UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMATICA

TESIS DE GRADO

“MAQUETACION DE COMPONENTES PARA APLICACIONES MOVILES BAJO NORMAS FORMALES DE DIAGRAMACION”

PARA OPTAR AL TÍTULO DE LICENCIATURA EN INFORMATICA
MENCIÓN: INGENIERIA DE SISTEMAS INFORMÁTICOS

POSTULANTE: OMAR RODRIGO POMAR AVALOS
TUTOR METODOLOGICO: M.Sc. FRANZ CUEVAS QUIROZ
ASESOR: Lic. GROVER ALEX RODRIGUEZ RAMIREZ

LA PAZ – BOLIVIA

2018

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMÁTICA

LA CARRERA DE INFORMÁTICA DE LA FACULTAD DE CIENCIAS PURAS Y NATURALES PERTENECIENTE A LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la referencia correspondiente respetando normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

A mi madre, Florencia Avalos, un ser humano ejemplar e inagotable, consecuente con sus compromisos como con sus sentimientos.

Omar Rodrigo Pomar Avalos

.

AGRADECIMIENTO

Al conocimiento objetivo y probatorio que me permiten concluir con esta investigación. Y a quien corresponda, gracias.

RESUMEN

Originalmente utilizadas para proyectos destinados a impresión mediante tintas y papeles, las retículas se hallan presentes en los sistemas de interfaces de usuario en diferentes proporciones de relevancia compartiendo importancia con los estudios de experiencia de usuario de páginas web y aplicaciones en general.

El presente trabajo plantea una aplicación más purista del sistema de retículas en maquetaciones de aplicaciones móviles mediante la selección de normas de diagramación aplicables a la construcción de interfaces de usuario planeadas para dispositivos móviles.

Se utiliza la metodología de Bruno Munari, que es aplicable a proyectos de investigación en general, dado que podrían presentarse etapas de creatividad que son contempladas en esta metodología.

Utilizando a Unity como plataforma de desarrollo se consigue programar un prototipo funcional en base a normas de diagramación que obtiene maquetaciones de interfaces de usuario para dispositivos móviles.

INDICE

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE PROGRAMAS

CAPITULO I: INTRODUCCION

El diseño es un proceso de creación visual con un propósito. A diferencia de la pintura y de la escultura, que son la realización de las visiones personales y los sueños de un artista, el diseño cubre exigencias prácticas. Una unidad de diseño gráfico debe ser colocada frente a los ojos del público y transportar el mensaje específico (Wong, 1991).

El diseño no es solo adorno. La silla bien diseñada no solo posee una apariencia exterior agradable, sino que se mantiene firme sobre el piso y da un confort adecuado a quien se siente en ella. Además debe ser segura y bastante duradera. (Wong, 1991).

Dentro del diseño gráfico, la diagramación, o maquetación, es el proceso mediante el que los componentes que integran el mensaje visual, texto e imagen, son organizados en un espacio bajo criterios de jerarquización buscando la fácil lectura del mensaje bajo una apariencia estética agradable.

Se considera una aplicación móvil al software orientado a teléfonos inteligentes que permiten al usuario ejecutar tareas específicas, muchas veces aprovechando las características de estos dispositivos (función táctil, acelerómetro, GPS etc.).

1.1 ANTECEDENTES

Existen herramientas online y de escritorio que ayudan a la tarea de maquetación de aplicaciones en general que incluyen la maquetación de aplicaciones móviles:

Balsamiq

Está es una aplicación que te permite crear wireframes en forma de dibujos. La particularidad

de esta es que es muy robusta y te da la flexibilidad de crear proyectos bastante grandes sin perder fluidez. Una de las ventajas de esta aplicación es que tiene la posibilidad de hacer las pantallas interactivas y ligar los mockups entre si dentro de cada proyecto. Tiene planes mensuales y en un solo pago (Román, 2012).

Prototyper

Una aplicación de escritorio gratuita (también con opción de pago) que es muy potente para crear maquetas tanto de web como de móviles. También tiene la habilidad de crear documentos de mockups con interacción (Román, 2012).

Mockflow

Uno de los puntos fuertes de esta aplicación es la gran interacción para trabajar en colaboración entre personas de un equipo. Además de tener los elementos similares a las aplicaciones anteriores, con Mockflow incluso podrás trabajar en el documento de forma desconectada de internet (Román, 2012).

Pidoco

Pidoco es otra interesante propuesta para crear diseños de prototipos e interfaces para nuestros proyectos. Aunque es una aplicación de paga las ventajas que tiene son grandes y es una buena opción. Tiene las características de las dos primeras aplicaciones, agregando la colaboración en tiempo real sobre los proyectos y correr test de forma remota (Román, 2012).

Mockingbird

Otra herramienta que no debemos dejar a un lado y que ha ganado buen terreno en el mundo de los wireframes o mockups es Mockingbird. Aunque es una aplicación online es tan potente como una aplicación de escritorio. Algunas de las características de esta aplicación es que puedes exportar tus maquetas a otros formatos como PDF, crear las pantallas que necesites por proyecto o no tener que instalar nada en tu computadora, así trabajar desde donde tu quieras (Román, 2012).

Axure

Una herramienta muy potente pero también un poco costosa es axure. Esta herramienta de creación de mockups tiene todo lo que las anteriores ofrecen con una interfaz muy ligera pero con los elementos necesarios para crear cualquier cosa en tu proyecto. Puedes dibujar y diseñar dentro de tus proyectos, interactuar entre todas las pantallas de un proyecto, hacer anotaciones en cada pantalla y colaborar con los usuarios del proyecto agregando categorías de usuario para tener un mejor control de los cambios y mejoras (Román, 2012).

Entre las tesis desarrolladas en la U.M.S.A., en el momento de la búsqueda de tesis similares, no se hallaron proyectos de aplicaciones que permitan diagramación de aplicaciones móviles, si una que desarrolla una metodología especifica solo para páginas web:

De Elviz Ivan Salgueiro Flores, “Herramienta metodológica para el desarrollo y aplicaciones web móvil”. Donde no desarrollo ninguna aplicación ejecutable pero proporciona reglas para la creación de una pagina web orientada a teléfonos móviles.

1.2 PLANTEAMIENTO DEL PROBLEMA

Al abordar la tarea de la programación de aplicaciones los componentes de su interfaz deben ajustarse a su espacio formal, definido por normas. Entonces, entre la primera planeación de una aplicación y su publicación final ajustada a esas normas se pueden identificar los siguientes problemas:

• Dentro del desarrollo de aplicaciones en general, se encuentra el desarrollo orientado a celulares. Existen claras diferencias entre las aplicaciones para escritorio y las aplicaciones para dispositivos móviles como la pantalla y los métodos de entrada.

• La mayoría de los programadores independientes no conocen las normas básicas de diseño o las utilizan erróneamente.

• El desarrollo de aplicaciones independientes realizadas por una sola persona o grupo de personas que pueden tener profundo conocimiento de programación pero carecen de un profesional en el área de diseño, por que nunca se han planteado su utilidad o porque lo consideran irrelevante, puede dar como resultado aplicaciones funcionales pero que por la apariencia no llegan a ser del agrado general.

El problema de investigación es el siguiente:

¿Una herramienta de maquetación permite maquetar interfaces de aplicaciones móviles en base a normas de diagramación?

1.3 JUSTIFICACION

El desarrollo de aplicaciones móviles por aficionados y profesionales independientes se ve estancado dado a la poca aceptación de estos proyectos por el público en general debido a un aspecto visual poco profesional.

El poseer una aplicación en el celular que permita maquetación de aplicaciones móviles representará la disponibilidad inmediata de dicha aplicación, que de un modo casual (al ser utilizada en cualquier momento) proveerá una herramienta seria de diseño de interfaz.

Una aplicación de diagramación busca beneficiar a programadores que desean publicar su proyecto con un aspecto más profesional sin recurrir a especialistas del diseño gráfico.

1.4 HIPOTESIS

La herramienta de maquetación en base a normas de diagramación permite el diseño de interfaces de aplicaciones móviles.

1.5 OBJETIVOS

1.5.1 Objetivo General

Desarrollar una herramienta de maquetación para aplicaciones móviles que obtenga *UI* regidas por normas de maquetación.

1.5.2 Objetivos Específicos

• Identificar los conceptos más relevantes en teoría de la diagramación.

• Diseñar los componentes de la maquetación.

• Identificar los campos reticulares para la disposición de elementos de interfaz.

• Programar un prototipo de la aplicación de maquetación.

1.6 ALCANCES

Los alcances de la presente investigación son los siguientes:

• La aplicación será desarrollada para dispositivos Android.

• La aplicación se limitará a la correcta disposición de los componentes de una interfaz sin profundizar en los colores que deberían tener estos componentes.

• Ya existen soluciones que realizan diagramado de interfaz, entonces, el presente proyecto presentará una restricción: solo se podrán disponer elementos que tengan justificación de despliegue en el espacio asignado.

CAPITULO II: MARCO TEORICO

2.1 LA METODOLOGIA DE BRUNO MUNARI

Este diseñador industrial / gráfico plantea una metodología proyectual basado en la resolución de problemas. Esta metodología evita el inventar la rueda con cada proyecto y plantea sistematizar la resolución de problemas. (Martin, 2004).

2.1.1 Definición del problema

El cliente del diseñador es la industria, esta es quien le propone el problema, pero e1 no debe salir inmediatamente en busca de una idea general que resuelva en seguida el problema, porque ésta es la manera artístico-romántica de buscar una solución. (Munari, 1989).

Lo primero que hay que hacer es definir el problema en su conjunto. Muchos diseñadores creen que los problemas ya han sido suficientemente definidos por sus clientes. Pero esto no es en absoluto suficiente. Por tanto es necesario empezar por la definición del problema, que servirá también para definir los límites en los que deberá moverse el proyectista. (Munari, 1989).

Una vez definido el problema hay que definir también el tipo de solución que se le quiere dar: una solución provisional una solución definitiva, una solución puramente comercial, una solución que perdure en el tiempo una solución técnicamente sofisticada o una solución sencilla y económica. (Munari, 1989).

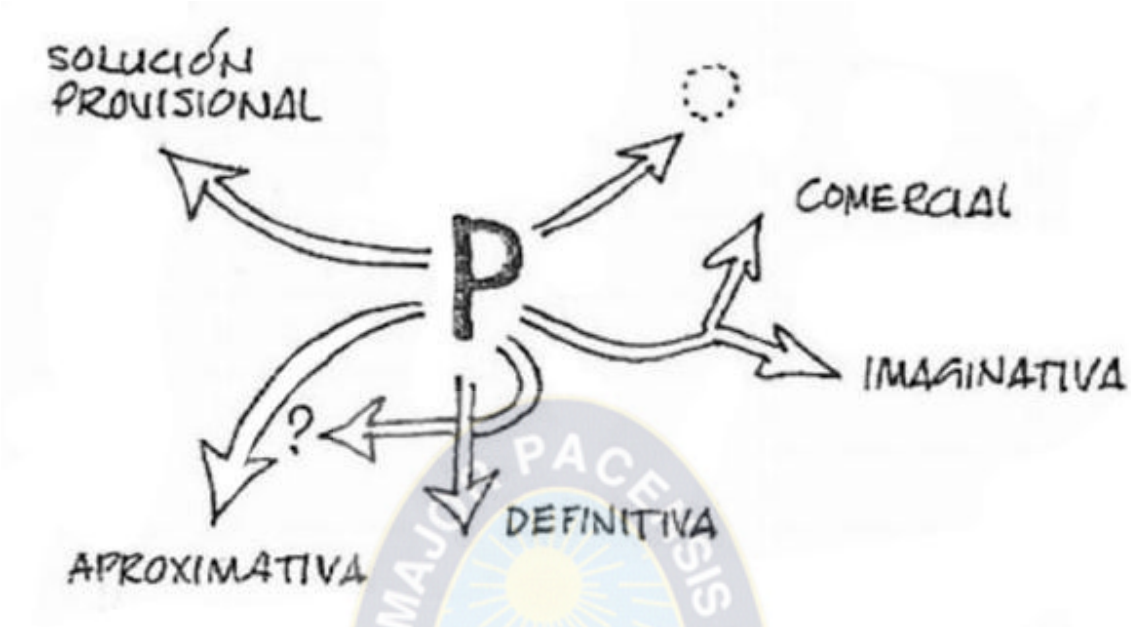


Figura 2.1: Definición de solución a un problema.

Fuente: Munari, 1989.

a) Búsqueda del tipo de solución

Se consideraron los tipos de soluciones que podrían darse al problema:

Tipo de solución	*Descripción*
Solución provisional	El prototipo puede crear una interfaz para una única aplicación requerida.
Solución definitiva	El prototipo puede crear una interfaz para cualquier aplicación requerida.
Solución puramente comercial	El prototipo solo puede crear los tipos de interfaces que sean más populares entre los usuarios.
Solución que perdure en el tiempo	El prototipo puede crear diferentes interfaces, puede aprender y puede incorporar nuevos elementos de interfaz.

Solución técnicamente sofisticada	El prototipo puede crear diferentes interfaces, puede aprender, puede incorporar nuevos elementos de interfaz, admite aspectos tipo "portrait" y tipo "landscape" de un dispositivo móvil, abarca la teoría de color y experiencia de usuario, hace ajustes de tipografía, etc.
Solución sencilla y económica	El prototipo puede crear interfaces tipo "portrait", una a una.

Tabla 2.1: Tipos de soluciones al problema principal.

2.1.2 Elementos del problema

Cualquier problema puede ser descompuesto en sus elementos. Esta operación facilita la proyección porque tiende a descubrir los pequeños problemas particulares que se ocultan tras los subproblemas. Una vez resueltos los pequeños problemas de uno en uno (y aquí empieza a intervenir la creatividad abandonando la idea de buscar una idea), se recomponen de forma coherente a partir de todas las características funcionales de cada una de las partes y funcionales entre sí, a partir de las características materiales, psicológicas, ergonómicas, estructurales, económicas y, por último, formales. (Munari, 1989).

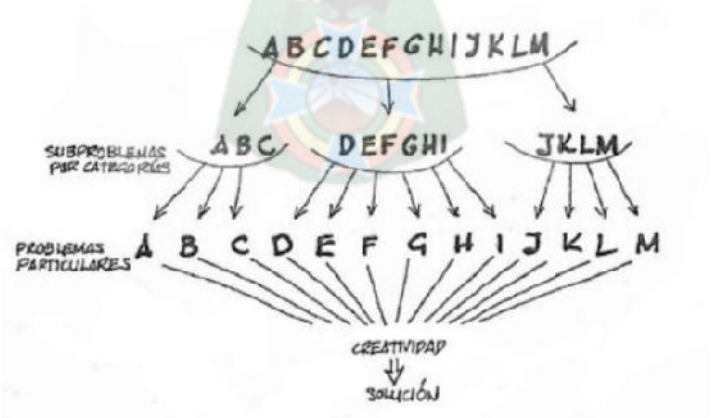


Figura 2.2: Descomposición de un problema en sus elementos.

Fuente: Munari, 1989.

El principio de descomponer un problema en sus elementos para poder analizarlo procede del método cartesiano. Como los problemas, sobre todo hoy en día, se han convertido en muy complejos y a veces en complicados, es necesario que el proyectista tenga toda una serie de informaciones sobre cada problema particular para poder proyectar con mayor seguridad. (Munari, 1989).

Cada subproblema tiene una solución óptima que no obstante puede estar en contradicción con las demás. La parte más ardua del trabajo del diseñador será la de conciliar las diferentes soluciones con el proyecto global. La solución del problema general consiste en la coordinación creativa de las soluciones de los subproblemas. (Munari, 1989).

2.1.3 Recopilación de datos

Consiste en la recopilación de datos relevantes para el proyecto encargado como proyectos similares ya conseguidos y documentación en general.

2.1.4 Análisis de datos

Con los datos obtenidos, se procede al desglose de los mismos para que sean relevantes para el proyecto.

2.1.5 Creatividad

La creatividad reemplazará a la idea intuitiva, vinculada todavía a la forma artístico-romántica de resolver un problema. Así pues, la creatividad ocupa el lugar de la idea y procede según su método. Mientras la idea, vinculada a la fantasía, puede proponer soluciones irrealizables por razones técnicas, materiales o económicas, la creatividad se mantiene en los límites del problema, límites derivados del análisis de los datos y de los subproblemas. (Munari, 1989).

2.1.6 Materiales - Tecnologías

La sucesiva operación consiste en otra pequeña recogida de datos relativos a los materiales y a las tecnologías que el diseñador tiene a su disposición en aquel momento para realizar su proyecto. La industria que ha planteado el problema al diseñador dispondrá ciertamente de una

tecnología propia para fabricar determinados materiales y no otros. Por tanto es inútil pensar en soluciones al margen de estos dos datos relativos a los materiales y a las tecnologías. (Munari, 1989).

2.1.7 Experimentación

Es ahora cuando el proyectista realizará una experimentación de los materiales y las técnicas disponibles para realizar su proyecto. Muy a menudo materiales y técnicas son utilizados de una única forma o de muy pocas formas según la tradición. Muchos industriales dicen: Siempre lo hemos hecho así, ¿por qué habría que cambiar? En cambio la experimentación permite descubrir nuevos usos de un material o de un instrumento. (Munari, 1989).

2.1.8 Modelos

Los modelos serán puntos de referencia para ser reproducidos, obtenidos a través de la experimentación, estos ayudaran a la resolución de subproblemas y estos a la solución general. De esta forma obtendremos un modelo de lo que eventualmente podrá ser la solución del problema. (Munari, 1989).

2.1.9 Verificación

Este es el momento de llevar a cabo una verificación del modelo o de los modelos (puede ocurrir que las soluciones posibles sean más de una). Se presenta el modelo a un determinado número de probables usuarios y se les pide que emitan un juicio sincero sobre el objeto en cuestión. Sobre la base de estos juicios se realiza un control del modelo para ver si es posible modificarlo, siempre que las observaciones posean un valor objetivo. (Munari, 1989).

En base a todos estos datos ulteriores se pueden empezar a preparar los dibujos constructivos a escala o a tamaño natural, con todas las medidas exactas y todas las indicaciones necesarias para la realización del prototipo. (Munari, 1989).

2.2 LA RETICULA

La retícula tipográfica es un principio organizador en diseño gráfico cuya influencia está profundamente arraigada en la práctica actual y, al mismo tiempo, se combate en las escuelas

de diseño, un principio que se ensalza y se vilipendia por igual debido a las ideas absolutas inherentes a su concepción. Se trata de un principio enraizado en las más antiguas sociedades del planeta. Llevar adelante u existencia con algún tipo de significado y crear un orden que haga comprensible ese significado es una de las actividades que diferencian a nuestra especie de todas las demás. El pensamiento estructural, incluso antes de su codificación más reciente, llevada a cabo por la modernidad europea y américa, ha sido el sello de las culturas que se han esforzado por alcanzar la civilización. La china, la japonesa, la grecorromana, la inca... todas estas culturas actuaban de acuerdo con ideas estructurales a la hora de trazar sus ciudades, de dirigir sus guerras, de definir la disposición de sus imágenes. En muchos casos, esta estructura estaba implícita en la noción de los ejes cruzados que corresponden a la intersección del cielo y la tierra. (Samara, 2006).

2.1.2 Partes de una retícula

a) Los márgenes

Los márgenes son el espacio entre el borde del medio que soporta la composición de una retícula y el contenido de esta misma retícula.

b) Las líneas de flujo

Las líneas de flujo son alineaciones que rompen el espacio dividiéndolo en bandas horizontales. Estas líneas guían alojo a través del formato y pueden utilizarse para imponer paradas adicionales y crear puntos de inicio para el texto o las imágenes. (Samara, 2006).

c) Las columnas

Las columnas son las divisiones verticales del trazado en el cual el texto es típicamente fluido. La grasa o delgada y aun torcido, cómo son las columnas manipuladas tiene un impacto marcado en la legibilidad del texto. (Ambrose, Harris, 2007).

d) Campos o módulos

Dividiendo una retícula en campos diferentes o módulos incrementa el rango de espacios disponibles para un diseñador, al mantener una estructura básica de la columna. Esto facilita

un uso más dinámico de texto e imágenes. (Ambrose, Harris, 2007).

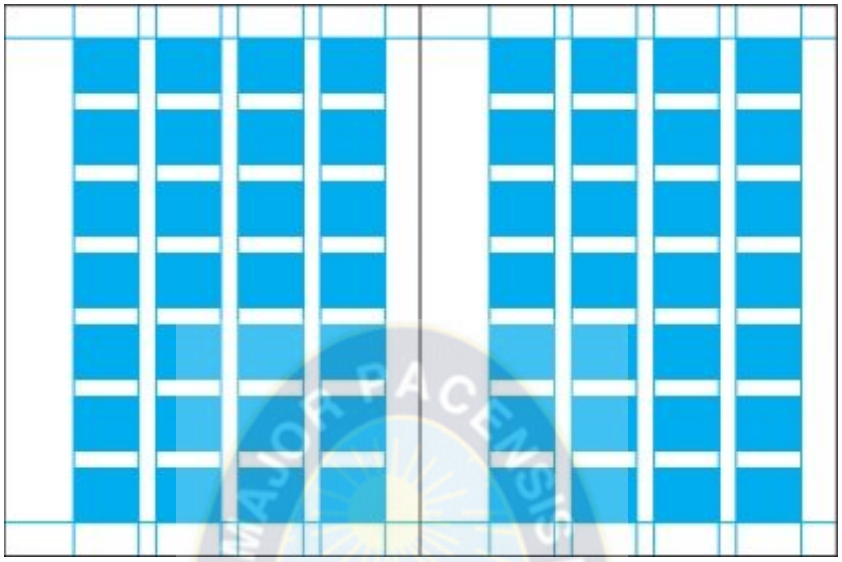


Figura 2.3: Los módulos en una retícula.

Fuente: Ambrose, Harris, 2007.

e) Las zonas espaciales

Son la unión de módulos.

f) Los marcadores

Indican el lugar donde van a colocarse elementos que deben ir entre el margen y la retícula como por ejemplo los números de página.

2.1.3 Tipos de retículas

a) Retícula de manuscrito

Figura 2.4: Ejemplos de retículas de manuscrito.

Fuente: Samara, 2006.

Este tipo de retícula tan solo cuenta márgenes, marcadores y un modulo para el texto.

b) Retícula de columnas

Figura 2.5: Ejemplos de retículas de columnas.

Fuente: Samara, 2006.

