroCorrecto))
```

-Desde aquí comprobamos nuestra respuesta con el número random y el dibujo que se debe mostrar en pantalla, tanto si fallamos como si acertamos saltará una notificación toast.

```
# diferenciamos cuantas cifras tiene el número y se envia para un sitio u otro
def calculaCifras(self, numero):
    cifras = list(str(numero))
    if 0 < len(cifras) <= 4:
        if len(cifras) == 1:
            self.calcula1(cifras)
        elif len(cifras) == 2:
            self.cacula10(cifras)
        elif len(cifras) == 3:
            self.calcula100(cifras)
        elif len(cifras) == 4:
            self.calcula1000(cifras)
```

-Este método se encarga de saber cuántas cifras tiene el número que se le envía y se lo lleva a su método correspondiente tanto si es un número en unidades, decenas, centenas y millares.

```
# tratamiento de las unidades
def calcula1(self, cifras):
    numero1 = cifras[0]
    self.pintaNumero1Kivy(numero1)

# tratamiento de las decenas
def cacula10(self, cifras):
    numero2 = cifras[0]
    numero1 = cifras[1]
    self.pintaNumero2Kivy(numero2)
    self.pintaNumero1Kivy(numero1)

# tratamiento de las centenas
def calcula100(self, cifras):
    numero3 = cifras[0]
    numero2 = cifras[1]
    numero1 = cifras[2]
    self.pintaNumero3Kivy(numero3)
    self.pintaNumero2Kivy(numero2)
    self.pintaNumero1Kivy(numero1)

# tratamiento dde los millares
def calcula1000(self, cifras):
    numero4 = cifras[0]
    numero3 = cifras[1]
    numero2 = cifras[2]
    numero1 = cifras[3]
    self.pintaNumero4Kivy(numero4)
    self.pintaNumero3Kivy(numero3)
    self.pintaNumero2Kivy(numero2)
    self.pintaNumero1Kivy(numero1)
```

-En cada uno de estos métodos se desmiembra el número que le corresponde y envía cada cifra a un método encargado de su representación.

```
# pintor de las unidades
def pintaNumero1Kivy(self, numero):
    cont1 = 0
    numero3 = int(numero)
    if numero3 == 0:
        print("nothing")
    else:
        num = numeros[(numero3 - 1)]
        for n in num:
            if n == 1:
                if cont1 > 0:
                    segmod2 = list(segmentos[cont1])
                    if segmod2[1] == y100:
                        segmod2[1] = y300
                    elif segmod2[1] == y150:
                        segmod2[1] = height / 2.4 + altura
                    if segmod2[3] == y100:
                        segmod2[3] = y300
                    elif segmod2[3] == y150:
                        segmod2[3] = height / 2.4 + altura
                    self.pintaCanvas(segmod2)
                else:
                    self.pintaCanvas(segmentos[cont1])
            cont1 = cont1 + 1
```

Este método se encarga de ajustar las dimensiones de las coordenadas para pintar las unidades, enviando los parámetros a **pintaCanvas,** si el número recibido es un 0, entonces no se imprime nada.

```
# pintor de las decenas
def pintaNumero2Kivy(self, numero):
    cont4 = 0
    numero4 = int(numero)
    if numero4 == 0:
        print("nothing")
    else:
        num4 = numeros[(numero4 - 1)]
        for n in num4:
            if n == 1:
                if cont4 > 0:
                    segmod3 = list(segmentos[cont4])
                    if segmod3[0] == segmod3[2]:
                        segmod3[0] = width / 4
                        segmod3[2] = width / 4
                    if segmod3[2] == x300:
                        segmod3[2] = width / 4
                    if segmod3[1] == y100:
                        segmod3[1] = y300
                    elif segmod3[1] == y150:
                        segmod3[1] = height / 2.4 + altura
                    if segmod3[3] == y100:
                        segmod3[3] = y300
                    elif segmod3[3] == y150:
                        segmod3[3] = height / 2.4 + altura
                    self.pintaCanvas(segmod3)
                else:
                    self.pintaCanvas(segmentos[cont4])
            cont4 = cont4 + 1
```

-Este método tiene la función de ajustar las coordenadas para pintar las decenas y se las envía a **pintaCanvas,** si el número recibido es un 0, entonces no se imprime nada.