En la maquetación editorial, a menudo se usan retículas de tres columnas y retículas de una o

dos columnas asimétricas. Una retícula precisa de cuatro columnas no excluye necesariamente una maquetación dinámica. En esta doble página, el cambio de escala de algunos elementos tipográficos contrasta con la retícula. (Samara, 2006).

La información que es discontinua presenta la ventaja de que puede disponerse en columnas verticales. Dado que las columnas pueden depender unas de las otras en el caso del texto corrido, pueden ser independientes si se trata de pequeños bloques de texto, o bien pueden cruzarse para crear columnas más anchas, la retícula de columna es muy flexible y puede utilizarse para separar diversos tipos de información. Por ejemplo, pueden reservarse algunas columnas para el texto corrido y las imágenes de mayor tamaño, tanto que los pies de foto pueden situarse en la una columna adyacente: esta disposición separa claramente los pies de foto del material principal, pero a la vez permite que el diseñador establezca una relación directa entre ambos. (Samara, 2006).

En una retícula de columna existe también una estructura subordinada. Se trata de las líneas de flujo: intervalos verticales que le permiten al diseñador acomodar los cortes poco frecuentes que se dan en el texto o las imágenes en una página, y que crean bandas horizontales que atraviesan el formato. La línea superior es una clase concreta de línea de flujo: se trata de la primera línea del bloque de texto corrido, situada encima de todas las demás. Esta línea define la distancia vertical desde la parte superior del formato del papel, desde donde comenzará siempre el texto de las columnas. En ocasiones, una línea de flujo próxima al borde superior de la página indica el lugar destinado a los folios explicativos, a la paginación o a las divisiones de sección; otras líneas de flujo adicionales, situadas en el medio o hacia el borde inferior de la página, pueden establecer áreas que el diseñador destine a imágenes solamente, o bien a diferentes tipos de texto corrido que deben aparecer junto al texto principal, como referencias temporales, subartículos o citas. (Samara, 2006).

c) Retícula modular

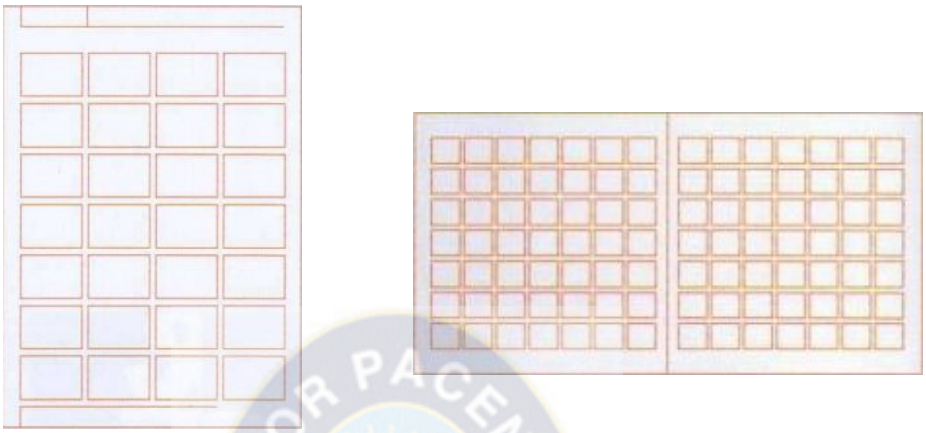


Figura 2.6: Ejemplos de retículas modulares.

Fuente: Samara, 2006.

Los proyectos de gran complejidad requieren cierto grado de control que va más allá del que ofrecería una retícula de columnas. En esta situación, la elección más eficaz podría ser una retícula modular. (Samara, 2006).

Una retícula modular es, en esencia, una retícula de columnas con un gran número de líneas de flujo horizontales que subdividen las columnas en filas, creando una matriz de celdas que se denominan módulos. Cada módulo define una pequeña porción de espacio informativo. Agrupados, estos módulos definen áreas llamadas zonas espaciales, a las que pueden asignárseles funciones específicas. El grado de control dentro de la retícula depende del tamaño de los módulos. Los módulos pequeños proporcionan mayor flexibilidad y precisión, pero el exceso de subdivisiones puede resultar confuso o redundante. (Samara, 2006).

Una retícula modular también resulta adecuada para el diseño de información tabulada, como cuadros, formularios, programaciones o sistemas de navegación. La repetición rigurosa del módulo ayuda a estandarizar el espacio de las tablas o los formularios, y también puede contribuir a integrarlos en la estructura del texto y las imágenes que los rodean. (Samara, 2006).

d) Retícula jerárquica

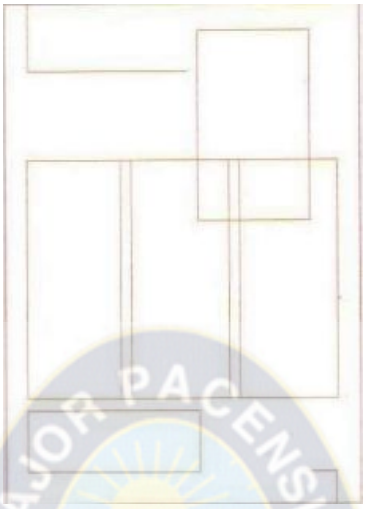


Figura 2.7: Ejemplo de retícula jerárquica.

Fuente: Samara, 2006.

Las páginas Web constituyen el ejemplo más común de retículas jerárquicas. Las alineaciones cambian según el contenido, pero mantienen las proporciones que las integran en el conjunto. (Samara, 2006).

A veces, las necesidades informativas y visuales de un proyecto exigen una retícula extraña que no encaje en ninguna otra categoría. Estas retículas se adaptan a las necesidades de la información que organizan, pero están basadas más bien en la disposición intuitiva de alineaciones vinculadas a las proporciones de los elementos, y no en intervalos regulares y repetidos. La anchura de las columnas, al igual que los intervalos entre éstas, tienden a presentar variaciones. (Samara, 2006).

2.3 PLATAFORMAS DE DESARROLLO MOVIL

Entre las principales plataformas de desarrollo móvil se tienen:

2.3.1 Android Studio

El IDE oficial creado por Google y al que hace referencia toda la documentación oficial

existente. Es por ello que es el entorno que con más facilidad permite seguir las directrices de diseño de Google y encontrar nuevas librerías para nuestros proyectos. Y aunque resulta un poquito complicado de configurar, lo cierto es que va mejorando con rapidez y ya es mejor de lo que era antes, especialmente cuando salió. (Lasso, 2017).

Android Studio es una gran opción de cara a desarrollar servicios y herramientas de productividad, aunque también permite el desarrollo de juegos sencillos. Si se quiere hacer algo que necesite el uso de una física más avanzada, se puede recurrir a librerías como LibGDX para que no haya que hacer todo desde cero, pero si lo que se quiere hacer es más complejo, será necesario conocer Android NDK (Native Development Kit) que permite usar C++ para un mayor uso de la CPU, algo que no es tan poderoso como otras opciones. (Lasso, 2017).

2.3.2 Eclipse

Al principio, Eclipse era el IDE recomendado para el desarrollo de aplicaciones. De hecho, fue la opción oficial que se planteaba desde Google hasta la aparición de Android Studio. Desde el lanzamiento de este último, se suele recomendar que los desarrolladores que utilicen Eclipse migren hacia él. (Lasso, 2017).

La configuración de Eclipse y el flujo de trabajo son muy parecidos a los de Android Studio. Pero el hecho de que no sea un entorno de desarrollo específico para la plataforma de Mountain View, sino que está pensado para desarrollar en varias plataformas e idiomas, hace que la experiencia en general sea más lenta y que haya cierta tendencia a que se produzcan errores. (Lasso, 2017).

2.3.3 Xamarin

Es un entorno creado por Microsoft, gratuito y que viene incluido en Visual Studio.

En Xamarin se desarrolla en C# y, aunque utilizarlo significa alejarse de la experiencia de desarrollo «puro» para Android, es una alternativa a considerar para aquellos que Java no les guste o no sea su fuerte. Entre algunas de las cosas interesantes con que cuenta están las

pruebas automatizadas a través de múltiples dispositivos reales conectados a la nube y el soporte de Monogame, un framework multiplataforma para juegos basados en el framework XNA de Microsoft, una gran herramienta para hacer juegos 2D y 3D. (Lasso, 2017).

2.3.4 AIDE

AIDE, que significa Android IDE, se diferencia del resto de entornos aquí reseñados en que es se ejecuta directamente en Android, de manera que es posible desarrollar una aplicación desde el mismo dispositivo móvil (como el celular o la tablet) y hacer pruebas allí sin necesidad de un emulador u otro dispositivo pensado únicamente para ello. Por otra parte, su funcionamiento es bastante similar al de Android Studio y Eclipse. (Lasso, 2017).

2.3.5 Python

Este popular lenguaje de programación puede ser también utilizado para desarrollar aplicaciones de Android. Existen varias opciones para hacerlo, como PyMob o la librería pgs4a (Pygame Subset for Android). Es un lenguaje muy accesible y fácil de aprender, considerado como muy elegante por muchos desarrolladores, por lo que puede ser una opción interesante a la hora de elaborar aplicaciones sencillas, porque para cosas complejas lo cierto es que no es muy recomendable porque hay que considerar que se pierden funcionalidades en comparación con el desarrollo en Android Studio. (Lasso, 2017).

2.3.6 B4A

B4A significa «Basic for Android» y es un IDE que permite desarrollar aplicaciones utilizando el lenguaje BASIC (Beginners All Purpose Symbolic Instruction Code), que es muy fácil de aprender pues prácticamente se lee como si fuese inglés. (Lasso, 2017).

Lo interesante de B4A es que no sacrifica ninguna funcionalidad a la hora de cumplir con lo que promete. Se puede acceder a las mismas API y librerías que en Android Studio, de manera que con un poco de creatividad se pueden hacer muchas cosas. Otra ventaja es que el código que usemos se puede trasladar con facilidad a B4i https://www.b4x.com/b4i.html de manera que podamos generar sin muchas complicaciones también una aplicación para iOS. Por otro

lado, en Android Studio es más fácil construir aplicaciones con Material Design que en B4A y, como es lógico, se pueden aprovechar mejor las nuevas características en el momento en que son introducidas. (Lasso, 2017).

2.3.7 Unity

Unity es tanto un motor como un entorno de desarrollo que nos permite crear juegos multiplataforma de todo tipo: desde cosas muy simples hasta muy complejas con física realista, iluminación dinámica y gráficos en 3D. La capacidad de añadir código en C# o Java nos brinda además toda la flexibilidad que necesitemos para poder ir más allá de lo que el IDE nos ofrece por sí mismo. (Lasso, 2017).

Muchos de los juegos más exitosos de la Play Store han sido elaborados con Unity. Unity ahorra la necesidad de tener que construir un motor desde cero, con todo lo que esto significa. Dado el éxito que las plataformas móviles han tenido en lo que se refiere a los juegos, Unity se ha convertido en uno de los principales referentes en cuanto al desarrollo y en una opción preferencial dentro de este campo. (Lasso, 2017).

Unity es esencialmente un ambiente de desarrollo para plataformas múltiples (Android, IOS, Blackberry, Windows Phone, Windows, PlayStation, Xbox, Macintosh, Linux, y Web). Unity es una tecnología muy popular y efectiva para crear los juegos de 2D y 3D y aplicaciones. Esta tecnología provee muchas herramientas útiles y efectivas para solucionar diversas tareas. (Cogut, 2015).

2.4 LENGUAJES DE PROGRAMACION

2.4.1 Tabla comparativa

Lenguaje	Características	Fortalezas	Debilidades
PHP	Utilizado para generar páginas web dinámicas. Se ejecuta en el servidor. Los usuarios no pueden ver el código PHP	Su sintaxis es muy similar a otros lenguajes. Fácil. Es un lenguaje muy popular tiene una comunidad muy grande.	Necesita un servidor para funcionar. La POO es deficiente para aplicaciones grandes. Todo el trabajo se realiza

	únicamente reciben en sus navegadores código HTML. Las páginas que genera son visibles para prácticamente cualquier navegador y computadora o dispositivos móviles que pueda interpretar el HTML. No se necesita la instalación de PHP en el lado del cliente. Versiones resiente permiten la POO. Lenguaje de alto nivel.	Rápido. Multiplataforma. Maneja base de datos. Bastante documentado. Libre y gratuito. Varias funciones. No requiere definición de variables. Puede ser combinado junto a HTML. Tiene muchos frameworks que facilitan el desarrollo en este lenguaje. Muchos servicios de alojamiento web tienen PHP.	el en servidor y mucha información o solicitudes pueden ser ineficiente.
RUBY	Orientado a objetos. Lenguaje de alto nivel. Sintaxis similar a Python y Perl. Opensource. Lenguaje para la creación de aplicaciones de escritorio y aplicaciones web.	Diferencia entre mayúsculas y minúsculas. Maneja excepciones. Puede cargar librerías si el sistema operativo lo permite. Multiplataforma. Portátil. Desarrollo de bajo costo. Software libre.	Es relativamente nuevo y no cuenta con mucha documenta4ción en comparación con otros lenguajes de programación. No está muy difundido en relación a otros lenguajes.
JSP (Java Server Pages)	Lenguaje para creación de sitios dinámicos. Necesita un servidor Tomcat. Motor basado en servlets de java. Multiplataforma.	Ejecución rápida de servlets. Código bien estructurado. Integridad con módulos java. La parte dinámica está escrita en java.	Complejidad de aprendizaje.
ASP (Active Server Pages)	Desarrollado por Microsoft. Tecnología del lado del servidor. Requiere de Internet Información Server (IIS).	Usa visual Basic script Comunicación optima con SQL server Soporta JScript	De paga. El hospedaje de sitios web es costos. Necesita de mucho código para funciones sencillas.
ASP.NET	Sucesor de ASP. Creada por Microsoft. De paga. Orientado a objetos.	Controles de usuarios y personalizados. Fácil mantenimiento. Incremento en velocidad.	Mayor consumo de recursos.

		Mayor seguridad.	
Python	Permite la creación de todo tipo de programas incluso sitios web. No requiere de compilación es un código interpretado.	Libre y código fuente abierto. Lenguaje de propósito general. Multiplataforma. Orientado a objetos. Portable.	Los lenguajes interpretados suelen ser relativamente lentos.
JavaScript	Es un lenguaje interpretado. Es similar a java. Es orientado a objetos.	Los script tienen capacidad limitada por razones de seguridad. Se ejecuta del lado del cliente. Lenguaje de scripting seguro y fiable.	No soporta herencias. Código visible por cualquier usuario. El código debe ser descargado completamente. Puede poner en riesgo la seguridad del sitio con el actual problema llamado XSS (significa en inglés Cross Site Scripting renombrado a XSS por su similitud con las hojas de estilo CSS).
C++	Orientado a objetos. Rápido.	Ideal para sistemas robustos. IDEs de desarrollo son DEV C++, BORLAND C, TURBO C. Es multiplaforma.	No soporta creación de aplicaciones web. Complejo visualmente.
C	Popular para la creación de software de sistema.	Rápido. Eficiente. Es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix.	No es popular para la creación de aplicaciones. Sintaxis compleja.
C#	Está orientado a objetos. Esta estandarizado por Microsoft como parte de su plataforma net.	Se desempeña de forma plena en los sistemas operativos Windows. Sintaxis más en comparación con C y C++. Posibilidad de realizar aplicaciones web, de escritorio y móviles.	Requiere un mínimo de 4 gb para su instalación.
Java	Es orientado a objetos.	Al ser orientado a objetos	Es un lenguaje

	Multiplataforma.	permite su modularización. Permite la creación de aplicaciones de escritorio. Tiene soporte a desarrollo de aplicaciones móviles y web.	interpretado así que es relativamente lento en comparación con otros lenguajes.

Tabla 2.2: Tabla comparativa de los lenguajes de programación

Fuente: http://desarrollowebydesarrolloweb.blogspot.com/2015/02/tabla-comparativa-de-los-lenguajes-de.html. Febrero 2015.

2.4.2 C#

C# es un lenguaje de programación desarrollado por Microsoft, orientado a objetos. Se trata de un lenguaje simple, eficaz y con seguridad de tipos. Las numerosas innovaciones de C# permiten desarrollar aplicaciones rápidamente y mantener la expresividad y elegancia de los lenguajes de estilo de C. (Ortego, 2017).

La sintaxis viene derivada de C y C+ + y utiliza el modelo de objetos de la plataforma .NET, muy parecido al de Java, aunque incluye mejoras propias de otros lenguajes. Como curiosidad, el nombre de este lenguaje fue inspirado por la escala musical. En ella, la letra C equivale a la nota musical do y el símbolo # significa sostenido, lo que indica que es un semitono más alta. Así, C# sugiere que es superior a C y C++. (Ortego, 2017).

2.5 APLICACION DEL TONO Y EL COLOR EN EL DISEÑO DE INTERFAZ DE USUARIO

2.5.1 Teoría del Color en el diseño de interfaces

La clave para definir una paleta de color en un sitio web, una app o cualquier producto digital está en habilidad de manipular un color, no solo elegirlo.

a) Cantidad de Colores

Repasando grandes productos o aplicaciones se puede descubrir esta tendencia: La complejidad de la interacción tiende a ser inversamente proporcional a la cantidad de colores. (Platzi, 2017).

b) Variaciones de color

Para que un color sea funcional, predecible y aún así transmita el espíritu de una marca se necesitan ligeras transformaciones del mismo para construir todos los componentes que necesitemos. (Platzi, 2017). Se tienen los siguientes ejemplos:

Esta es la vista móvil de Quora: 1 color principal y 2 variaciones.

Figura 2.8: Interfaz de Quora.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

Ahora Facebook: 1 color principal y 2 variaciones.

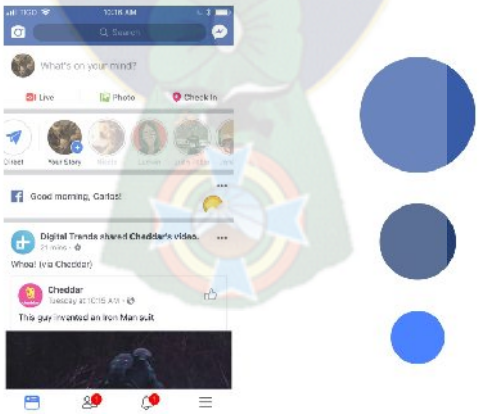


Figura 2.9: Interfaz de Facebook.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

El camino fácil para obtener variaciones de color es tomar el color principal y llevarlo hacia el negro para obtener una variación más oscura o hacia el blanco para obtener una versión más clara. (Platzi, 2017).

Se tiene el siguiente ejemplo del mundo real para obtener variaciones más interesantes:

Figura 2.10: Ejemplo del mundo real para obtener colores.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

Al ver con detenimiento el panel de color se tiene un comportamiento menos predecible pero un resultado mucho más interesante.(Platzi, 2017):

Figura 2.11: Paleta de colores.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

c) Profundizando en el color digital

El modelo de color con el que se trabaja acostumbradamente es el RGB, combinación de valores de 0 a 255 para cada canal. Un sistema muy útil es el HSB. (Platzi, 2017).

HSB

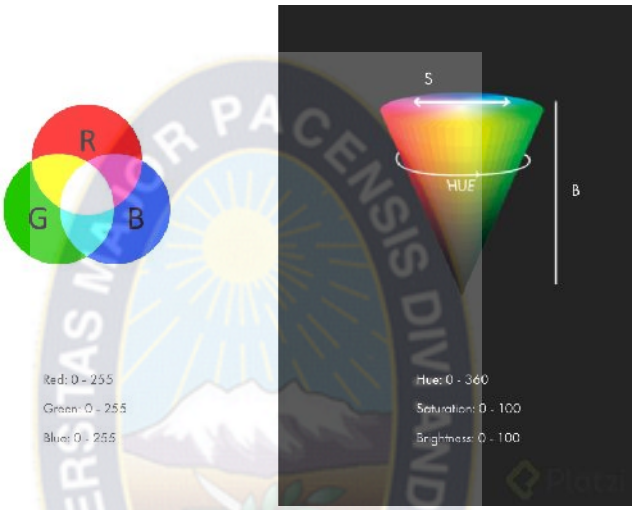


Figura 2.12: Sistema RGB vs. sistema HSB.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

Hue Matiz

Escala de 0 a 360 de todos los colores “puros”.

Saturation Saturación

Escala de 0 a 100 dónde 100 es la saturación máxima de un color y 0 es blanco

Brightness Brillo.

Escala de 0 a 100 siendo 100 el color limpio y 0 el negro puro

Regresando a la Figura 2.12 hay unos valores interesantes. En la variación oscura la saturación aumentó y el brillo bajó, al contrario de la variación clara donde la saturación bajó y el brillo aumentó. (Platzi, 2017).

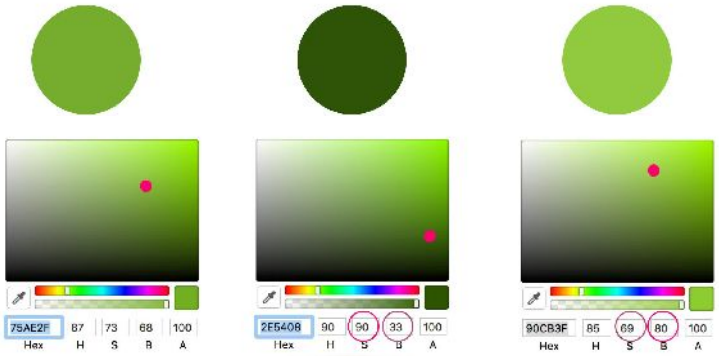


Figura 2.13: Reconocimiento de un patrón en la paleta de colores.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

Esto ya es un patrón repetible:

Figura 2.14: Patrón repetible para la obtención de colores.

Fuente: https://platzi.com/blog/color-en-interfaces/. Octubre, 2017.

2.7 RESOLUCIONES FRECUENTES EN SMARTPHONES

Descripción	*Resolución en pixeles*
QVGA	320 x 240
HVGA	480 x 320
WVGA	800 x 480
QHD	960 x 540
HD ó 720p	1280 x 720
WXGA	1280 x 800
FHD, Full HD ó 1080p	1920 x 1080

2K	2048 x 1080
Retina Display (solo en dispositivos Apple)	2048 x 1536

Tabla 2.3: Resoluciones frecuentes en smartphones.

Fuente: https://www.elgrupoinformatico.com/todo-sobre-las-resoluciones-pantalla-moviles-t18462.html, 3 de junio de 2018.

2.8 PAGINAS Y SERVICIOS WEB DE DIAGRAMACION DE APLICACIONES

En la búsqueda de aplicaciones, servicios y soluciones similares al problema propuesto en este trabajo, se hallaron:

2.8.1 Pttrns

Esta página web ofrece ejemplos dedicados a las interfaces de usuario, agrupados por categorías y plataformas. Es un servicio gratuito aunque cuenta con una versión de pago que elimina publicidad.

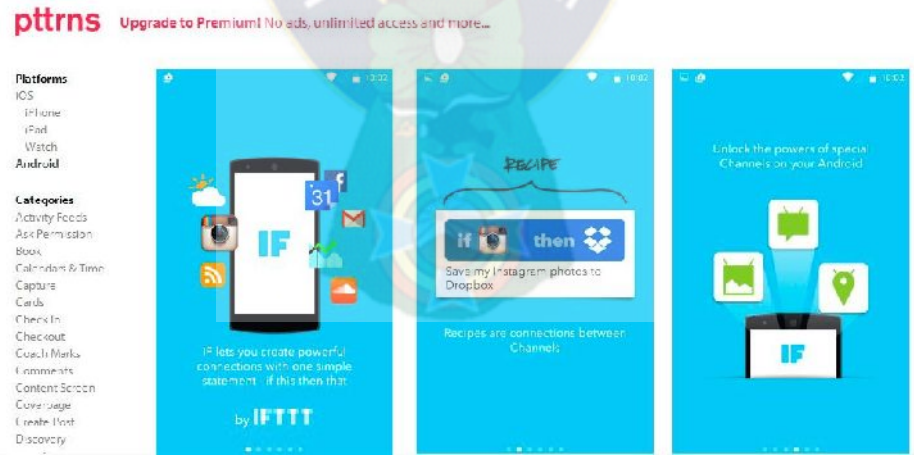


Figura 2.14: Pagina web de ejemplos de interfaces de usuario, pttrns.

Fuente: https://pttrns.com, 8 de abril 2018

2.8.2 Lovelyui

Se trata de un repositorio de elementos de interfaz de usuario para dispositivos móviles.

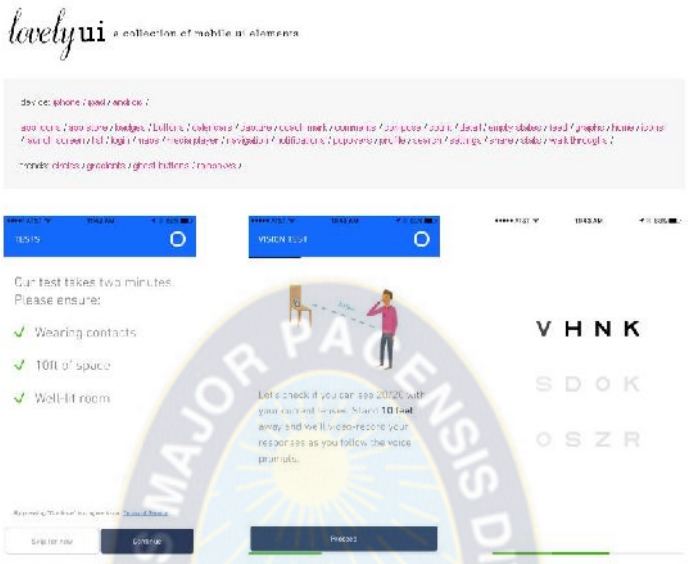


Figura 2.15: Pagina web de ejemplos de interfaces de usuario, lovelyui.

Fuente: https://www.lovelyui.com, 8 de abril 2018.

Similares a las anteriores páginas web se tienen las siguientes:

2.8.3 Mobile Patterns

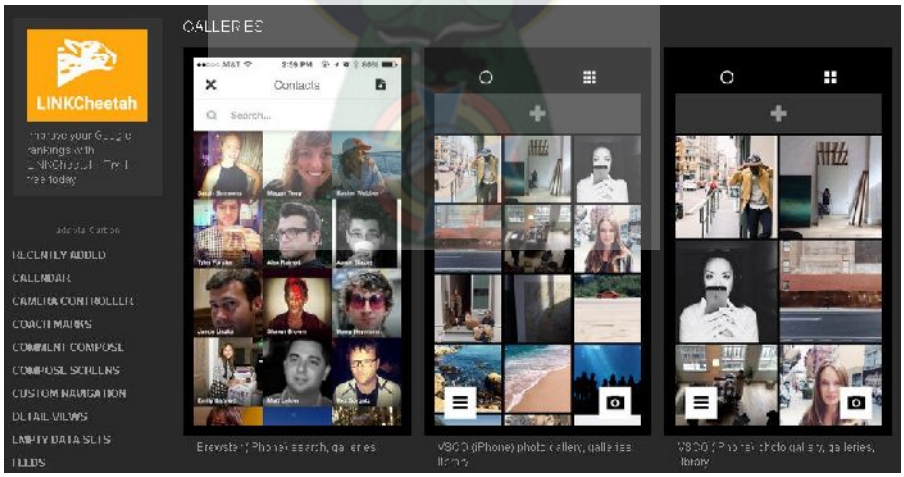


Figura 2.16: Pagina web de ejemplos de interfaces de usuario, Mobile Patterns.

Fuente: https://mobile-patterns.com, 8 de abril 2018.

2.8.4 Mobile Mozaic

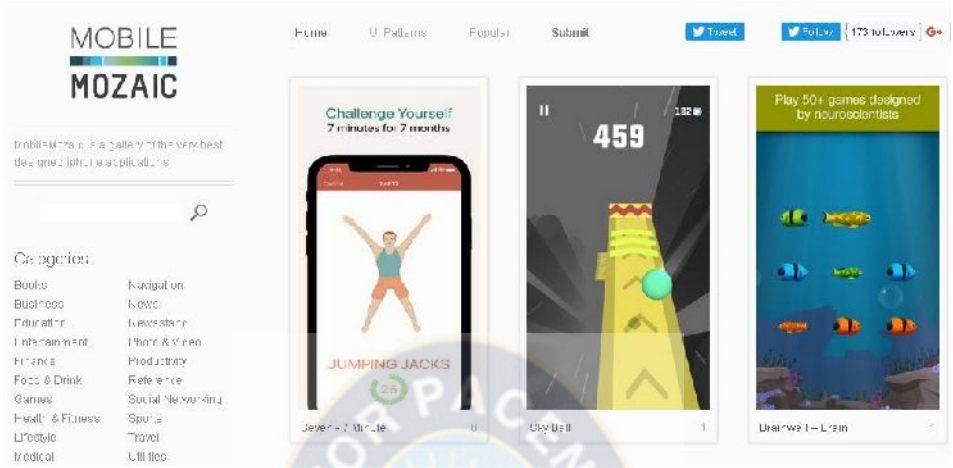


Figura 2.17: Pagina web de ejemplos de interfaces de usuario, Mobile Mozaic.

Fuente: https://www.mobilemozaic.com, 8 de abril 2018.

De las páginas web consultadas se tienen los siguientes *UI* para servir como referencia:

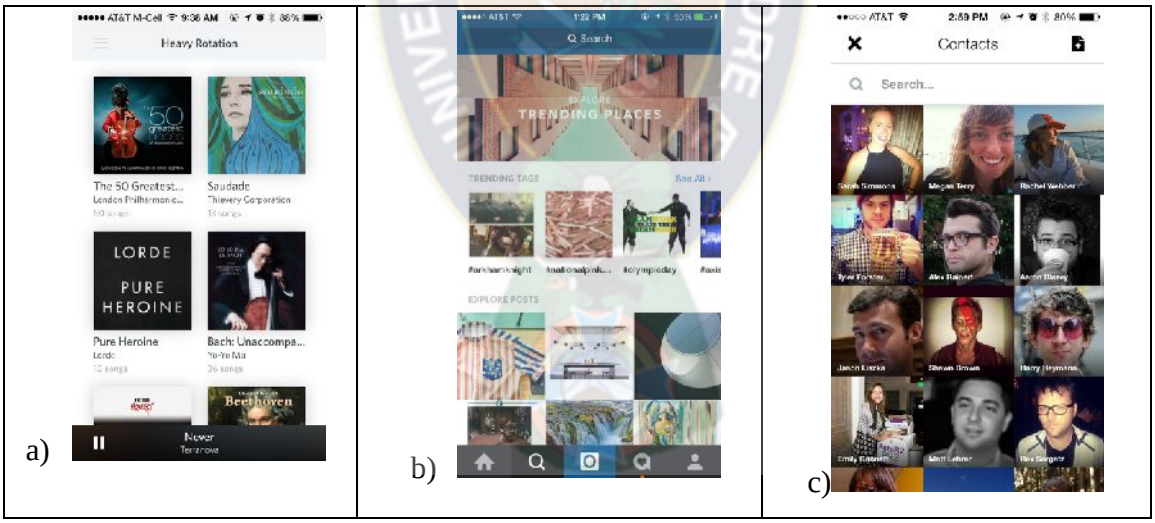


Figura 2.18: Interfaces de dispositivos móviles.

Fuente a): https://mobile-patterns.com, 8 de abril 2018.

Fuente b): https://mobile-patterns.com, 8 de abril 2018.

Fuente c): https://mobile-patterns.com, 8 de abril 2018.

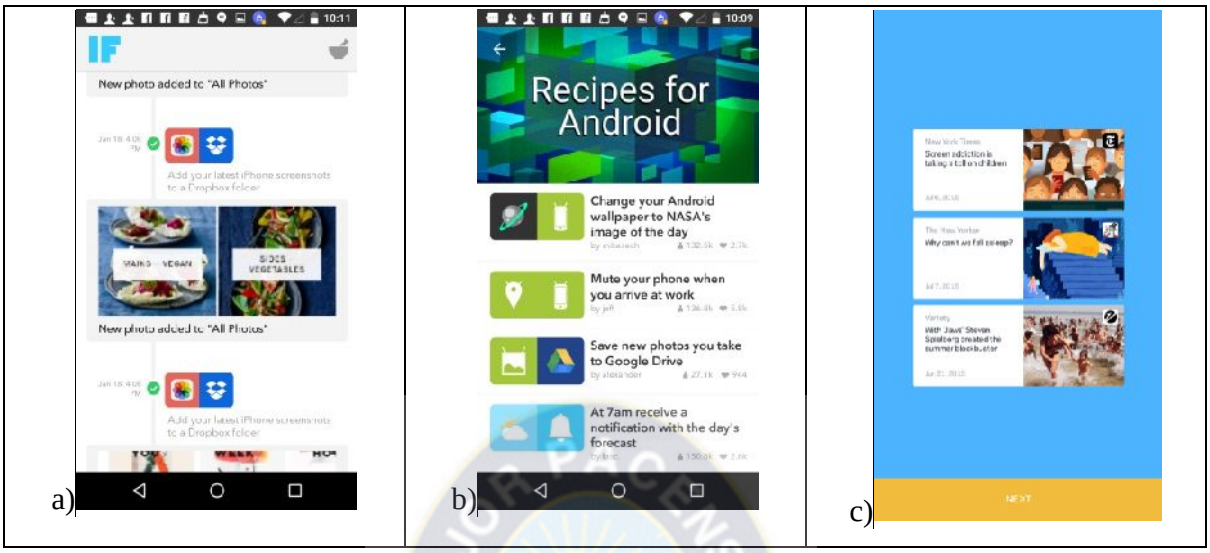


Figura 2.19: Interfaces de dispositivos móviles.

Fuente a): https://pttrns.com, 8 de abril 2018.

Fuente b): https://pttrns.com, 8 de abril 2018.

Fuente a): https://mobile-patterns.com, 8 de abril 2018.

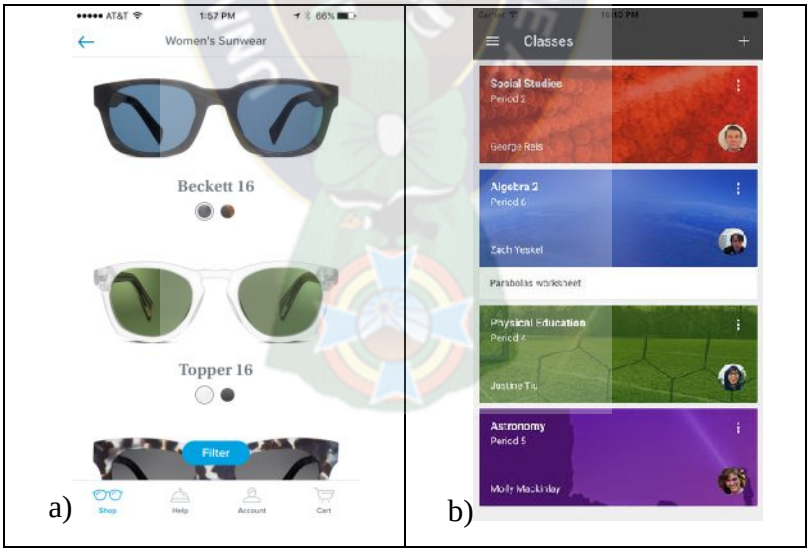


Figura 2.20: Interfaces de dispositivos móviles.

Fuente a): https://www.lovelyui.com, 8 de abril 2018.

Fuente b): https://www.mobilemozaic.com, 8 de abril 2018.

También existen soluciones de pago aunque con versiones de prueba.

2.8.5 Axure

Esta herramienta permite crear prototipos para aplicaciones en diferentes plataformas, además de la posibilidad de crear diagramas o mockups y permite trabajo cooperativo. Es una aplicación compleja que considera animaciones, funciones matemáticas, y contenido dinámico entre otros. Solo cuenta con una versión de escritorio.

2.8.6 Balsamiq

Esta herramienta posee versión web y versión de escritorio, ofrece retroalimentación, un diseño tipo borrador para alentar el brainstorming, extensiones de terceros, entre las características más relevantes. Cuenta entre sus clientes a empresas como: Sony, Adobe, Cisco, Ebay o EA. Tiene versión de pago y de prueba.

2.8.7 Mockingbird

Similar opción a las anteriores dos es mockingbird con trabajo colaborativo, enlace entre los diferentes mockups que se hayan creado y drag & drop para los elementos de interfaz de usuario que se quieran utilizar como características destacadas.

2.9 ESTILOS ACCESIBLES

El equipo de Google posee documentación referente a tamaños de interfaz, diseño responsivo, animaciones, color, etc. Para el presente trabajo presenta alta importancia el siguiente aspecto:

2.9.1 Tamaño mínimo de objetivo táctil

El tamaño mínimo de objetivo táctil recomendado es de alrededor de 48 píxeles independientes del dispositivo en un sitio con una ventana móvil establecida adecuadamente. Por ejemplo, a pesar de que un ícono puede solo tener un ancho y un alto de 24 px, puedes usar relleno adicional para llevar el tamaño de del objetivo de presión a hasta 48 px. El área de 48x48 píxeles corresponde a alrededor de 9 mm, que es alrededor del tamaño del área de la yema del dedo. (Google Developers, 2018).

Figura 2.21: Tamaño mínimo de objetivo táctil.

Fuente: https://developers.google.com/web/fundamentals/accessibility/accessible-styles?hl=es-, 3 de Junio de 2018.

Los objetivos táctiles también deberían tener un espacio de alrededor de 8 píxeles de separación, en orientación horizontal y vertical, de manera que el dedo del usuario, al presionar un objetivo de presión, no toque inintencionalmente otro objetivo de presión. (Google Developers, 2018).

2.10 TESTEO MANUAL EN DISPOSITIVOS MOVILES

Las pruebas manuales todavía tienen un papel importante para tomar parte, aun para organizaciones que exitosamente automatizan la mayor parte de sus tareas. No hay otro lugar donde la importancia del *manual testing* sea más evidente que en el contexto de escribir software para dispositivos móviles. (Monteiro, 2017).

Planear el testeo

La mayoría de las cosas comienza con un buen plan, y el testeo no es la excepción. El primer paso al preparar una testeo es planificar al los pasos que comprenderán la prueba, y su como ejecutarlos. (Monteiro, 2017).

Experimentar

Con el testeo planificado, ejecutarlo. Probar todo lo que se ha planeado, notar la ruta que se utilizo y como se alcanzo cada punto para evitar confusión cuando la prueba es corrida otra vez. Cuando un error aparece, intentar reproducir todos los pasos en la misma forma que se hizo la primera vez. Determinar qué causó el error. (Monteiro, 2017).

Anotar los errores

Registrar errores es necesario para reproducirlos otra vez. Después de solucionar cada uno de los problemas, ejecutar todas las pruebas para comprobar que nada se ha estropeado. (Monteiro, 2017).

Repetir

Se deben repetir todos los pasos otra vez después de realizar cualquier clase de cambio en el código, reproduciendo las pruebas de la misma forma desde el principio, y anotando cualquier cambios, reduciendo la tasa de error del testeo manual. (Monteiro, 2017).

Seleccionar dispositivos

Seleccionar diferentes dispositivos para realizar pruebas. Probar con diferentes marcas, plataformas, y sistemas operativos. Diferenciar cuando sea posible el hardware y software, comprobar la versión de software auxiliar, y anotar si hay diferencias entre actualizaciones. Y porque hay a menudo diferencias entre pantallas y la resolución, verificar si la aplicación es responsiva es obligatorio. (Monteiro, 2017).

CAPITULO III: MARCO APLICATIVO

3.1 APLICACION DE LA METODOLOGIA DE BRUNO MUNARI

3.1.1 Planificación de la metodología

Con sustentación de las palabras del autor de la metodología, Munari (1989): ”El esquema del método de proyección, ilustrado en páginas precedentes, no es un esquema fijo, o esta completo y no es único y definitivo” (p.30). Se aplicaron las adaptaciones necesarias para ajustarse a una investigación cuantitativa manteniendo todas las etapas originales. Se planteo la planificación:

Etapa	**Descripción**
Tipo de solución.	o Establecimiento del tipo de solución.
Elementos del problema.	o Descomposición del problema en sus elementos obteniendo subproblemas.
Recopilación de datos.	o Recopilación de datos para estudiar cada subproblema.
Análisis de datos.	o Análisis de los datos recopilados. o Eliminación de los valores estéticos. o Consideración de los valores técnicos.
Creatividad.	o Planteamiento de una solución al problema dentro de los limites y considerando el análisis de datos.
Materiales y Tecnologías.	o Recopilación de datos referidos a los materiales y tecnologías a disposición.
Experimentación de materiales y	o Experimentación con los materiales y tecnologías

tecnologías.	disponibles.
Modelos.	o Obtención de modelos mediante la experimentación de materiales y tecnologías.
Verificación.	o Evaluación del prototipo de maquetación. o Evaluación de las interfaces obtenidas realizando pruebas de: o Organización de la pantalla. o Alineación.

Tabla 3.2: Planificación de metodología.

3.1.2 Tipo de solución

Con el problema planteado en el capitulo uno se procedió al establecimiento del tipo de solución:

Tomando en consideración los distintos tipos de soluciones, se opto por la *Solución Sencilla y Económica* porque este tipo de solución cubriría con la maquetación de componentes de interfaz.

3.1.3 Elementos del problema

Ya que esta etapa de la metodología proviene del método cartesiano se descompuso el problema en tantas partes como fue necesario para su comprensión descubriendo subproblemas.

Se descompuso el problema principal y se identificaron los siguientes elementos:

¿Como desarrollar aplicaciones móviles?

- ¿Que plataforma de desarrollo se utilizara?
- ¿Que lenguaje de programación se utilizara?
- ¿Que colores utilizara el prototipo?

¿Cuales son las normas de maquetación necesarias para obtener un aspecto profesional?

- ¿Sera necesaria toda la teoría de sistemas de retículas?
- ¿Aceptara cualquier tipo de retículas?
- ¿Que diferentes resoluciones de pantalla se consideraran?

¿Como se crearan los componentes?

- ¿Que tipo de componentes debe soportar?
  - ¿Existe un límite para la adición de componentes de interfaz?
- ¿Cual será el tamaño mínimo de los componentes de interfaz?
- ¿Cual será el tamaño máximo de los componentes de interfaz?

Figura 3.1: División del problema principal en subproblemas.

3.1.4 Recopilación de datos

Se indagaron en los métodos de recopilación de datos: cuestionarios, entrevistas y encuestas. Ninguno de estos métodos se ajustaba a la labor de recolectar datos de páginas y servicios web ya que estas metodologías requerían una interacción entrevistador-entrevistado.

Considerando los subproblemas obtenidos se recopilo diferente documentación en el *Marco Teórico* con el objetivo de dar una respuesta a los mismos.

3.1.5 Análisis de datos

Se procedió a seleccionar la información relevante de toda la documentación recopilada para darle respuesta a los subproblemas.

a) Respecto a la plataforma de desarrollo

Unity represento la mejor opción debido a las siguientes razones:

- ✓ Su carácter gratuito.
- ✓ C# como lenguaje de programación.
- ✓ Agrupación de elementos de interfaz personalizados dentro de los *Canvas*.

- ✓ Alto rango de creación y edición de componentes de interfaz mediante un sistema de *Canvas.*
- ✓ Rápida ejecución de las aplicaciones nativas Android.

b) Respecto al sistema de retículas

Se singularizaron las siguientes *normas de diagramación* aplicables a la creación de interfaces de usuario en *dispositivos móviles*:

Norma	*Descripción*
1	Dado que el soporte físico de las pantallas tiene volumen físico, las retículas pueden no tener margen.
2	Considerando el tamaño reducido, no existen marcadores en las retículas.
3	La *Retícula de manuscrito* representa un caso inexistente en dispositivos móviles, este sistema considera el caso de ver dos páginas al mismo tiempo lo que representarían dos pantallas a la vez.
4	Una *Retícula de columnas* puede existir en una variación generalizada de 1 a 3 columnas por retícula.
5	Una *Retícula de columnas* puede ser simétrica o asimétrica.
6	En una *Retícula modular* los módulos pueden ser de proporción horizontal o vertical.
7	En una *Retícula modular* los módulos pueden unirse definiendo zonas espaciales para presentar información más relevante.
8	Una *Retícula jerárquica* no se basa en tamaños regulares porque se adapta al tamaño de sus elementos.
9	Exceptuando el caso de una *Retícula de columnas* con una única columna, todas las líneas de flujo son horizontales.

Tabla 3.2 Normas de diagramación aplicables a *UI* en dispositivos móviles.