```
# pintor de las centenas
def pintaNumero3Kivy(self, numero):
    cont3 = 0
    num3 = int(numero)
    if num3 == 0:
        print("nothing")
    else:
        numero3 = numeros[(num3 - 1)]
        for n in numero3:
            if n == 1:
                self.pintaCanvas(segmentos[cont3])
            cont3 = cont3 + 1
```

-Este método tiene la función de ajustar las coordenadas para pintar las centenas y se las envía a **pintaCanvas.**

```
# pintor de los millares
def pintaNumero4Kivy(self, numero):
    cont2 = 0
    numero2 = int(numero)
    if numero2 == 0:
        print("nothing")
    else:
        num2 = numeros[(numero2 - 1)]
        for n in num2:
            if n == 1:
                if cont2 > 0:
                    segmod = list(segmentos[cont2])
                    if segmod[0] == segmod[2]:
                        segmod[0] = width / 4
                        segmod[2] = width / 4
                    else:
                        segmod[2] = width / 4
                    self.pintaCanvas(segmod)
                else:
                    self.pintaCanvas(segmentos[cont2])
            cont2 = cont2 + 1
```

-Este método tiene la función de ajustar las coordenadas para pintar los millares y se las envía a **pintaCanvas.**

```
# aqui se evalúa si tenemos que pintar en el tab correspondiente
# mediante el valor de eligeCanvas si es 1, 2 o 3 para
# pintar en cada espacio que le toque
def pintaCanvas(self, seg):
    if self.eligeCanvas == 1:
        with self.painter.canvas:
            self.lineador(seg)
    elif self.eligeCanvas == 2:
        with self.painter2.canvas:
            # este es el color de la linea que se pinta
            segCont = (seg[0] + width / 6, seg[1], seg[2] + width / 6, seg[3])
            self.lineador(segCont)
    elif self.eligeCanvas == 3:
        with self.painter3.canvas:
            # este es el color de la linea que se pinta
            self.lineador(seg)
```

-Este método es casi el final de la cadena para imprimir cualquier número, es capaz de diferenciar en que pestaña debe dibujar según el parámetro **eligeCanvas** y le envia al método **lineador** para que al fin pinte.

```
# metodo final que pinta una linea segun las coordenadas que hemos dicho con seg
def lineador(self, seg):
    # este es el color de la linea que se pinta
    Color(203 / 255, 205 / 255, 1, 1)
    Line(points=[seg], width=10)
```

-Aquí acaba la magia, este método pinta según las coordenadas que le llegan.

```
class CanvasWidget(Widget):
    pass


class CanvasWidget2(Widget):
    pass


class CanvasWidget3(Widget):
    pass
```

-Aquí declaramos las clases de los canvas para poder añadirlos luego a la app y llamarlos según nuestro antojo.

```
class RelojWidget(Widget):
    number = NumericProperty()

    # Para incrementar el tiempo
    def increment_time(self, interval):
        # txtFin
        self.number += .1
        Aplicacion.pinta2(Aplicacion.pinta2(), int(self.number))

    # Para empezar la cuenta
    def start(self):
        Clock.unschedule(self.increment_time)
        Clock.schedule_interval(self.increment_time, .1)

    # Para parar la cuenta
    def stop(self):
        Clock.unschedule(self.increment_time)
```

-Hemos declarado el widget del reloj para la pestaña del contador.

-Se declara la propiedad numérica "number", para determinar el comienzo de la cuenta, además consta de tres métodos, uno que se dedica a incrementar el tiempo, y es llamado periódicamente desde la función "**start**" y no parará hasta que activemos el botón de "**stop**", además desde la parte en idioma Kivy podemos reiniciar la animación como vemos en la siguiente captura:

```
MDFillRoundFlatButton:
    text: 'REINICIAR'
    on_press: root.number = 0
    text_color: (1, 209 / 255, 60 / 255, 1)  # Azul
    line_color:(1, 209 / 255, 60 / 255, 1) # Amarillo
    md_bg_color:(56/255, 62/255, 241/255, 0.81) # Azul
    font_style: 'Button'
```

-Desde la propiedad "**on_press**" le especificamos al botón que marque "**number**" a cero de nuevo.

```
# ejecución de la app
if __name__ == '__main__':
    Aplicacion().run()
```

-Por último, este código le indica al programa que inicie su ejecución.

5.- Resultados de las diferentes pruebas realizadas al proyecto para su visto bueno

La primera prueba que realizamos fue con tkinter donde probamos a pintar los números cistercienses. Una vez terminado ese ensayo, pasamos a hacer nuestra primera prueba con Kivy y KivyMd, donde añadimos las tres pestañas que iban a componer la aplicación y comenzamos a desarrollar el primero, el conversor.

Una vez terminado, hicimos nuestro primer intento de crear la apk donde tras varias pruebas erróneas conseguimos crearla e instalarla en un móvil, donde funcionaba perfectamente por lo que a partir de ahí nos pusimos a desarrollar las dos pestañas que nos quedaban para después volver a formar el apk final.

Más tarde había un problema de compatibilidad entre el uso del reloj de nuestra app en kivy y nuestro teléfono android donde realizamos las pruebas, este dispositivo es un Xiaomi Redmi Note 9, a continuación se enumeran sus características:

Xiaomi Redmi Note 9: Características

- Procesador: Procesador MediaTek Helio G85 Octa-core.
- Sistema Operativo: Android 10.
- Memoria RAM: 4GB.
- Almacenamiento: 128GB.
- Pantalla: 6,53 pulgadas.
- Resolución: 2340 x 1080 FullHD+
- Brillo: 466 nits.
- Cámara principal: Cuádruple, 48MP+8MP+2MP+2MP.

A día de hoy sigue habiendo alguna incompatibilidad que será resuelta de cara a la defensa del proyecto.

6.- Presupuesto de la realización del proyecto

PRESUPUESTO MANO DE OBRA				
Actividad	Precio €/h	Horas	Total sin IVA	Total con IVA
Investigación	30	100	3.000,00 €	3.630,00 €
Diseño	30	120	3.600,00 €	4.356,00 €
Desarrollo	30	230	6.900,00 €	8.349,00 €
Mantenimiento	30	3	90,00 €	108,90 €
Documentación	30	18	540,00 €	653,40 €
Presentación	30	8	240,00 €	290,40 €
Google Play Cuenta Desarrollador	30		30,00 €	36,30 €
		TOTAL:	480	14.400,00 €	17.424,00 €

TOTAL COSTE PROYECTO:	17.424,00 €

7.- Propuestas de mejora a futuro

En un futuro, unas de las posibles mejoras que se nos vienen a la cabeza sería algún cambio de diseño en cuanto a Labels, Buttons, TextField… dándoles otro aspecto, otros colores etc... Y más concretamente en la pestaña del juego, añadiremos algunos sonidos para hacer más entretenida la experiencia, una cuenta atrás para poder responder en un tiempo concreto y si no se hace dar como fallado ese número, un contador de aciertos al final del juego donde se vaya guardando el mejor tiempo y el mayor número de aciertos para así crearle al jugador la necesidad de superación constante…

Otra mejora de cara al futuro, es poder realizar una entrada de números cistercienses, de manera que, podamos dibujarlos en pantalla con el dedo y que la aplicación sepa reconocerlos a modo de ampliar la pestaña del juego.