Se aplicaron los sistemas de retículas documentados a diferentes UI de dispositivos móviles tratando de hallar un sistema reticular e identificando las normas que se cumplen en cada uno. Se trazaron guías y se obtuvieron los siguientes resultados:

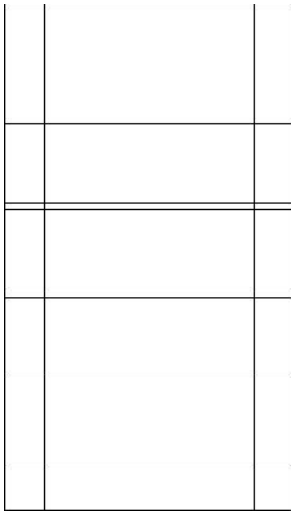
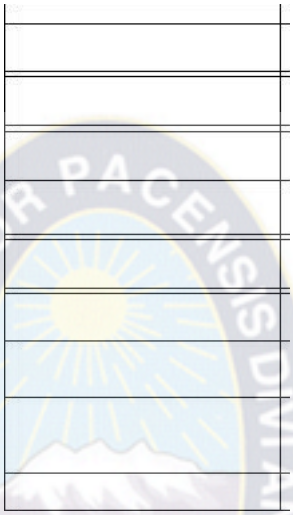
a)	b)	
		a) Retícula de columnas: una única columna (Norma 4) y simétrica (Norma 5). b) Retícula de columnas: una única columna (Norma 4) y simétrica (Norma 5).

Figura 3.2: Retículas de columnas.

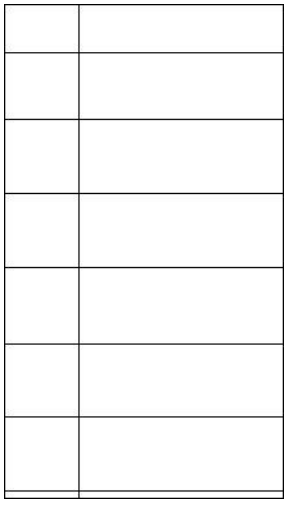
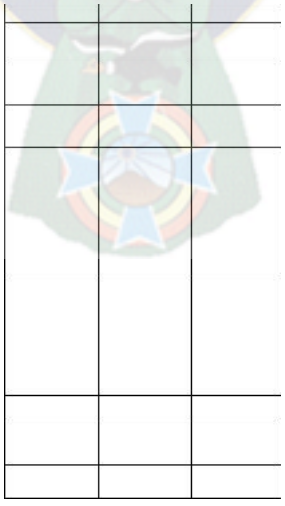
a)	b)	
		a) *Retícula modular* de proporción horizontal (Norma 6) y *jerárquica* (Norma 8). b) *Retícula modular* de proporción vertical (Norma 6) y *jerárquica* (Norma 8).

Figura 3.3: Retículas modulares y jerárquicas.

c) Respecto al tono y el color en el diseño de interfaz de usuario

El proceso de obtener diferentes colores puros para aplicarlos a UI es un proceso que requiere de un estudio complejo, ya sea orientando estos colores a provocar un estado de animo predeterminado para quienes los contemplan o experimentando con contrastes que indiquen una jerarquía de elementos de interfaz. Dicho estudio esta fuera del alcance del presente trabajo. Sin embargo, se aplicó la teoría para obtener una única paleta de tonos grises aplicable en caso de necesidad:

Se inicio con los siguientes valores:	Se incremento la saturación y disminuyo el brillo:	Se disminuyo la saturación y incremento el brillo:
Hue Matiz: 0	Hue Matiz: 0	Hue Matiz: 0
Saturation: 0 %	Saturation: 23 %	Saturation: 0
Brightness: 21 %	Brightness: 10 %	Brightness: 40 %

Figura 3.4: Paleta de tonos grises.

d) Respecto a las resoluciones de diferentes dispositivos

Con el conocimiento de que existen variadas resoluciones en dispositivos móviles, se proyecto el prototipo de manera que el mismo pueda reconocer el tamaño de la pantalla donde es ejecutado y en base a esa resolución realizar el despliegue de componentes de interfaz. Sin embargo al disponer solo de dos dispositivos con diferentes resoluciones para las pruebas en smartphones reales se atendió a estas resoluciones que son bastante regulares: QHD y HVGA

e) Respecto a las páginas y servicios web de diagramación de aplicaciones

De las páginas web Pttrns, Lovelyui, Mobile Pattern y Mobile Mozaic se obtuvieron los diferentes ejemplos de UI para dispositivos móviles y se identifico el tipo de retículas empleadas en el diseño de los mismos en el punto c).

3.1.6 Creatividad

Se proyecto un prototipo ejecutable en dispositivos Android que hace uso de sistemas de retículas con una única paleta de colores de tonos grises con las siguientes especificaciones:

Capaz de instanciar botones, imágenes, textos y cajas de entrada de texto.

Instanciaría componentes normales y componentes importantes.
Los componentes importantes tendrían mayor tamaño.

Soportar dos tipos de gestos: presión y presión larga.
Mediante el gesto de presión se añadiría un componente de interfaz normal.
Mediante el gesto de presión larga se añadiría un componente de interfaz importante.

La medida base para la creación del sistema de retículas empleado seria el intervalo comprendido entre 9mm y 11mm que representa las medidas mínimas asignables al alto y ancho ocupado por un botón que debe ser presionado con un dedo de la mano, en dispositivos móviles.

Las dimensiones mínimas para cualquier componente de interfaz son las mismas dimensiones de un modulo.

La manera de crear UI es innumerable pero se reconocen los siguientes casos generalizados:
Al instanciar componentes normales:
Máximo tres componentes por fila.
Si existe un solo componente en la fila, debe ocupar todo el ancho.

Si existen dos componentes en la fila deben ocupar cada uno la mitad.

Si existen tres componentes en la fila deben ocupar cada uno un tercio

Las dimensiones mínimas de un modulo son de X * X. Donde X = {x| x>=9mm, x<=11mm}.

Una vez asignadas las dimensiones mínimas de un módulo estas son constantes.

Cada componente tiene una altura mínima igual a la altura de un modulo excepto por el componente tipo texto que mínimamente tiene el alto igual la mitad del alto de un módulo.

Al instanciar componentes importantes:

Una imagen importante mínimamente tiene una altura y un ancho igual doble del alto de un módulo.

Un botón importante mínimamente tiene el ancho igual doble del alto de un módulo y el alto igual al alto de un módulo.

Un texto importante mínimamente tiene el ancho igual doble del alto de un módulo y el alto igual a la mitad de un módulo.

Si en una misma fila existe un componente importante y dos normales, los componentes normales ocupan cada uno el alto de un modulo a lo ancho y el componente importante el resto.

Si en una misma fila existen dos componente importantes y uno normal, el componente normal ocupa el alto de un modulo a lo ancho y los componentes importantes ocupan el resto del ancho entre dos.

Si en una misma fila todos los componentes son importantes cada uno ocupa el mismo espacio a lo ancho, un tercio.

Al instanciar componentes tipo cabecera y pie de página:

Una fila de componentes que representen el equivalente de una cabecera de una página web deben ubicarse en la parte superior de la pantalla, estos componentes pueden ser caja de entrada de texto, imagen, botón o texto con las siguientes restricciones:

El alto de los componentes dentro de esta fila es el mismo alto de un módulo.

Máximo tres componentes en esta fila.

Si existe una caja de entrada de texto es la única de ese tipo y de tamaño preferente (importante).

Una fila de componentes que representen el equivalente de un pie de página de una página web deben ubicarse en la parte inferior de la pantalla estos componentes pueden ser botones o texto con las siguientes restricciones:

El alto de los componentes dentro de esta fila es el mismo alto de un módulo.

Si existe un texto es el único de ese tipo.

Se pueden instanciar hasta cinco botones

Pueden existir el componente tipo cabecera y el tipo pie de página al mismo tiempo, existir solo uno de ellos o no existir ninguno.

3.1.7 Materiales y Tecnologías

Los datos recopilados referentes a materiales y tecnologías se encuentran en el *Marco Teórico* eligiendo a Unity como plataforma de desarrollo y a C# como lenguaje de programación, representando ambos a las *Tecnologías* a ser empleadas en la metodología. Los *Materiales* son representados el equipo de computación a ser utilizado: un ordenador PC, un teclado, un *mouse* y un monitor.

3.1.8 Experimentación con los materiales y tecnologías

Según la solución planteada se utilizó Unity y C# para alcanzar los objetivos descritos en *Creatividad* obteniendo los siguientes resultados:

Se desarrollaron cuatro clases importantes:

- ✓ Una clase que controla variables globales.
- ✓ Una clase que controla componentes instanciados tipo cabecera.
- ✓ Una clase que controla componentes instanciados tipo pie de página.
- ✓ Una clase que controla componentes instanciados por filas.

El código completo de los programas descritos en los siguientes incisos puede ser consultado en *Anexo A*.

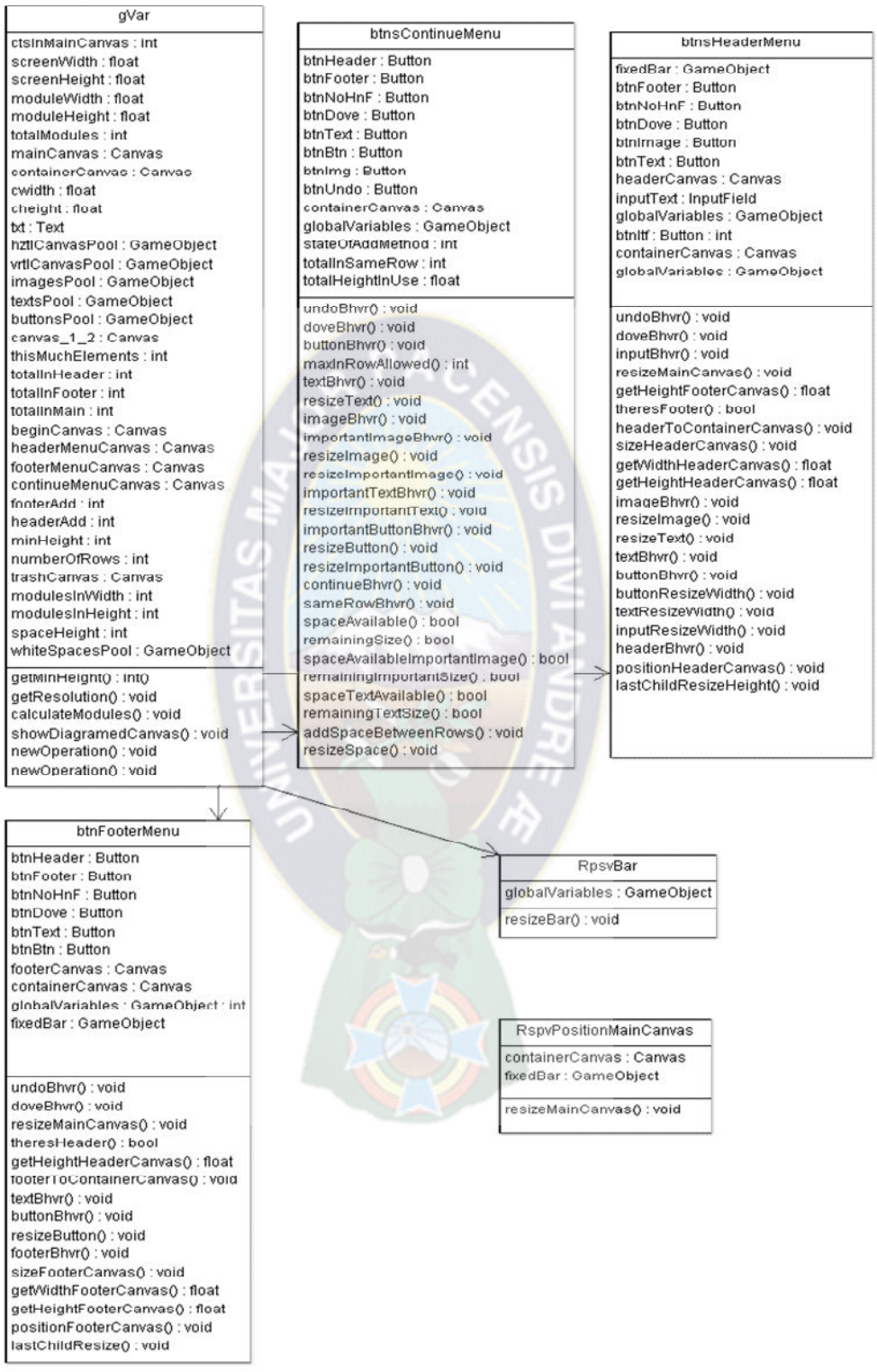


Figura 3.5: Diagrama de clases del prototipo de maquetación.

a) Una clase que controla variables globales (gVar)

Esta clase controla variables necesarias para las demás clases como el total de elementos en la cabecera, en el pie de página, en la retícula principal, también fue útil durante el desarrollo al desplegar información en el entorno de Unity. Como programas principales dentro de esta clase se tienen los siguientes:

```
public void getResolution()
        {
                screenWidth =
containerCanvas.GetComponent<RectTransform>().rect.width;
                screenHeight =
containerCanvas.GetComponent<RectTransform>().rect.height;
        }
```

Programa 1: Obtención de la resolución del dispositivo.

Que recupera la resolución de la pantalla del dispositivo en pixeles en las variables screenWidth y screenHeight.

```
public void calculateModules()
{
moduleWidth = minHeight;
moduleHeight = minHeight;
modulesInWidth= (int)
mainCanvas.GetComponent<RectTransform>().rect.width/minHeight;
modulesInHeight = (int)
mainCanvas.GetComponent<RectTransform>().rect.height/minHeight;
totalModules = modulesInWidth*modulesInHeight;
}
```

Programa 2: Cálculo de módulos.

Este programa asigna las dimensiones de los módulos, calcula cuantos existen a lo ancho y a lo alto y obtiene el total de módulos.

b) Una clase que controla componentes instanciados tipo cabecera.

Esta clase controla la instanciación de componentes en la parte superior de la pantalla, los programas principales de esa clase son:

```
public void imageBhvr()
    {
    if (globalVariables.GetComponent<gVar>().totalInHeader < 3)
        {

            globalVariables.GetComponent<gVar>
            ().imagesPool.transform.GetChild (0).
                transform.SetParent (headerCanvas.transform);
            headerToContainerCanvas();
            positionHeaderCanvas ();
            sizeHeaderCanvas ();
            lastChildResize ();
            globalVariables.GetComponent<gVar>().totalInHeader++;
            globalVariables.GetComponent<gVar>().txt.
                GetComponent<Text>().text = ">Instanciada Imagen.";
            globalVariables.GetComponent<gVar> ().headerAdd = 1;
            resizeMainCanvas();
        }
    }
```

Programa 3: Comportamiento del botón de instanciación de imagen.

headerToContainerCanvas(): Convierte a el *Canvas* de tipo cabecera en hijo del *Canvas* principal.

positionHeaderCanvas (): Asigna las coordenadas requeridas para la correcta ubicación en la parte superior de la pantalla.

sizeHeaderCanvas (): Asigna el alto y el ancho a el *Canvas* tipo cabecera.

lastChildResize (): Asigna el ancho necesario a el último hijo añadido a el *Canvas* tipo cabecera.

resizeMainCanvas(): Es en el *mainCanvas* donde se añaden elementos por filas, el tamaño del mismo debe ser redefinido si se crea un *Canvas* tipo cabecera o tipo pie de pagina, este método le resta al *mainCanvas* el tamaño de la cabecera si se creo.

Los métodos:

public void inputBhvr(),
public void textBhvr() y
public void buttonBhvr().

Tienen un comportamiento similar a *public void imageBhvr() descrito previamente.*

c) Una clase que controla componentes instanciados tipo pie de página

Esta clase controla la instanciación de componentes en la parte inferior de la pantalla, los métodos de esta clase son similares a los de la clase anterior.

d) Una clase que controla componentes instanciados por filas

La clase más importante, podría existir sin necesidad de las dos clases anteriores. Los programas más importantes son:

```
public void undoBhvr()
{
if(globalVariables.GetComponent<gVar>().mainCanvas.transform.childCount>0
        {
if(globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
                        globalVariables.GetComponent<gVar>().numberOfRows-1).
                transform.childCount>0)
                {
globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
                globalVariables.GetComponent<gVar>().numberOfRows-1)
                                transform.GetChild(totalInSameRow-1).
                                transform.SetParent(
globalVariables.GetComponent<gVar>().trashCanvas.transform);
                        totalInSameRow --;
                }
        }
}
```

Programa 4: Comportamiento del botón deshacer.

Este programa primero verifica que exista algún hijo en el *Canvas* principal, luego verifica si este hijo contiene a otros componentes y procede a eliminar el último componente añadido al enviarlo a un *Canvas* invisible que sirve de papelera de reciclaje.

```
if(globalVariables.GetComponent<gVar>().mainCanvas.transform.childCount>0)
        {
        if(globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
        globalVariables.GetComponent<gVar>().numberOfRows-1).
                        transform.childCount>0)
```

```
                {
                totalHeightInUse = totalHeightInUse +

globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
globalVariables.GetComponent<gVar>().numberOfRows-1).
GetComponent<RectTransform>().rect.height;
                }
        }
```

Programa 5: Control del espacio ocupado.

Estas líneas de código obtienen el espacio ocupado del total del alto del *Canvas* principal.

```
public void buttonBhvr()
    {
        if(spaceAvailable())
        {
            if(stateOfAddMethod == 1)
            {
                if (totalInSameRow >= maxInRowAllowed())
                {
                    globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Error.";
                }
                if (totalInSameRow < maxInRowAllowed())
                {
                    resizeButton();
                    globalVariables.GetComponent<gVar>
```

```
().buttonsPool.transform.GetChild (0).
                                        transform.SetParent (
        globalVariables.GetComponent<gVar> ().mainCanvas.transform.GetChild(
        globalVariables.GetComponent<gVar>    ().mainCanvas.transform.childCount-
1));
                                    totalInSameRow ++;
        globalVariables.GetComponent<gVar>().totalInMain++;
                                    globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text          =
">Instanciado boton.";
                                }
                            }
                }
                else
                        globalVariables.GetComponent<gVar>().txt.
                            GetComponent<Text>().text = ">No hay espacio libre.";
        }
```

Programa 6: Comportamiento del botón de instanciación de un componente *Botón.*

Este programa consta primero si existe espacio para añadir un componente de interfaz, si no se a alcanzado el limite de componentes por fila puede instanciar un *Botón*, primero verifica su tamaño mediante el método *resizeButton()*, luego lo añade como hijo en un *Canvas* que es a su vez hijo del *Canvas* principal. También puede desplegar un mensaje de verificación o de error.

```
public void importantButtonBhvr()
        {
                if(spaceAvailable())
                {
                        if(stateOfAddMethod == 1)
                        {
                                if (totalInSameRow >= maxInRowAllowed())
                                {
                                        globalVariables.GetComponent<gVar>().txt.
                                                GetComponent<Text>().text = ">Error.";
                                }
                                if (totalInSameRow < maxInRowAllowed())
                                {
                                        resizeImportantButton();
                                        globalVariables.GetComponent<gVar>
().buttonsPool.transform.GetChild (0).
                                                transform.SetParent (

        globalVariables.GetComponent<gVar> ().mainCanvas.transform.GetChild(

        globalVariables.GetComponent<gVar>    ().mainCanvas.transform.childCount-
1));
                                        totalInSameRow ++;
        globalVariables.GetComponent<gVar>().totalInMain++;
                                        globalVariables.GetComponent<gVar>().txt.
                                                GetComponent<Text>().text          =
">Instanciado boton importante.";
                                }}}}
```

Programa 7: Comportamiento del botón de instanciación de un componente *Botón Importante*.

Similar al anterior programa con una importante diferencia, el método *resizeImportantButton()* que asigna un tamaño mayor al componente botón instanciado en relación con cualquier otro componente de tamaño normal.

Los programas:

- ✓ public void imageBhvr().
- ✓ public void importantImageBhvr().
- ✓ public void textBhvr().
- ✓ public void importantTextBhvr().

Tienen un comportamiento equivalente a los programas 6 y 7.

3.1.9 Modelos

Los modelos son representados por los diferentes prototipos obtenidos con la herramienta de prototipado. Se obtuvieron ejecutando el prototipo un dispositivo con las siguientes características:

Modelo: HUAWEI G730.
Resolución: QHD, 540x960 pixeles.
Densidad: 240dpi.

Para tener algún sentido de coherencia en la creación de modelos se intentaron obtener diseños tipos de interfaces utilizadas a menudo y emular otras ya observadas en el presente trabajo:

Se reconocieron un par de interfaces generalizadas:

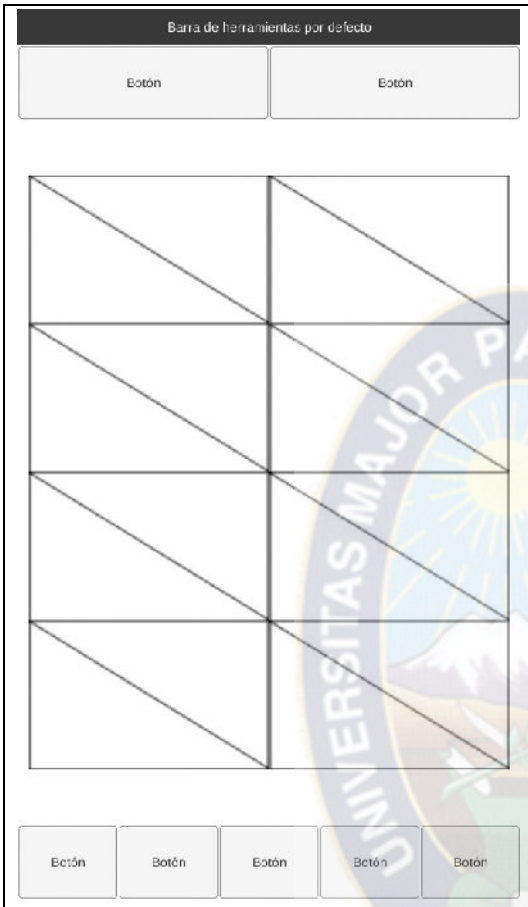


Figura 3.8: Modelo 1, Retícula para galería de imágenes.

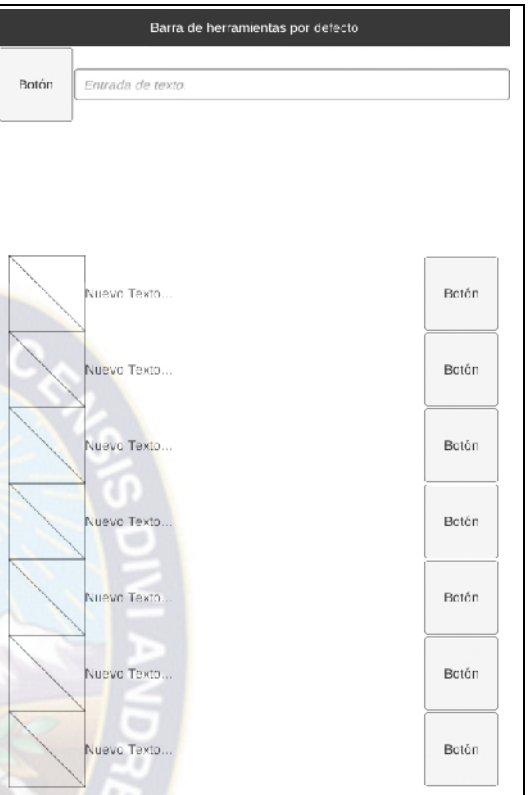


Figura 3.9: Modelo 2, Retícula de un menú tipo lista

Se emularon las siguientes interfaces:

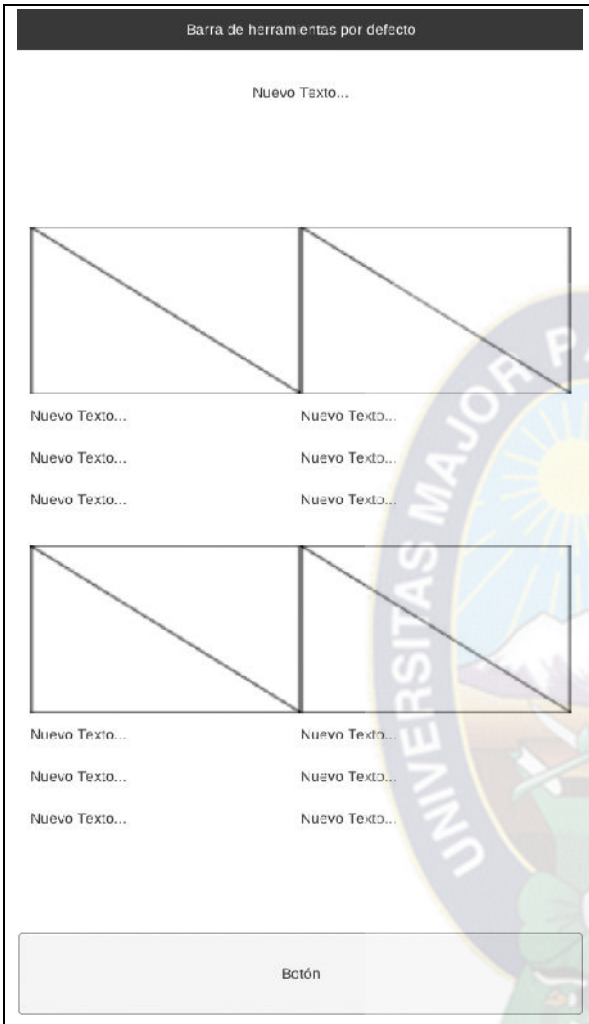


Figura 3.10: Modelo 3, Retícula emulada de la Figura 2.18 a).

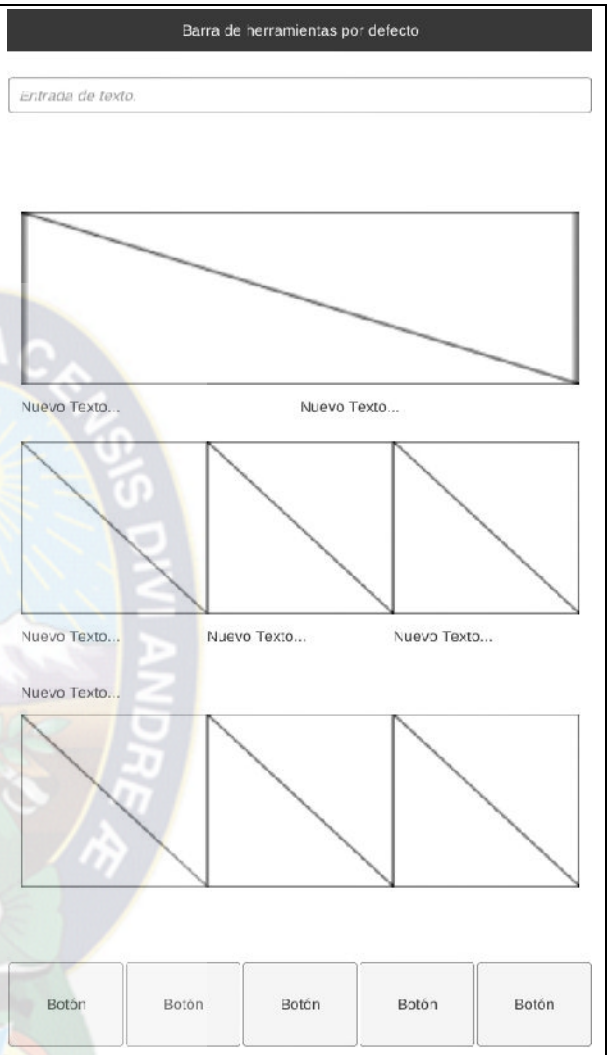


Figura 3.11: Modelo 4, Retícula emulada de la Figura 2.18 b).

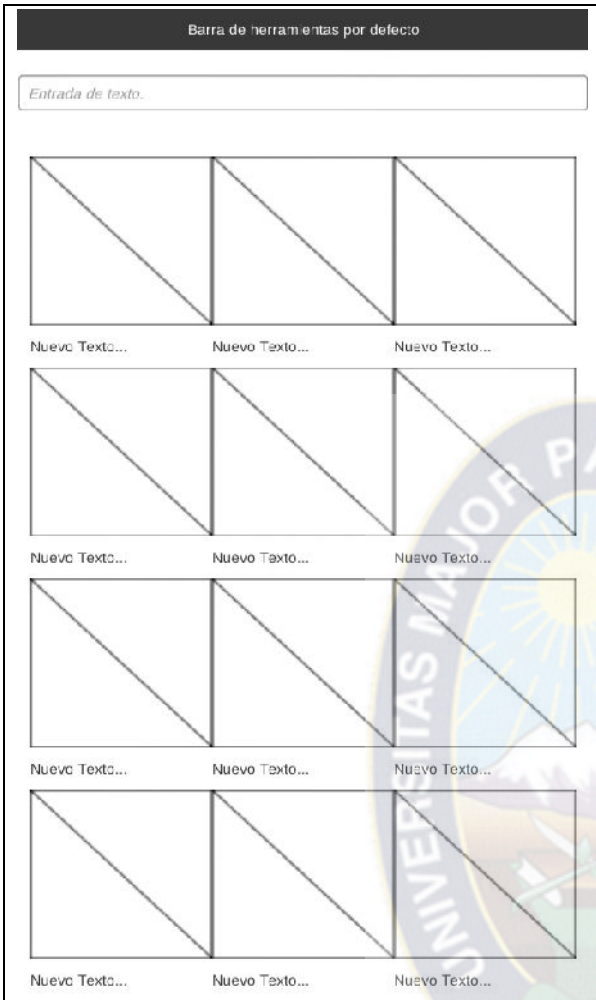


Figura 3.12: Modelo 5, Retícula emulada de la Figura 2.18 c).

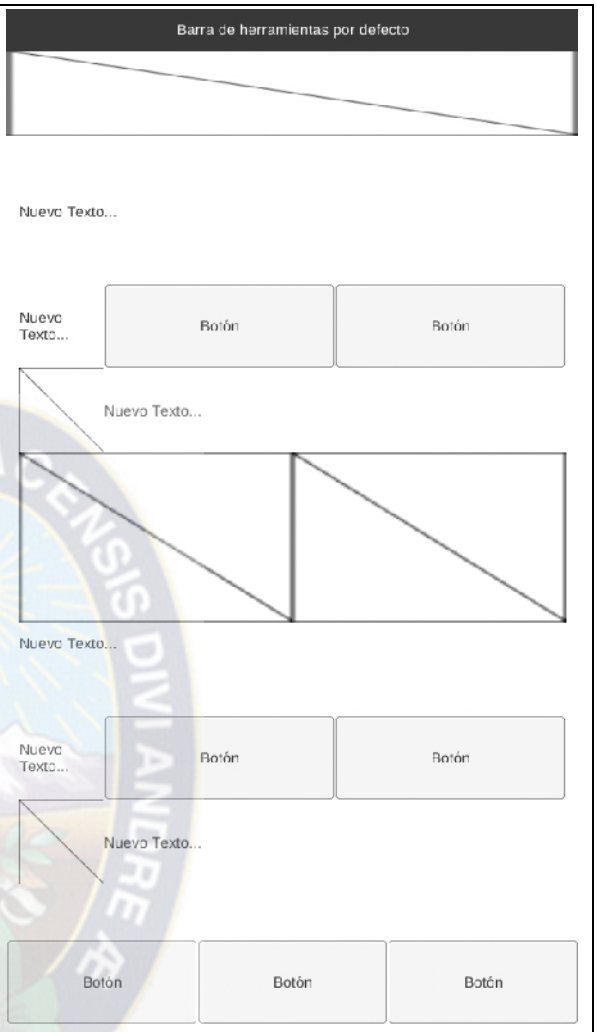


Figura 3.13: Modelo 6, Retícula emulada de la Figura 2.19 a).

.

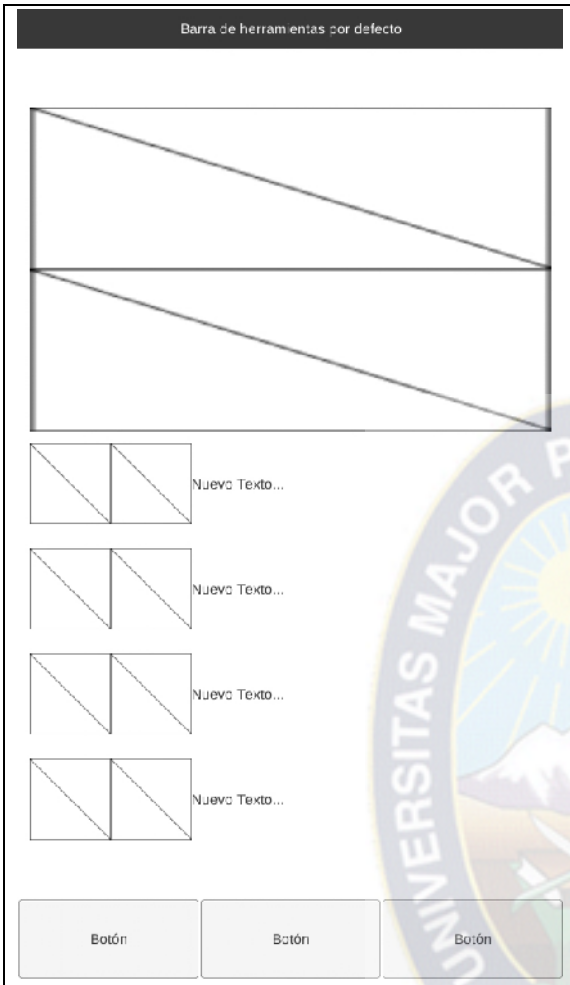


Figura 3.14: Modelo 7, Retícula emulada de la Figura 2.19 b).

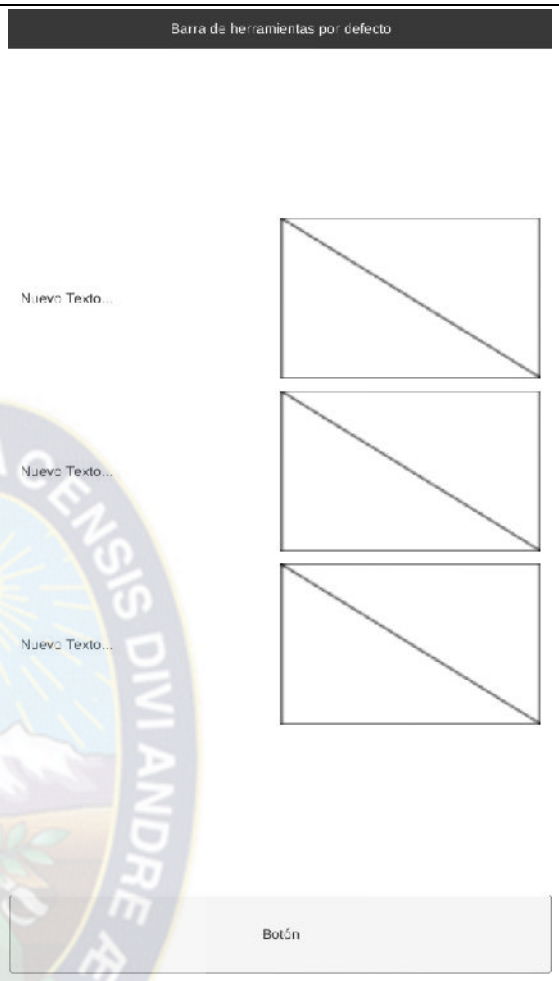


Figura 3.15: Modelo 8, Retícula emulada de la Figura 2.19 c).

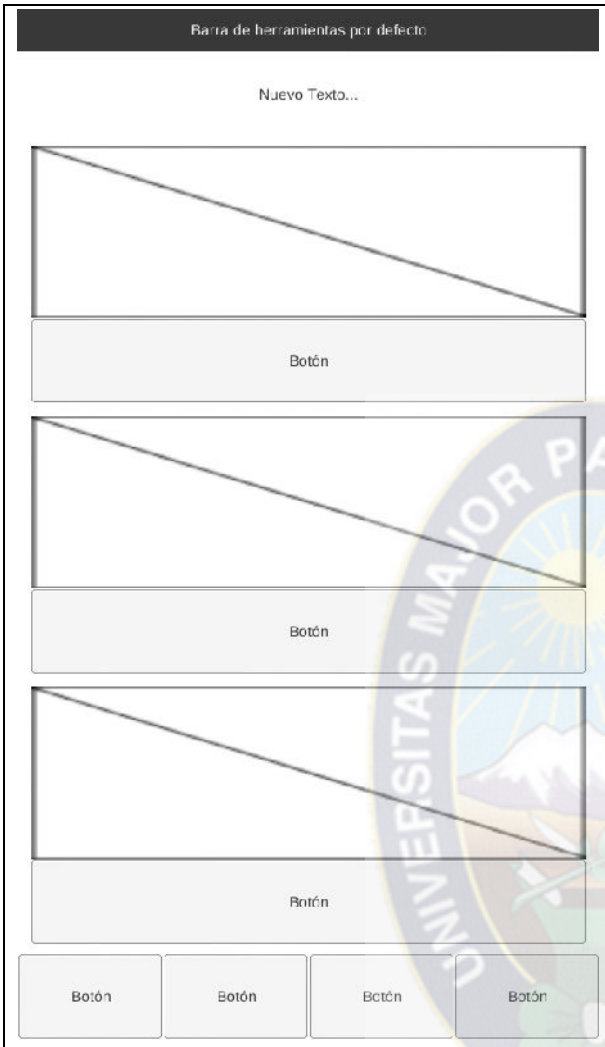


Figura 3.16: Modelo 9, Retícula emulada de la Figura 2.20 a).

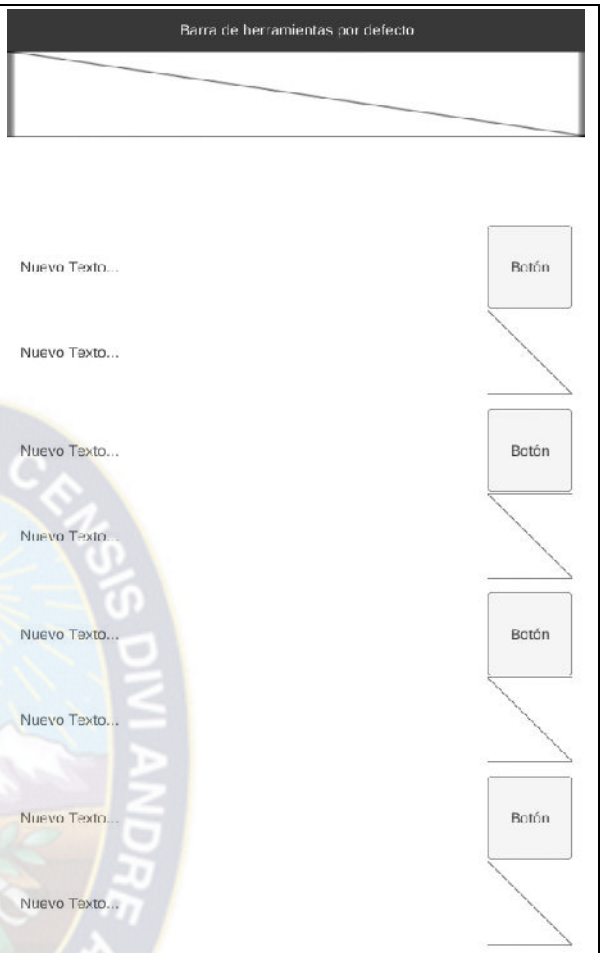


Figura 3.17: Modelo 10, Retícula emulada de la Figura 2.20 b).

3.1.10 Verificación

El objetivo principal fue la obtención de modelos mediante el uso de la herramienta de prototipado programada. Sin embargo, la herramienta en si debía someterse mínimamente a una verificación de funcionamiento. Los prototipos obtenidos mediante su aplicación debían cumplir con las normas de diagramación y con otros aspectos relacionados a interfaz de usuario en dispositivos móviles estudiados. Con base en el siguiente diagrama se propuso una tabla de verificación.

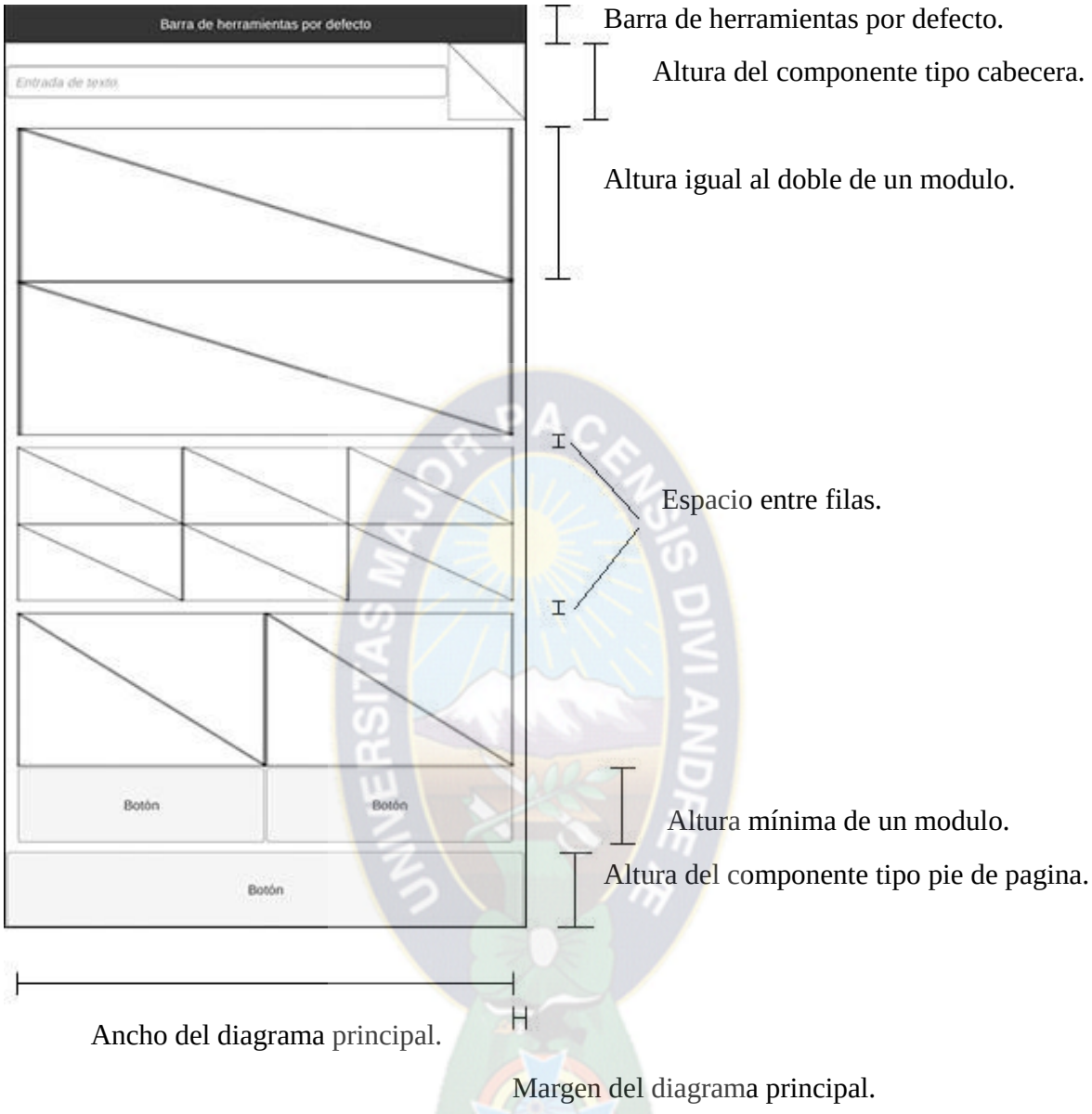


Figura 3.18: Diagrama para la verificación.

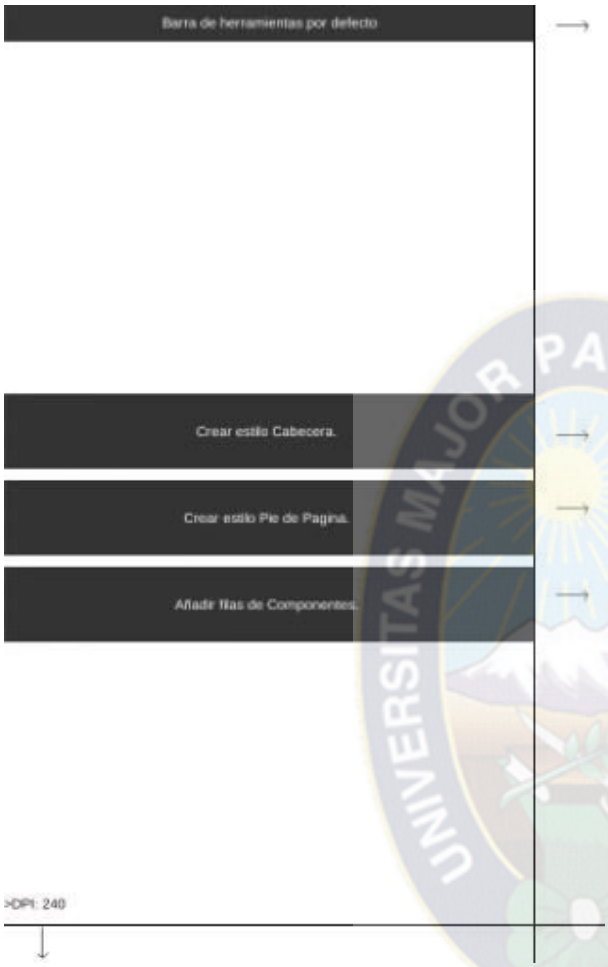
Se propuso la siguiente tabla de verificación de modelos de interfaces obtenidos a través de la utilización del prototipo de herramienta de maquetación basada en las *Normas de Diagramación* propuestas.

Regla	*Descripción*	*Justificación*
1	En las dimensiones de la retícula debe reconocerse un patrón constante o una medida base.	Cuando existe una retícula de columnas todas tienen el mismo ancho, en una retícula modular todos los módulos son del mismo tamaño, cuando existe una retícula jerárquica los tamaños varían en múltiplos y submúltiplos. También se presenta el caso de retículas compuestas donde se distinguen agrupaciones de tamaños.
2	Máximo tres componentes por fila.	Las interfaces analizadas en el marco teórico presentan un máximo de tres componentes por fila a excepción de las galerías que pueden tener cuatro componentes en la misma fila. Como deben existir restricciones generales se eligió el límite de tres.
3	El tamaño mínimo de los módulos es de 10 mm x 10mm y representa la medida base de la retícula.	Google Developers plantea un tamaño mínimo de alrededor de 9mm para elementos táctiles y una separación de 1.5mm entre ellos. Ya que entre componentes en la misma fila no existe espacio se propuso esa medida.
4	Un espacio entre filas mide 1.5mm.	Google Developers plantea una distancia mínima de alrededor de 1.5mm entre elementos táctiles.
5	El espacio a izquierda y derecha del diagrama principal es de 1.5mm.	Es la misma medida del espacio entre filas.
6	Un componente de texto ocupa menos espacio a lo alto.	Un submúltiplo del tamaño mínimo de un modulo es 5mm y representa la altura para los componentes de texto.
7	Los componentes normales ocupan espacios iguales	Los componentes en la misma fila varían de las siguiente manera: ➢ Un componente ocupa todo el ancho. ➢ Dos componentes ocupan la mitad del ancho cada

		uno. ➢ Tres componentes ocupan un tercio del ancho cada uno. Estas medidas garantizan un sistema de retículas constante entre filas.
8	Los componentes importantes son de mayor tamaño.	Los componentes en la misma fila varían de las siguiente manera: ➢ Un componente importante inicia ocupando todo el ancho, este ancho disminuye en tamaños de 10mm, la medida base, cada vez que se instancie un componente normal. ➢ Dos componentes importantes en la misma fila pueden ocupar la mitad del ancho cada uno o la mitad del ancho menos la medida base (10mm) cada uno si existe un tercer componente. ➢ Tres componentes importantes ocupan un tercio del ancho cada uno. Estas medidas garantizan un sistema de retículas constante entre filas.
9	Un estilo pie de página y un estilo cabecera se anclan en la parte inferior y superior de la pantalla respectivamente.	Comportamiento heredado de las páginas web.
10	Un diagrama principal puede existir sin cabecera o sin pie de página.	Cabecera y pie de página son componentes complementarios al diagrama principal y pueden ser prescindibles.

Tabla 3.3: Reglas para la verificación de retículas.

Para la verificación de la herramienta de maquetación se tienen los siguientes diagramas guía:



Barra de herramientas.

Botón Crear estilo *Cabecera*.

Botón Crear estilo *Pie de Página*.

Botón *Añadir filas de Componentes.*

Espacio para mensajes de comportamiento.

Figura 3.19: Menú inicial de la aplicación.

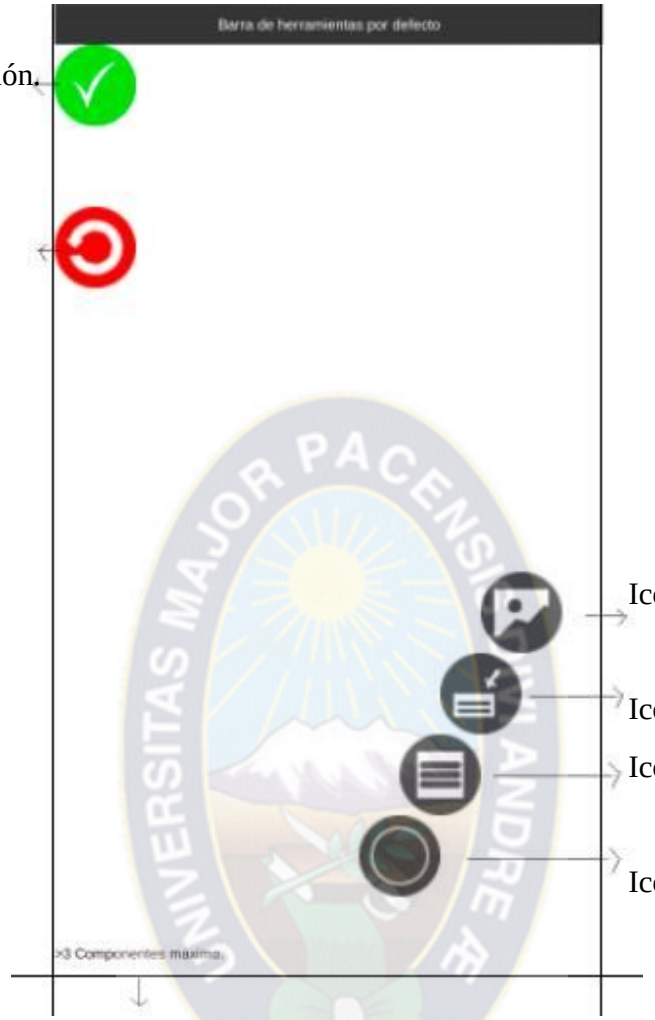


Figura 3.20: Menú de creación de cabecera.

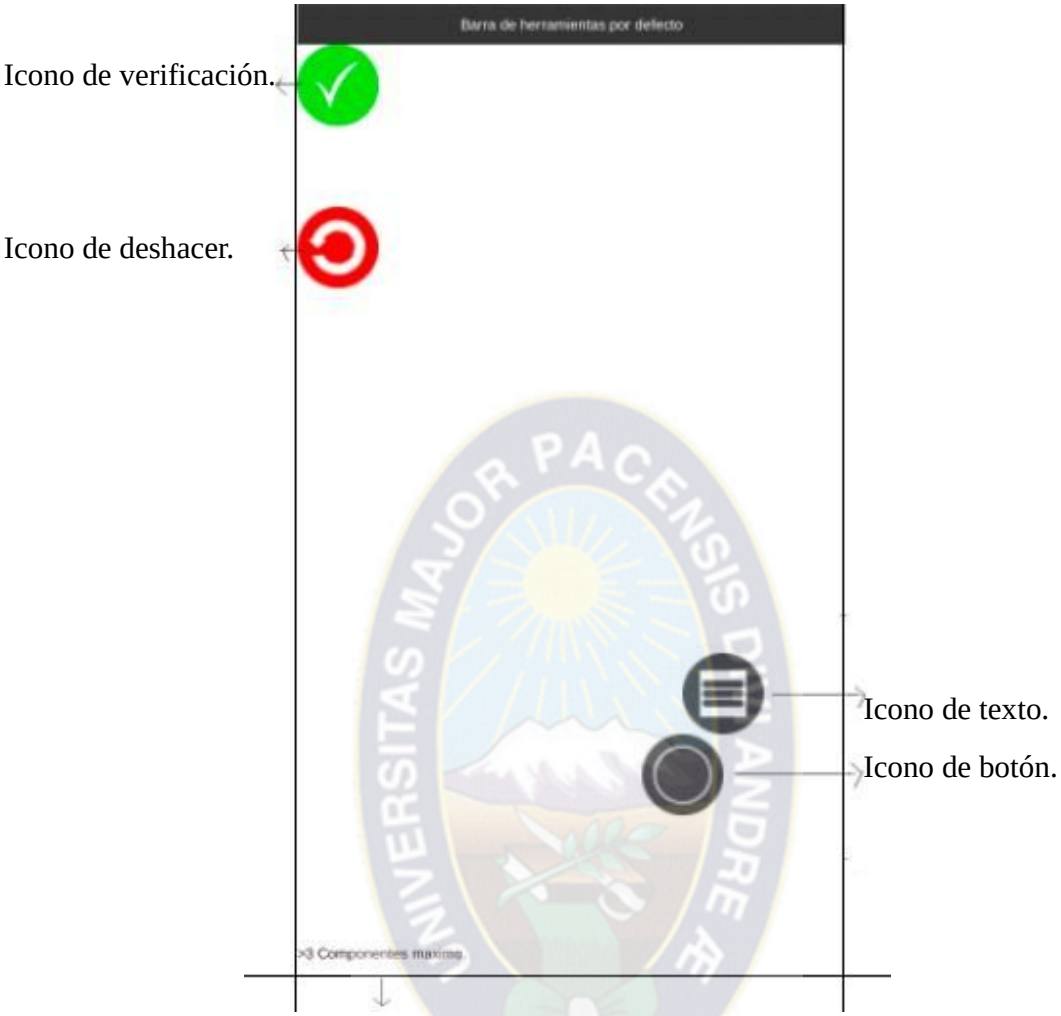


Figura 3.21: Menú de creación de pie de pagina.

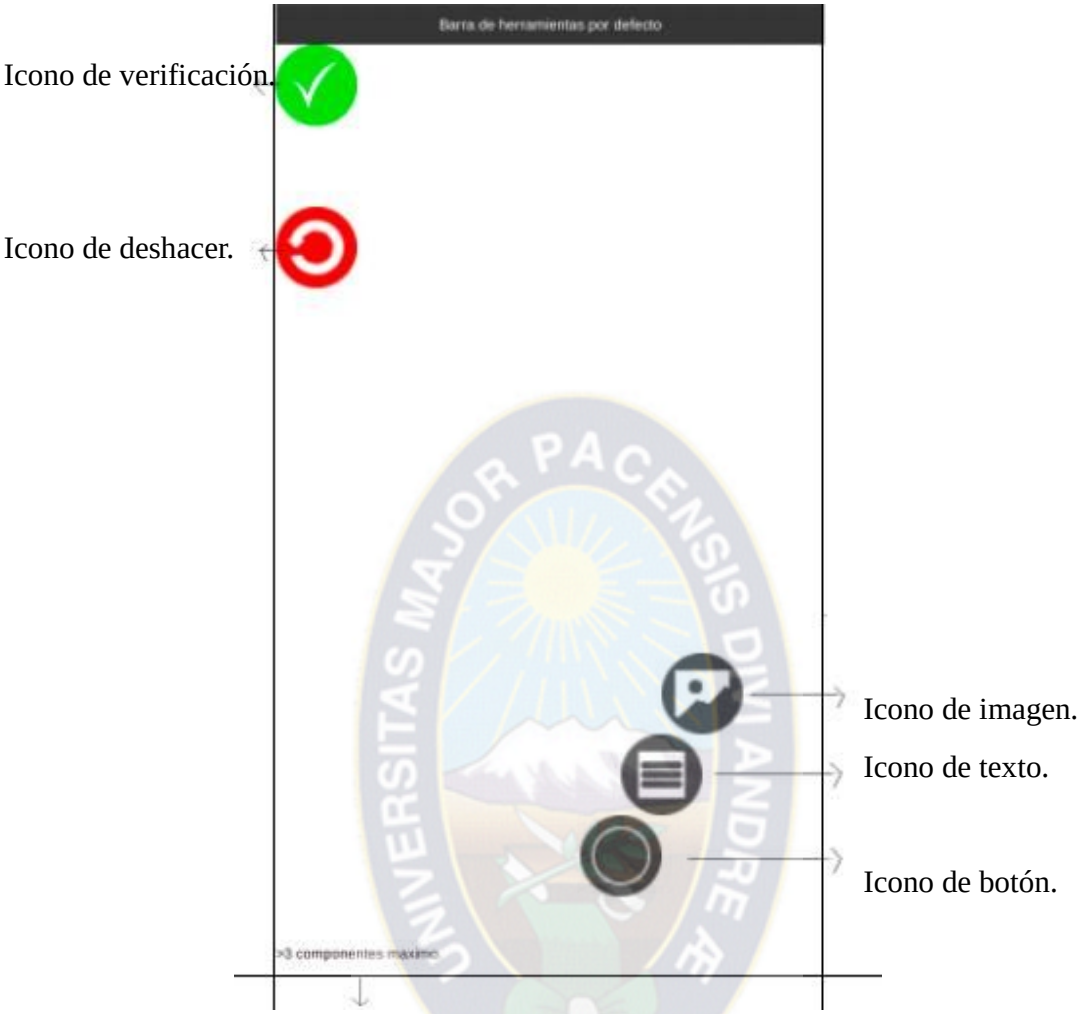


Figura 3.22: Menú de creación de filas de componentes.

a) Herramienta de maquetación

La herramienta de prototipado fue una aplicación funcional (no un prototipo) por tanto debía someterse a una verificación de funcionamiento contemplada dentro del *Mobile Testing*.

Se procedió a ejecutar un *Native Mobile App Testing* (Testeo de Aplicación Móvil Nativa) manual en un dispositivo HUAWEI G730, QHD y con 240dpi.:

Nro.	*Precondición*	*Objetivo*	*Descripción del paso ejecutado.*	*Resultado esperado*	*Resultado de la ejecución*
1	Aplicación instalada.	Verificar el arranque del programa.	Se inicio la aplicación.	El menú inicial debe ser visible.	Éxito.
		Verificar el escalamiento a lo ancho del menú inicial.	Se inicio la aplicación.	Los botones iniciales se ajustan al ancho de la pantalla.	Éxito.
		Verificar el dpi de la pantalla.	Se inicio la aplicación.	Se muestra un mensaje con el dpi actual.	Éxito.
		Verificar el posicionamiento y tamaño de la barra de herramientas por defecto.	Se inicio la aplicación.	La barra de herramientas se sitúa el la parte superior, tiene un ancho igual al de la pantalla y una altura de 5mm.	Éxito.
2	Menú inicial	Verificar el	Se presiono la	El menú de	Éxito.

	visible.	despliegue del menú de creación de cabecera.	opción *Crear estilo Cabecera.*	instanciación y verificación correspondiente es visible.	
		Verificar el despliegue del menú de creación de pie de pagina.	Se presiono la opción *Crear estilo Pie de Pagina.*	El menú de instanciación y verificación correspondiente es visible.	Éxito.
		Verificar el despliegue del menú de instanciación de componentes en filas.	Se presiono la opción *Añadir filas de Componentes.*	El menú de instanciación y verificación correspondiente es visible.	Éxito.
3	Cualquier menú de instanciación y verificación es visible.	Verificar la instanciación de un componente *Imagen*.	Se presiono sobre el icono de imagen.	Se instancio un componente *Imagen* en la cabecera.	Éxito.
		Verificar la instanciación de un componente *Entra de de texto*.	Se presiono sobre el icono de entrada de texto	Se instancio un componente *Entrada de texto* en la cabecera.	Éxito.
		Verificar la instanciación de un componente *Texto*.	Se presiono sobre el icono de texto.	Se instancio un componente *Texto* en la cabecera.	Éxito.
		Verificar la instanciación de un componente *Botón*.	Se presiono sobre el icono de botón.	Se instancio un componente *Botón* en la	Éxito.

				cabecera.	
		Verificar la confirmación de fila añadida.	Se presiono sobre el icono de confirmación.	El menú inicial es visible.	Éxito.
		Verificar la acción de deshacer.	Se presiono sobre el icono de deshacer.	El ultimo componente instanciado es eliminado.	Éxito.
		Verificar la acción de añadido de fila con margen inferior.	Se ejecuto un gesto *longpress* sobre el icono de confirmación.	La ultima fila se añade con un margen inferior.	Éxito.
		Verificar la acción de mostrar la maquetación final.	Se ejecuto un gesto *longpress* sobre el icono de deshacer.	Solo es visible la maquetación creada, los menús ya no son accesibles	Éxito.
4	El menú de instanciación y verificación de *Añadir filas de Componentes* es visible.	Verificar la acción de instanciar un componente importante.	Se ejecuto un gesto *longpress* sobre cualquier icono del menú de instanciación.	Se instancia un componente con el doble de tamaño mínimo a lo ancho.	Éxito.
		Verificar la acción de instanciar un componente *Imagen* importante.	Se ejecuto un gesto *longpress* sobre el icono de imagen del menú de	Se instancia un componente imagen con el doble de tamaño mínimo a lo alto.	Éxito.

			instanciación.		
		Verificar la acción de instanciar un componente *Texto*.	Se presiono sobre el icono de texto.	Se instancia un componente texto con una altura igual a la mitad de un módulo.	Éxito.

Tabla 3.4 Tabla de verificación *Native Mobile App Testing*.

b) Prototipos obtenidos

Dado que esta etapa debe ser lo menos subjetiva posible y circunscribirse a las normas de diagramado la búsqueda de la consistencia de los UI creados con los sistemas de retículas fue la máxima prioridad.

Se analizaron algunos modelos obtenidos reproduciéndolos en el entorno de Unity 5 para tener un despliegue visual del sistema de retículas obtenido:

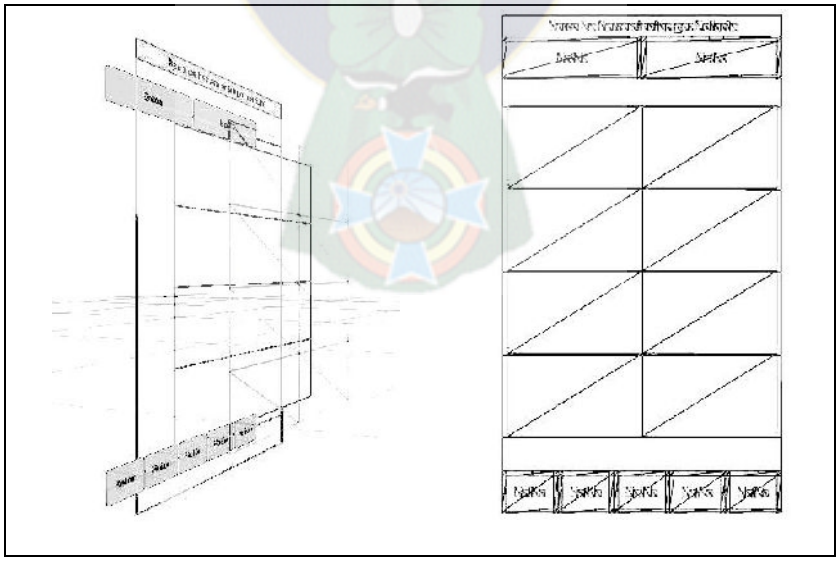


Figura 3.23: Retículas del Modelo 1.

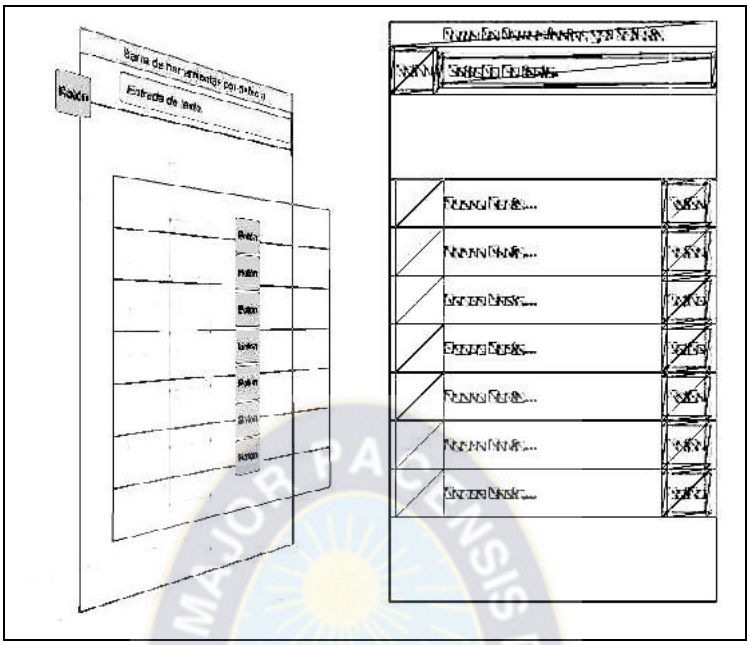


Figura 3.24: Retículas del Modelo 2.

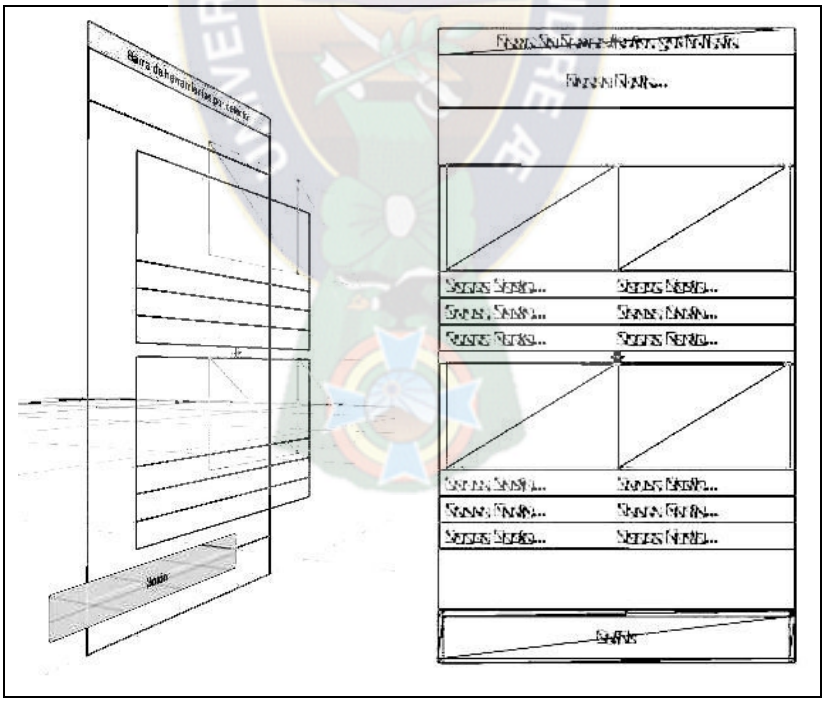


Figura 3.25: Retículas del Modelo 3.

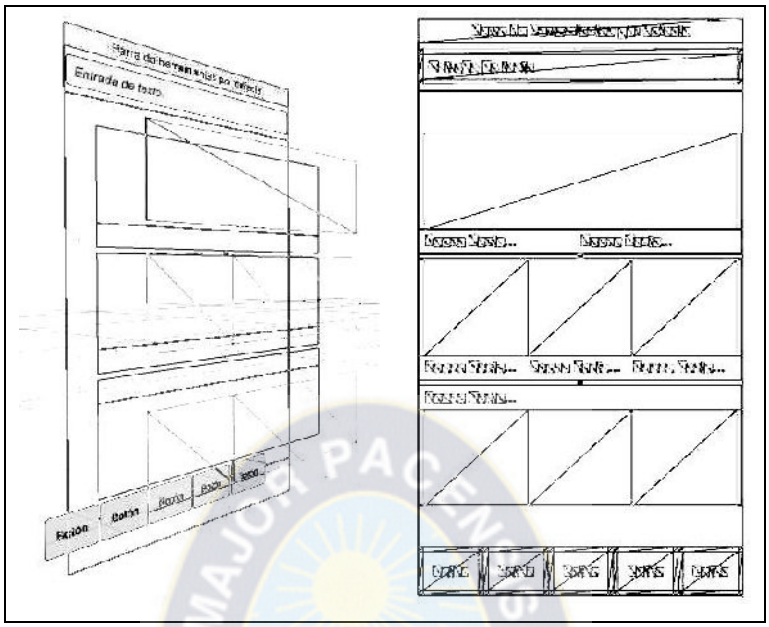


Figura 3.26: Retículas del Modelo 4.

Todos los modelos cumplen con las *Reglas para la verificación de retículas* propuestas en la Tabla 3.3. demostrándose así la consistencia del tamaño de los módulos y por lo tanto la consistencia de las retículas.

CAPITULO IV: DISCUSIÓN

4.1 Conclusiones

Al momento de la conclusión del presente trabajo se tienen las siguientes conclusiones:

- ✓ Los sistemas de retículas son actualmente utilizados en la diagramación de interfaces de usuario para pantallas de dispositivos móviles aunque estas maquetaciones se basan en igual medida en criterios de *Experiencia de Usuario*, y de ese modo, restándole un porcentaje de importancia a la teoría de retículas propia del *Diseño Grafico*.

- ✓ El programa obtenido consigue, mediante un enfoque más purista de teoría de retículas, maquetar interfaces de usuario circunscritas a un rígido sistema de medidas constantes y equivalentes obteniendo UI validadas por los sistemas de maquetación.

- ✓ La gran variedad de resoluciones de pantalla en dispositivos móviles convierte en dificultosa la tarea de obtener medidas físicas (centímetros, milímetros, etc.) iguales en cualquier pantalla: dos dispositivos con igual resolución e igual densidad de pixeles pueden tener diferentes tamaños físicos. Así, el testeo en diferentes dispositivos es de vital importancia a la hora de proyectar aplicaciones móviles. Para este fin existen en internet servicios de virtualización de dispositivos físicos.

- ✓ En la teoría de diagramación se reconocen los sistemas de retículas más relevantes para la creación de UI de dispositivos móviles. Estos sistemas de retículas crean espacios donde los elementos de interfaz pueden situarse. Así, esta teoría de retículas es la base para el prototipo de maquetación obtenido en el presente trabajo.

- ✓ *Unity 5* provee un sistema de creación *UI* mediante elementos de interfaz prefabricados por lo que el objetivo específico de *diseñar componentes de interfaz* es innecesario.

- ✓ Se tiene la herramienta de maquetación basada en normas de diagramación, su aplicación da como resultado *UI* de aplicaciones móviles regidas por estas normas que asignan espacios de despliegue de componentes de interfaz.

- ✓ Con sustento en las conclusiones establecidas se puede evaluar la hipótesis concluyendo que efectivamente el prototipo obtenido en base a normas de diagramación permite el diseño de interfaces de aplicaciones móviles.

4.2 Recomendaciones

El presente trabajo puede ser mejorado en diversas formas, algunas son:

- o La utilización de redes neuronales para predecir la adición de un elemento de interfaz y sugerir esta acción.

- o El estudio profundo de teoría de colores para crear paletas de colores aplicables a una interfaz de usuario para transmitir estados de ánimo según requerimiento.

- o Programación de un modulo *Drag & Drop* para la instanciación de componentes de interfaz.

- o Programación de métodos de exportación del UI creado obteniendo paletas de colores y coordenadas de posicionamiento y dimensión de componentes importables por diferentes *frameworks*.

- o Incorporar un estudio referente a interfaces de pantallas de *login*, componentes de video, interfaces de aplicaciones web y aplicaciones hibridas.

BIBLIOGRAFIA

Wong, W. (1991). *Fundamentos del diseño bi y tri-dimensional*. España: Gustavo Gili, S. A.

Salgueiro, E.I. (2014) .*Herramienta metodológica para el desarrollo y aplicaciones web móvil*.

Diseño gráfico. (19 oct 2017). En Wikipedia. Recuperado el 24 de octubre de 2017 de https://es.wikipedia.org/wiki/Dise%C3%B1o_gr%C3%A1fico

Aplicación informática. (24 oct 2017). En Wikipedia. Recuperado el 24 de octubre de 2017 de https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica

Román, I. (2012). 6 aplicaciones online para maquetar tus diseños. (13 ago 2012). En Trazos Web. Recuperado el 24 de octubre de 2017 de http://www.trazos-web.com/2012/08/13/6-aplicaciones-online-para-maquetar-tus-disenos/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+TrazosWeb+(Trazos+Web)

Martin, C. (29 jul 2004). Bruno Munari. Qué es un problema. Metodología para el diseño. En Alzado.org. Consultado el 9 de noviembre de 2017. https://www.alzado.org/articulo.php?id_art=354

Samara, T. (2006). *Diseñar con y sin retícula.* España: Gustavo Gili, S. A.

Ambrose, G., Harris, Paul. (2007). *The Layout Book.* Suiza :AVA Publishing S.A.

Cogut, V. (2015). *Unity 5 for Android Essentials*. Birmingham: Packt Publishing Ltd.

Martin, C. (29 jul 2004). *Bruno Munari. Qué es un problema. Metodología para el diseño*. Recuperado el 3 de abril de 2018 de https://alzado.org/articulo.php?id_art=354.

Ortego, D. (20 mar 2017). *¿Qué es C#?*. Recuperado el 3 de abril de 2018 de https://openwebinars.net/blog/que-es-c-introduccion/.

Munari, B. (1989). *Como nacen los objetos. Apuntes para una metodología proyectual*. España: Gustavo Gili, S. A.

Rosado, S. (2 feb 2015). *Tabla comparativa de los lenguajes de programación*. Recuperado el 7 de mayo de 2018 de http://desarrollowebydesarrolloweb.blogspot.com/2015/02/tabla-comparativa-de-los-lenguajes-de.html.

Lasso, I. (27 abr 2017). *7 Plataformas diferentes para desarrollar Android Apps*. Recuperado el 8 de mayo de 2018 de https://tekzup.com/7-plataformas-diferentes-desarrollar-android-apps/.

Android Developers. (25 abr 2018). *Permisos del Sistema*. Recuperado el 8 de mayo de 2018 de https://developer.android.com/guide/topics/security/permissions?hl=es-419.

Platzi. (oct 2017). *Teoría del Color en el diseño de interfaces*. Recuperado el 8 de mayo de 2018 de https://platzi.com/blog/color-en-interfaces/.

NorfiPC. (2018). *Medidas de la pantalla y resolución de los teléfonos celulares y tabletas*. Recuperado el 8 de mayo de 2018 de https://norfipc.com/celulares/ultimos-telefonos-celulares-moviles-con-android.php.

Google Developers. (23 may 2018). *Estilos accesibles*. Recuperado el 3 de junio de 2018 de https://developers.google.com/web/fundamentals/accessibility/accessible-styles?hl=es.

Estévez, J. (16 feb 2014). *Todo sobre las resoluciones en pantallas móviles*. Recuperado el 3 de junio de 2018 de https://www.elgrupoinformatico.com/todo-sobre-las-resoluciones-pantalla-moviles-t18462.html.

ANEXO A: CODIGO FUENTE

btnsHeaderMenu.cs:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class btnsHeaderMenu : MonoBehaviour {
	public Button btnHeader;
	public Button btnFooter;
	public Button btnNoHnF;
	public Button btnDove;
	public Button btnImage;
	public Button btnText;
	public Button btnBtn;
	public Button btnItf;
	public Canvas headerCanvas;
	public Canvas containerCanvas;
	public InputField inputText;
	public GameObject globalVariables;
	public GameObject fixedBar;

	void Start () {
	}
	void Update () {
	}
	public void undoBhvr()
	{
		if(headerCanvas.transform.childCount>0)
		{
			headerCanvas.transform.GetChild(
```

```
                    globalVariables.GetComponent<gVar>().totalInHeader-1).

transform.SetParent(globalVariables.GetComponent<gVar>().trashCanvas.transform);
                globalVariables.GetComponent<gVar>().totalInHeader--;
        }
}
public void doveBhvr()
{
        if(globalVariables.GetComponent<gVar>().totalInHeader>=3)
        {
                btnHeader.GetComponent<Image> ().enabled = false;
                btnHeader.GetComponent<Button> ().enabled = false;
        }
        if(globalVariables.GetComponent<gVar>().totalInFooter>=5)
        {
                btnFooter.GetComponent<Image> ().enabled = false;
                btnFooter.GetComponent<Button> ().enabled = false;
        }
        globalVariables.GetComponent<gVar> ().
                beginCanvas.GetComponent<Canvas>().enabled = true;
        globalVariables.GetComponent<gVar> ().
                footerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                headerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                continueMenuCanvas.GetComponent<Canvas>().enabled = false;
}
public void inputBhvr()
{
```

```
            if (globalVariables.GetComponent<gVar>().totalInHeader < 3) {
                inputText.transform.SetParent (headerCanvas.transform);
                headerToContainerCanvas ();
                positionHeaderCanvas ();
                sizeHeaderCanvas ();
                inputResizeWidth();
                inputText.GetComponent<LayoutElement> ().preferredWidth =
                        globalVariables.GetComponent<gVar> ().mainCanvas.
                        GetComponent<RectTransform>().rect.width -
                        globalVariables.GetComponent<gVar>
().moduleWidth*2;
                inputText.GetComponent<LayoutElement> ().flexibleWidth =
                    globalVariables.GetComponent<gVar> ().mainCanvas.
                        GetComponent<RectTransform>().rect.width -
                        globalVariables.GetComponent<gVar> ().moduleWidth;
                globalVariables.GetComponent<gVar>().totalInHeader++;
                globalVariables.GetComponent<gVar>().txt.
                    GetComponent<Text>().text = ">Instanciada Entrada de texto.";

                globalVariables.GetComponent<gVar> ().headerAdd = 1;
                resizeMainCanvas();
            }
        }
        public void resizeMainCanvas()
        {
            if(!theresFooter()){
                globalVariables.GetComponent<gVar> ().mainCanvas.
                    GetComponent<RectTransform> ().localPosition =
                        new Vector3 (0, -(
```

```
        fixedBar.GetComponent<RectTransform>().rect.height/2)
                                        -getHeightHeaderCanvas()/2, 0);
                globalVariables.GetComponent<gVar> ().mainCanvas.
                    GetComponent<RectTransform> ().sizeDelta =
                        new                                     Vector2
(containerCanvas.GetComponent<RectTransform> ().rect.width,
containerCanvas.GetComponent<RectTransform> ().rect.height -
fixedBar.GetComponent<RectTransform>().rect.height
                                        -getHeightHeaderCanvas());
            }
            if(theresFooter())
            {
                globalVariables.GetComponent<gVar> ().mainCanvas.
                    GetComponent<RectTransform> ().localPosition =
                        new          Vector3          (0,          -
(fixedBar.GetComponent<RectTransform>().rect.height/2)
                                        -getHeightHeaderCanvas()/2
                                        +getHeightFooterCanvas()/2,
                                        0);
                globalVariables.GetComponent<gVar> ().mainCanvas.
                    GetComponent<RectTransform> ().sizeDelta =
                        new                                     Vector2
(containerCanvas.GetComponent<RectTransform> ().rect.width,
containerCanvas.GetComponent<RectTransform> ().rect.height -
fixedBar.GetComponent<RectTransform>().rect.height
```

```
                                        -getHeightHeaderCanvas()
                                        -getHeightFooterCanvas());
        }
}
public float getHeightFooterCanvas()
{
        float f = globalVariables.GetComponent<gVar> ().minHeight;
        return f;
}
public bool theresFooter()
{
        if (globalVariables.GetComponent<gVar> ().footerAdd == 1)
                return true;
        else
                return false;
}
public void headerToContainerCanvas()
{
        headerCanvas.transform.SetParent (
                globalVariables.GetComponent<gVar>().containerCanvas.transform);
}
public void sizeHeaderCanvas()
{
        headerCanvas.GetComponent<RectTransform> ().sizeDelta = new Vector2 (
                getWidthHeaderCanvas(),
           getHeightHeaderCanvas());
}
public float getWidthHeaderCanvas()
{
```

```
            float f = globalVariables.GetComponent<gVar> ().screenWidth;
            return f;
        }
        public float getHeightHeaderCanvas()
        {
            float f = globalVariables.GetComponent<gVar> ().minHeight;
            return f;
        }
        public void imageBhvr()
        {
            if (globalVariables.GetComponent<gVar>().totalInHeader < 3)
            {
                resizeImage();
                globalVariables.GetComponent<gVar>
().imagesPool.transform.GetChild (0).
                    transform.SetParent (headerCanvas.transform);
                headerToContainerCanvas();
                positionHeaderCanvas ();
                sizeHeaderCanvas ();
                globalVariables.GetComponent<gVar>().totalInHeader++;
                globalVariables.GetComponent<gVar>().txt.
                    GetComponent<Text>().text = ">Instanciada Imagen.";
                globalVariables.GetComponent<gVar> ().headerAdd = 1;
                resizeMainCanvas();
            }
        }
        public void resizeImage()
        {
            globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
```

```
                GetComponent<LayoutElement>().minWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
}
public void resizeText()
{
        globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
}
public void textBhvr()
{
```

```
            if (globalVariables.GetComponent<gVar>().totalInHeader < 3)
            {
                resizeText();
                globalVariables.GetComponent<gVar>  ().textsPool.transform.GetChild
(0).
                    transform.SetParent (headerCanvas.transform);
                headerToContainerCanvas();
                positionHeaderCanvas ();
                sizeHeaderCanvas ();

        headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                    GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

                globalVariables.GetComponent<gVar>().totalInHeader++;
                globalVariables.GetComponent<gVar>().txt.
                    GetComponent<Text>().text = ">Instanciado Texto.";
                globalVariables.GetComponent<gVar> ().headerAdd = 1;
                resizeMainCanvas();
            }
        }
        public void buttonBhvr()
        {
            if (globalVariables.GetComponent<gVar>().totalInHeader < 3)
            {
                globalVariables.GetComponent<gVar>
().buttonsPool.transform.GetChild (0).
                    transform.SetParent (headerCanvas.transform);
                headerToContainerCanvas();
                positionHeaderCanvas ();
```

```
                sizeHeaderCanvas ();
                lastChildResizeHeight ();
                buttonResizeWidth();
                globalVariables.GetComponent<gVar>().totalInHeader++;
                globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Instanciado Boton.";
                globalVariables.GetComponent<gVar> ().headerAdd = 1;
                resizeMainCanvas();
        }
    }
    public void buttonResizeWidth()
    {

    headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                GetComponent<LayoutElement> ().minWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;

    headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                GetComponent<LayoutElement> ().preferredWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
    }
    public void textResizeWidth()
    {

    headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                GetComponent<LayoutElement> ().minWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
```

```
    }
    public void inputResizeWidth()
    {

    headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                GetComponent<LayoutElement> ().minWidth =
                    globalVariables.GetComponent<gVar>().screenWidth -
                    globalVariables.GetComponent<gVar> ().minHeight*2;
    }
    public void headerBhvr()
    {
        globalVariables.GetComponent<gVar>().txt.
                GetComponent<Text>().text = ">3 Componentes maximo.";
        globalVariables.GetComponent<gVar> ().
                beginCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                footerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                headerMenuCanvas.GetComponent<Canvas>().enabled = true;
        globalVariables.GetComponent<gVar> ().
                continueMenuCanvas.GetComponent<Canvas>().enabled = false;
    }
    public void positionHeaderCanvas ()
    {
        headerCanvas.GetComponent<RectTransform> ().localPosition =
                new Vector3(
                    0,
                    containerCanvas.GetComponent<RectTransform> ().position.y-
```

```
                fixedBar.GetComponent<RectTransform>().rect.height,
                0);
        }
        public void lastChildResizeHeight()
        {

        headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                GetComponent<LayoutElement> ().minHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;



        headerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInHea
der).
                GetComponent<LayoutElement> ().preferredHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
        }
}
```

btnsFooterMenu.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class btnsFooterMenu : MonoBehaviour {
        public Button btnHeader;
        public Button btnFooter;
        public Button btnNoHnF;
        public Button btnDove;
```

```
public Button btnText;
public Button btnBtn;
public Canvas footerCanvas;
public Canvas containerCanvas;
public GameObject globalVariables;
public GameObject fixedBar;
void Start () {
}
void Update () {
}
public void undoBhvr()
{
        if(footerCanvas.transform.childCount>0)
        {
                footerCanvas.transform.GetChild(
                        globalVariables.GetComponent<gVar>().totalInFooter-1).

transform.SetParent(globalVariables.GetComponent<gVar>().trashCanvas.transform);
                globalVariables.GetComponent<gVar>().totalInFooter--;
        }
}
public void doveBhvr()
{
        if(globalVariables.GetComponent<gVar>().totalInHeader>=3)
        {
                btnHeader.GetComponent<Image> ().enabled = false;
                btnHeader.GetComponent<Button> ().enabled = false;
        }
        if(globalVariables.GetComponent<gVar>().totalInFooter>=5)
```

```
        {
            btnFooter.GetComponent<Image> ().enabled = false;
            btnFooter.GetComponent<Button> ().enabled = false;
        }
        globalVariables.GetComponent<gVar> ().
            beginCanvas.GetComponent<Canvas>().enabled = true;
        globalVariables.GetComponent<gVar> ().
            footerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
            headerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
            continueMenuCanvas.GetComponent<Canvas>().enabled = false;
    }
    public void resizeMainCanvas()
    {
        if(!theresHeader()){
            globalVariables.GetComponent<gVar> ().mainCanvas.
                GetComponent<RectTransform> ().localPosition =
                    new Vector3 (0, -(
fixedBar.GetComponent<RectTransform>().rect.height/2)
                        +getHeightFooterCanvas()/2, 0);
            globalVariables.GetComponent<gVar> ().mainCanvas.
                GetComponent<RectTransform> ().sizeDelta =
                    new Vector2
(containerCanvas.GetComponent<RectTransform> ().rect.width,
containerCanvas.GetComponent<RectTransform> ().rect.height -
```

```
fixedBar.GetComponent<RectTransform>().rect.height
                                        -getHeightFooterCanvas());
            }
            if(theresHeader())
            {
                globalVariables.GetComponent<gVar> ().mainCanvas.
                    GetComponent<RectTransform> ().localPosition =
                        new Vector3 (0, -
(fixedBar.GetComponent<RectTransform>().rect.height/2)
                                        -getHeightHeaderCanvas()/2
                                        +getHeightFooterCanvas()/2,
                                        0);
                globalVariables.GetComponent<gVar> ().mainCanvas.
                    GetComponent<RectTransform> ().sizeDelta =
                        new Vector2
(containerCanvas.GetComponent<RectTransform> ().rect.width,

containerCanvas.GetComponent<RectTransform> ().rect.height -

fixedBar.GetComponent<RectTransform>().rect.height
                                        -getHeightHeaderCanvas()
                                        -getHeightFooterCanvas());
            }
        }
        public bool theresHeader()
        {
            if (globalVariables.GetComponent<gVar> ().headerAdd == 1)
                return true;
            else
```

```
                return false;
        }
        public float getHeightHeaderCanvas()
        {
                float f = globalVariables.GetComponent<gVar> ().minHeight;
                return f;
        }
        public void footerToContainerCanvas()
        {
                footerCanvas.transform.SetParent (
                        globalVariables.GetComponent<gVar>().containerCanvas.transform);
        }
        public void textBhvr()
        {
                if (globalVariables.GetComponent<gVar>().totalInFooter < 1)
                {
                        globalVariables.GetComponent<gVar>  ().textsPool.transform.GetChild
(0).
                                transform.SetParent (footerCanvas.transform);
                        footerToContainerCanvas();
                        positionFooterCanvas ();
                        sizeFooterCanvas ();

        footerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInFoot
er).
                                GetComponent<Text>().alignment = TextAnchor.MiddleCenter;

                        lastChildResize ();
                        globalVariables.GetComponent<gVar>().totalInFooter++;
```

```
                globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Instanciado Texto.";
                globalVariables.GetComponent<gVar> ().footerAdd = 1;
                resizeMainCanvas();
        }
    }
    public void buttonBhvr()
    {
        if (globalVariables.GetComponent<gVar>().totalInFooter < maxCtsAllowed())
        {
                resizeButton ();
                globalVariables.GetComponent<gVar>
().buttonsPool.transform.GetChild (0).
                        transform.SetParent (footerCanvas.transform);
                footerToContainerCanvas();
                positionFooterCanvas ();
                sizeFooterCanvas ();
                globalVariables.GetComponent<gVar>().totalInFooter++;
                globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Instanciado Boton.";
                globalVariables.GetComponent<gVar> ().footerAdd = 1;
                resizeMainCanvas();
        }
    }
    public void resizeButton ()
    {
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
```

```
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;
}
public void footerBhvr()
{
        globalVariables.GetComponent<gVar>().txt.
                GetComponent<Text>().text = ">3 Componentes maximo.";
        globalVariables.GetComponent<gVar> ().
                beginCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                footerMenuCanvas.GetComponent<Canvas>().enabled = true;
        globalVariables.GetComponent<gVar> ().
                headerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                continueMenuCanvas.GetComponent<Canvas>().enabled = false;
}
public void sizeFooterCanvas()
{
        footerCanvas.GetComponent<RectTransform> ().sizeDelta = new Vector2 (
                getWidthFooterCanvas(),      getHeightFooterCanvas());
}
public float getWidthFooterCanvas()
```

```
{
        float f = globalVariables.GetComponent<gVar> ().screenWidth;
        return f;
}
public float getHeightFooterCanvas()
{
        float f = globalVariables.GetComponent<gVar> ().minHeight;
        return f;
}
public void positionFooterCanvas ()
{
        footerCanvas.GetComponent<RectTransform> ().localPosition =
                new Vector3(
                        0,
                        -containerCanvas.GetComponent<RectTransform> ().position.y,
                        0);
}
public void lastChildResize()
{

footerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInFoot
er).
                GetComponent<LayoutElement> ().minHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;


footerCanvas.transform.GetChild(globalVariables.GetComponent<gVar>().totalInFoot
er).
                GetComponent<LayoutElement> ().preferredHeight =
```

```
                    globalVariables.GetComponent<gVar> ().minHeight;
        }
        public int maxCtsAllowed()
        {
            if(Screen.dpi>=240)
                return 5;
            else
            {
                if(Screen.dpi<240 && Screen.dpi >96)
                    return 4;
                else
                {
                    if(Screen.dpi <=96)
                        return 5;
                    else
                        return 5;
                }
            }
        }
}
```

btnContinueMenu.cs:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class btnsContinueMenu : MonoBehaviour {
        public Button btnHeader;
        public Button btnFooter;
        public Button btnNoHnF;
```

```
public Button btnDove;
public Button btnText;
public Button btnBtn;
public Button btnImg;
public Button btnUndo;
public Canvas containerCanvas;
public GameObject globalVariables;
public int stateOfAddMethod;
public int totalInSameRow;
public float totalHeightInUse;
void Start () {
        stateOfAddMethod = 0;
        totalInSameRow = 0;
        totalHeightInUse = 0;
}
void Update () {
}
public void undoBhvr()
{

if(globalVariables.GetComponent<gVar>().mainCanvas.transform.childCount>0)
        {

if(globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
                    globalVariables.GetComponent<gVar>().numberOfRows-1).
                transform.childCount>0)
                {

globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
```

```
globalVariables.GetComponent<gVar>().numberOfRows-1).
                                    transform.GetChild(totalInSameRow-1).
                                        transform.SetParent(

globalVariables.GetComponent<gVar>().trashCanvas.transform);
                        totalInSameRow --;
                }
        }
}
public void doveBhvr()
{
        if(globalVariables.GetComponent<gVar>().totalInHeader>=3)
        {
                btnHeader.GetComponent<Image> ().enabled = false;
                btnHeader.GetComponent<Button> ().enabled = false;
        }
        if(globalVariables.GetComponent<gVar>().totalInFooter>=5)
        {
                btnFooter.GetComponent<Image> ().enabled = false;
                btnFooter.GetComponent<Button> ().enabled = false;
        }

if(globalVariables.GetComponent<gVar>().mainCanvas.transform.childCount>0)
        {

if(globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
                        globalVariables.GetComponent<gVar>().numberOfRows-1).
                  transform.childCount>0)
```

```
                {
                        totalHeightInUse = totalHeightInUse +

globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(

globalVariables.GetComponent<gVar>().numberOfRows-1).
                                        GetComponent<RectTransform>().rect.height;
                }
        }
        globalVariables.GetComponent<gVar> ().
                beginCanvas.GetComponent<Canvas>().enabled = true;
        globalVariables.GetComponent<gVar> ().
                footerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                headerMenuCanvas.GetComponent<Canvas>().enabled = false;
        globalVariables.GetComponent<gVar> ().
                continueMenuCanvas.GetComponent<Canvas>().enabled = false;
}
public void buttonBhvr()
{
        if(spaceAvailable())
        {
                if(stateOfAddMethod == 1)
                {
                        if (totalInSameRow >= maxInRowAllowed())
                        {
                                globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text = ">Error.";
                        }
```

```
                        if (totalInSameRow < maxInRowAllowed())
                        {
                                resizeButton();
                                globalVariables.GetComponent<gVar>
().buttonsPool.transform.GetChild (0).
                                        transform.SetParent (
                                                globalVariables.GetComponent<gVar>
().mainCanvas.transform.GetChild(
                                                globalVariables.GetComponent<gVar>
().mainCanvas.transform.childCount-1));
                                totalInSameRow ++;
                                globalVariables.GetComponent<gVar>().totalInMain++;
                                globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text = ">Instanciado
boton.";
                        }
                }
        }
        else
                globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">No hay espacio libre.";
    }
    public int maxInRowAllowed()
    {
        if(Screen.dpi>=240)
                return 3;
        else
        {
                if(Screen.dpi<240 && Screen.dpi >96)
```

```
                return 2;
            else
            {
                if(Screen.dpi <=96)
                    return 3;,
                else
                    return 3;
            }
        }
    }
    public void textBhvr()
    {
        if(spaceTextAvailable())
        {
            if(stateOfAddMethod == 1)
            {
                if (totalInSameRow >= maxInRowAllowed())
                {
                    globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Error.";
                }
                if (totalInSameRow < maxInRowAllowed())
                {
                    resizeText();

                    globalVariables.GetComponent<gVar>
().textsPool.transform.GetChild (0).
                        transform.SetParent (
                            globalVariables.GetComponent<gVar>
```

```
().mainCanvas.transform.GetChild(
                                                globalVariables.GetComponent<gVar>
().mainCanvas.transform.childCount-1));
                                totalInSameRow ++;
                                globalVariables.GetComponent<gVar>().totalInMain++;
                                globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text     =     ">Instanciada
imagen.";
                                }
                        }
                }
                else
                        globalVariables.GetComponent<gVar>().txt.
                                GetComponent<Text>().text = ">No hay espacio libre.";
        }
        public void resizeText()
        {
                globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                        GetComponent<LayoutElement>().minWidth =
                                globalVariables.GetComponent<gVar> ().minHeight;
                globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                        GetComponent<LayoutElement>().minHeight =
                                globalVariables.GetComponent<gVar> ().minHeight/2;
                globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                        GetComponent<LayoutElement>().preferredWidth =
                                globalVariables.GetComponent<gVar> ().minHeight;
                globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                        GetComponent<LayoutElement>().preferredHeight =
                                globalVariables.GetComponent<gVar> ().minHeight/2;
```

```
    }
    public void imageBhvr()
    {
        if(spaceAvailable())
        {
            if(stateOfAddMethod == 1)
            {
                if (totalInSameRow >= maxInRowAllowed())
                {
                    globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Error.";
                }
                if (totalInSameRow < maxInRowAllowed())
                {
                    resizeImage();
                    globalVariables.GetComponent<gVar>
().imagesPool.transform.GetChild (0).
                        transform.SetParent (
                            globalVariables.GetComponent<gVar>
().mainCanvas.transform.GetChild(
                            globalVariables.GetComponent<gVar>
().mainCanvas.transform.childCount-1));
                    totalInSameRow ++;
                    globalVariables.GetComponent<gVar>().totalInMain++;
                    globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Instanciada
imagen.";
                }
            }
```

```
            }
            else
                globalVariables.GetComponent<gVar>().txt.
                    GetComponent<Text>().text = ">No hay espacio libre.";
        }
        public void importantImageBhvr()
        {
            if(remainingImportantSize())
            {
                if(stateOfAddMethod == 1)
                {
                    if (totalInSameRow >= maxInRowAllowed())
                    {
                        globalVariables.GetComponent<gVar>().txt.
                            GetComponent<Text>().text = ">Error.";
                    }
                    if (totalInSameRow < maxInRowAllowed())
                    {
                        resizeImportantImage();

                        globalVariables.GetComponent<gVar>
().imagesPool.transform.GetChild (0).
                            transform.SetParent (
                                globalVariables.GetComponent<gVar>
().mainCanvas.transform.GetChild(
                                globalVariables.GetComponent<gVar>
().mainCanvas.transform.childCount-1));
                        totalInSameRow ++;
                        globalVariables.GetComponent<gVar>().totalInMain++;
```

```
                                globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text = ">Instanciada
imagen importante.";
                        }
                    }
                }
                else
                    globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">Sin espacio para imagen
importante.";

        }
        public void resizeImage()
        {
            globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minWidth =
                    globalVariables.GetComponent<gVar> ().minHeight;
            globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minHeight =
                    globalVariables.GetComponent<gVar> ().minHeight;
            globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredWidth =
                    globalVariables.GetComponent<gVar> ().minHeight;
            globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredHeight =
                    globalVariables.GetComponent<gVar> ().minHeight;
        }
        public void resizeImportantImage()
        {
```

```
        resizeImage();
        globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredHeight =
                        globalVariables.GetComponent<gVar> ().moduleHeight*2;
        globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredWidth =
                        globalVariables.GetComponent<gVar> ().mainCanvas.
                        GetComponent<RectTransform>().rect.width -
                        globalVariables.GetComponent<gVar> ().moduleWidth*2;
        globalVariables.GetComponent<gVar> ().imagesPool.transform.GetChild (0).
                GetComponent<LayoutElement>().flexibleWidth =
                        globalVariables.GetComponent<gVar> ().mainCanvas.
                        GetComponent<RectTransform>().rect.width -
                        globalVariables.GetComponent<gVar> ().moduleWidth*3;
}
public void importantTextBhvr()
{
        if(spaceTextAvailable())
        {
                if(stateOfAddMethod == 1)
                {
                        if (totalInSameRow >= maxInRowAllowed())
                        {
                                globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text = ">Error.";
                        }
                        if (totalInSameRow < maxInRowAllowed())
                        {
                                resizeImportantText();
```

```
                                    globalVariables.GetComponent<gVar>
().textsPool.transform.GetChild (0).
                                        transform.SetParent (
                                            globalVariables.GetComponent<gVar>
().mainCanvas.transform.GetChild(
                                            globalVariables.GetComponent<gVar>
().mainCanvas.transform.childCount-1));
                                    totalInSameRow ++;
                                    globalVariables.GetComponent<gVar>().totalInMain++;
                                    globalVariables.GetComponent<gVar>().txt.
                                        GetComponent<Text>().text = ">Instanciado
Texto Importante.";
                                }
                            }
                        }
                    }
                    public void resizeImportantText()
                    {
                        resizeText();
                        globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                            GetComponent<LayoutElement>().preferredWidth =
                                globalVariables.GetComponent<gVar> ().mainCanvas.
                                GetComponent<RectTransform>().rect.width -
                                globalVariables.GetComponent<gVar> ().moduleWidth*2;
                        globalVariables.GetComponent<gVar> ().textsPool.transform.GetChild (0).
                            GetComponent<LayoutElement>().flexibleWidth =
                                globalVariables.GetComponent<gVar> ().mainCanvas.
                                GetComponent<RectTransform>().rect.width -
                                globalVariables.GetComponent<gVar> ().moduleWidth*3;
```

```
        }
        public void importantButtonBhvr()
        {
                if(spaceAvailable())
                {
                        if(stateOfAddMethod == 1)
                        {
                                if (totalInSameRow >= maxInRowAllowed())
                                {
                                        globalVariables.GetComponent<gVar>().txt.
                                                GetComponent<Text>().text = ">Error.";
                                }
                                if (totalInSameRow < maxInRowAllowed())
                                {
                                        resizeImportantButton();
                                        globalVariables.GetComponent<gVar>
().buttonsPool.transform.GetChild (0).
                                                transform.SetParent (
                                                        globalVariables.GetComponent<gVar>
().mainCanvas.transform.GetChild(
                                                        globalVariables.GetComponent<gVar>
().mainCanvas.transform.childCount-1));
                                        totalInSameRow ++;
                                        globalVariables.GetComponent<gVar>().totalInMain++;
                                        globalVariables.GetComponent<gVar>().txt.
                                                GetComponent<Text>().text = ">Instanciado
boton importante.";
                                }
                        }
```

```
        }
}
public void resizeButton()
{
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minWidth =
                        globalVariables.GetComponent<gVar> ().minHeight;

        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().minHeight =
                        globalVariables.GetComponent<gVar> ().minHeight;

}
public void resizeImportantButton()
{
        resizeButton();
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().preferredWidth =
                        globalVariables.GetComponent<gVar> ().mainCanvas.
                        GetComponent<RectTransform>().rect.width -
                        globalVariables.GetComponent<gVar> ().moduleWidth*2;
        globalVariables.GetComponent<gVar> ().buttonsPool.transform.GetChild (0).
                GetComponent<LayoutElement>().flexibleWidth =
                        globalVariables.GetComponent<gVar> ().mainCanvas.
                        GetComponent<RectTransform>().rect.width -
                        globalVariables.GetComponent<gVar> ().moduleWidth*3;
}
public void continueBhvr()
{
```

```
                globalVariables.GetComponent<gVar>().txt.
                        GetComponent<Text>().text = ">3 componentes maximo.";
                globalVariables.GetComponent<gVar> ().
                        beginCanvas.GetComponent<Canvas>().enabled = false;
                globalVariables.GetComponent<gVar> ().
                        footerMenuCanvas.GetComponent<Canvas>().enabled = false;
                globalVariables.GetComponent<gVar> ().
                        headerMenuCanvas.GetComponent<Canvas>().enabled = false;
                globalVariables.GetComponent<gVar> ().
                        continueMenuCanvas.GetComponent<Canvas>().enabled = true;
        }
        public void sameRowBhvr()
        {
                continueBhvr();
                stateOfAddMethod = 1;
                totalInSameRow = 0;
                globalVariables.GetComponent<gVar>
().hztlCanvasPool.transform.GetChild(0).
                        GetComponent<HorizontalLayoutGroup>().padding.left =
                                (globalVariables.GetComponent<gVar>().spaceHeight/5)*3;
                globalVariables.GetComponent<gVar>
().hztlCanvasPool.transform.GetChild(0).
                        GetComponent<HorizontalLayoutGroup>().padding.right =
                                (globalVariables.GetComponent<gVar>().spaceHeight/5)*3;
                globalVariables.GetComponent<gVar>
().hztlCanvasPool.transform.GetChild(0).
                        transform.SetParent(
                                globalVariables.GetComponent<gVar>
().mainCanvas.transform);
```

```
            globalVariables.GetComponent<gVar> ().numberOfRows ++;
        }

        public void noSameRowBhvr()
        {
            continueBhvr();
            globalVariables.GetComponent<gVar>().txt.
                GetComponent<Text>().text = ">Diagramacion principal sin filas.";
            stateOfAddMethod = 0;
        }
        public bool spaceAvailable()
        {
            if(remainingSize())
                return true;
            else
                return false;
        }

        public bool remainingSize()
        {
            if(globalVariables.GetComponent<gVar>
().mainCanvas.GetComponent<RectTransform>().rect.height
                - totalHeightInUse > globalVariables.GetComponent<gVar> ().minHeight)
            {
                return true;
            }
            else
                return false;
        }
```

```
        public bool spaceAvailableImportantImage()
        {
                if(remainingImportantSize())
                        return true;
                else
                        return false;
        }
        public bool remainingImportantSize()
        {
                if(globalVariables.GetComponent<gVar>
().mainCanvas.GetComponent<RectTransform>().rect.height
                  - totalHeightInUse > globalVariables.GetComponent<gVar> ().minHeight*2)
                {
                        return true;
                }
                else
                        return false;
        }
        public bool spaceTextAvailable()
        {
                if(remainingTextSize())
                        return true;
                else
                        return false;
        }
        public bool remainingTextSize()
        {
                if(globalVariables.GetComponent<gVar>
().mainCanvas.GetComponent<RectTransform>().rect.height
```

```
            - totalHeightInUse > globalVariables.GetComponent<gVar> ().minHeight/2)
        {
            return true;
        }
        else
            return false;
    }
    public void addSpaceBetweenRows()
    {
        if(spaceAvailable())
        {
            resizeSpace();
            globalVariables.GetComponent<gVar>
().whiteSpacesPool.transform.GetChild (0).
                transform.SetParent (
                    globalVariables.GetComponent<gVar>
().mainCanvas.transform);
            globalVariables.GetComponent<gVar>().txt.
                GetComponent<Text>().text = ">Espacio añadido despues de la
fila.";
        }
        else
            globalVariables.GetComponent<gVar>().txt.
                GetComponent<Text>().text = ">No hay espacio libre.";
        if(globalVariables.GetComponent<gVar>().totalInHeader>=3)
        {
            btnHeader.GetComponent<Image> ().enabled = false;
            btnHeader.GetComponent<Button> ().enabled = false;
        }
```

```
        if(globalVariables.GetComponent<gVar>().totalInFooter>=5)
        {
                btnFooter.GetComponent<Image> ().enabled = false;
                btnFooter.GetComponent<Button> ().enabled = false;
        }
        globalVariables.GetComponent<gVar> ().numberOfRows ++;

if(globalVariables.GetComponent<gVar>().mainCanvas.transform.childCount>0)
        {

if(globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(
                        globalVariables.GetComponent<gVar>().numberOfRows-2).
                  transform.childCount>0)
                {
                        totalHeightInUse = totalHeightInUse +

globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(

globalVariables.GetComponent<gVar>().numberOfRows-1).
                                        GetComponent<RectTransform>().rect.height +

globalVariables.GetComponent<gVar>().mainCanvas.transform.GetChild(

globalVariables.GetComponent<gVar>().numberOfRows-2).
                                        GetComponent<RectTransform>().rect.height;
                }
        }
        globalVariables.GetComponent<gVar> ().
                beginCanvas.GetComponent<Canvas>().enabled = true;
        globalVariables.GetComponent<gVar> ().
```

```
                footerMenuCanvas.GetComponent<Canvas>().enabled = false;
            globalVariables.GetComponent<gVar> ().
                headerMenuCanvas.GetComponent<Canvas>().enabled = false;
            globalVariables.GetComponent<gVar> ().
                continueMenuCanvas.GetComponent<Canvas>().enabled = false;
        }
        public void resizeSpace()
        {
            globalVariables.GetComponent<gVar>  ().whiteSpacesPool.transform.GetChild
(0).
                GetComponent<LayoutElement>().minWidth =
                    (globalVariables.GetComponent<gVar> ().spaceHeight/5)*3;
            globalVariables.GetComponent<gVar>  ().whiteSpacesPool.transform.GetChild
(0).
                GetComponent<LayoutElement>().minHeight =
                    (globalVariables.GetComponent<gVar> ().spaceHeight/5)*3;
            globalVariables.GetComponent<gVar>  ().whiteSpacesPool.transform.GetChild
(0).
                GetComponent<LayoutElement>().preferredWidth =
                    (globalVariables.GetComponent<gVar> ().spaceHeight/5)*3;
            globalVariables.GetComponent<gVar>  ().whiteSpacesPool.transform.GetChild
(0).
                GetComponent<LayoutElement>().preferredHeight =
                    (globalVariables.GetComponent<gVar> ().spaceHeight/5)*3;
        }
}
```

gVar.cs:

```
using UnityEngine;
using System.Collections;
```

```
using UnityEngine.UI;

public class gVar : MonoBehaviour {
        public int ctsInMainCanvas;
        public float screenWidth;
        public float screenHeight;
        public float moduleWidth;
        public float moduleHeight;
        public int totalModules;
        public Canvas mainCanvas;
        public Canvas containerCanvas;
        public float cwidth;
        public float cheight;
        public Text txt;
        public Canvas canvas_1_2;
        public int thisMuchElements;
        public GameObject hztlCanvasPool;
        public GameObject vrtlCanvasPool;
        public GameObject imagesPool;
        public GameObject textsPool;
        public GameObject buttonsPool;
        public int totalInHeader;
        public int totalInFooter;
        public int totalInMain;
        public Canvas beginCanvas;
        public Canvas headerMenuCanvas;
        public Canvas footerMenuCanvas;
        public Canvas continueMenuCanvas;
        public int footerAdd;
```

```
public int headerAdd;
public int minHeight;
public int numberOfRows;
public Canvas trashCanvas;
public int modulesInWidth;
public int modulesInHeight;
public int spaceHeight;
public GameObject whiteSpacesPool;
void Start () {
        print ("Resolution: " + Screen.currentResolution);
        print ("DPI: " + Screen.dpi);
        txt.text = ">DPI: " + Screen.dpi;
        ctsInMainCanvas = 0;
        getResolution();
        cwidth = 0;
        cheight = 0;
        thisMuchElements = 0;
        totalInHeader = 0;
        totalInFooter = 0;
        totalInMain = 0;
        footerAdd = 0;
        headerAdd = 0;
        minHeight = getMinHeight();
        spaceHeight = minHeight/4;
        numberOfRows = 0;
        resizeBeginCanvasMenu();
}
public int getMinHeight()
{
```

```
        if(Screen.dpi>=240)
                return 80;
        else
        {
                if(Screen.dpi<240 && Screen.dpi >96)
                        return 56;
                else
                {
                        if(Screen.dpi <=96)
                                return 48;,
                        else
                                return 80;
                }
        }
}
public void getResolution()
{
        screenWidth = containerCanvas.GetComponent<RectTransform>().rect.width;
        screenHeight = containerCanvas.GetComponent<RectTransform>().rect.height;
}
void Update () {
        getResolution();
        calculateModules();
}
public void calculateModules()
{
        moduleWidth = minHeight;
        moduleHeight = minHeight;
        modulesInWidth                          =                          (int)
```

```
mainCanvas.GetComponent<RectTransform>().rect.width/minHeight;
                modulesInHeight                                    =                    (int)
mainCanvas.GetComponent<RectTransform>().rect.height/minHeight;
                totalModules = modulesInWidth*modulesInHeight;
        }
        public void showDiagramedCanvas()
        {
                beginCanvas.GetComponent<Canvas>().enabled = false;
                headerMenuCanvas.GetComponent<Canvas>().enabled = false;
                footerMenuCanvas.GetComponent<Canvas>().enabled = false;
                continueMenuCanvas.GetComponent<Canvas>().enabled = false;
                GetComponent<gVar>().txt.GetComponent<Text>().text = "";
        }
}
```