Universidad Zaragoza

Proyecto Fin de Carrera

Desarrollo de una aplicación móvil para la asistencia de personas con visión daltónica

Development of a mobile application for the assistance of colorblind people

Autor

José María López Tella

Director

Gonzalo López Nicolás

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2017

i. Agradecimientos

A mis padres, por todo lo que me han dado y lo poco que he hecho para merecerlo.

A mis hermanos, por la aventura que ha sido crecer a vuestro lado.

A ti, Sergio, por dejarme estar siempre a tu lado, y por la paciencia que tienes conmigo.

A mis amigos y seres queridos, por estar siempre ahí.

ii. Tabla de contenido

1. Resumen

El daltonismo es una alteración de tipo genético que afecta a la capacidad de distinguir los colores, y que padecen alrededor de un 8% de los hombres y sólo un 0,5% de las mujeres. Se produce cuando uno o más tipos de conos (células sensoriales responsables de la visión del color) están ausentes, no funcionan correctamente, o detectan un color diferente al normal.

Hoy en día, la práctica totalidad de los dispositivos de telefonía móvil disponen de una cámara, así se propone emplear esta tecnología para ayudar a las personas que padecen daltonismo a diferenciar unos colores de otros a través de una aplicación. El objetivo principal de este proyecto es desarrollar una aplicación para teléfonos móviles de asistencia a personas daltónicas para discernir apropiadamente los colores mediante la transformación de las imágenes proporcionadas por la cámara.

La aplicación que se propone desarrollar en este proyecto, llamada *Colorblind Helper*, se ha diseñado y programado según la arquitectura UWP (*Universal Windows Platform*), creada por Microsoft y compatible con todos los dispositivos que funcionen con Windows 10. La parte visual estará programada en XAML (*eXtensible Application Markup Language*), mientras que el funcionamiento de la aplicación se realizará utilizando C#. Todo el desarrollo se ha realizado en *Microsoft Visual Studio 2017 Community Edition*, un entorno de programación gratuito orientado a pequeños proyectos.

Partiendo de un análisis del estado de la materia en cuanto a aplicaciones existentes relacionadas con el tema abordado, se definieron las funcionalidades básicas de la nuestra, en particular: (1) modificar en tiempo real el color de las imágenes captadas por la cámara del dispositivo, (2) calibrar la modificación de los colores en función del tipo de daltonismo del usuario, (3) guardar dicha calibración del usuario y (4) visualizarse correctamente con independencia del dispositivo utilizado.

El diseño de alto nivel del funcionamiento del programa es la base para realizar el prototipado de cada una de las pantallas de la aplicación, lo que nos ha permitido desarrollar posteriormente el código necesario. La evaluación funcional de la aplicación ha sido verificada por terceras partes, mediante la comprobación de su correcto funcionamiento en diferentes dispositivos. Adicionalmente, se envió la aplicación a Microsoft para su evaluación y publicación.

El resultado de este proyecto es una aplicación que estará disponible de forma gratuita para todos los usuarios a través de la Tienda de Microsoft (https://www.microsoft.com/store/apps/9MT0ZT0PS0SH). Este entorno permitirá una mayor difusión de la aplicación desarrollada, y los comentarios y opiniones de los usuarios servirán para plantear funcionalidades adicionales en futuras versiones.

2. Introducción

Motivación

El daltonismo es una alteración de tipo genético que afecta a la capacidad de distinguir los colores, y que padecen aproximadamente un 8% de los hombres (entre ellos, el autor del presente proyecto) y sólo un 0,5% de las mujeres. La mayoría de las personas daltónicas ven las cosas tan bien como los demás, pero no son capaces de percibir correctamente la luz roja, verde o azul.

No es posible lograr que un daltónico pueda ver exactamente igual que una persona que no padece esta enfermedad, pues sus células fotosensibles no perciben determinadas longitudes de onda. Esto hace que la gama de colores que pueden ver se reduzca. Sin embargo, hoy en día cualquier persona dispone de un dispositivo móvil con cámara, así que vamos a emplear esta tecnología para ayudar a las personas que sufren daltonismo a diferenciar unos colores de otros.

Objetivos

El principal objetivo del presente proyecto será desarrollar una aplicación para dispositivos móviles que permita a un daltónico distinguir colores. Emplearemos la cámara del dispositivo para, modificando los colores que muestra en pantalla, ayudar a la persona a diferenciar colores que de otro modo vería iguales.

Esta aplicación se pondrá a disposición de los usuarios de forma completamente gratuita, y se irá modificando con el paso del tiempo para añadir nuevas funcionalidades.

Alcance

El proyecto se centrará en el desarrollo de la aplicación, desde la idea inicial hasta su validación y publicación en la tienda de aplicaciones del dispositivo.

No se incidirá en la teoría que hay detrás de la modificación de los colores para que un daltónico pueda diferenciarlos mejor, aunque sí que se describirá brevemente para poder tener la base matemática necesaria para la creación del código.

Estructura de la memoria

Partiendo de una introducción sobre daltonismo y el funcionamiento de la visión, la memoria sigue todos los pasos del desarrollo de la aplicación. Comenzaremos seleccionando la plataforma para la que se desarrollará y el entorno de programación que utilizaremos. Seguidamente definiremos las funcionalidades de la aplicación y cómo se va a estructurar visualmente cada una de las pantallas de la misma. Esta información nos servirá para desarrollar posteriormente el código necesario, que será testeado por terceras partes. Por último, se enviará la aplicación para su evaluación y publicación.

Estado de la materia

Antes de comenzar con el desarrollo, se han analizado otras aplicaciones para ayudar a las personas con daltonismo. De todas ellas se han obtenido ideas para posibles funcionalidades. Algunas de ellas se implementarán, como el centrar los tipos de daltonismo en los más representativos. Otras, sin embargo, se desviaban demasiado de nuestro objetivo y quedarán pendientes para futuras versiones, como por ejemplo un simulador de visión daltónica para no daltónicos.

3. Daltonismo

3.1. Introducción

Al hablar de daltonismo pensamos generalmente en la incapacidad para distinguir el rojo y el verde. Se trata, sin embargo, de un error muy común, aunque sí que es cierto que esta incapacidad es la más común de las deficiencias visuales de color. En realidad, el daltonismo es una alteración genética que afecta a la capacidad de distinguir los colores.

La palabra *daltonismo* proviene del químico y matemático John Dalton, que fue el primero que describió[1] esta alteración en el año 1794, tras descubrir que él mismo lo era. Otros muchos científicos se interesaron por el tema de la visión del color, como Young, Helmholtz, Maxwell y Wilson; que allanaron el camino para comprender la percepción y la percepción errónea de los colores.

3.2. Funcionamiento de la visión

Para comprender el origen de las deficiencias visuales de color, es importante conocer antes cómo funciona nuestra visión.

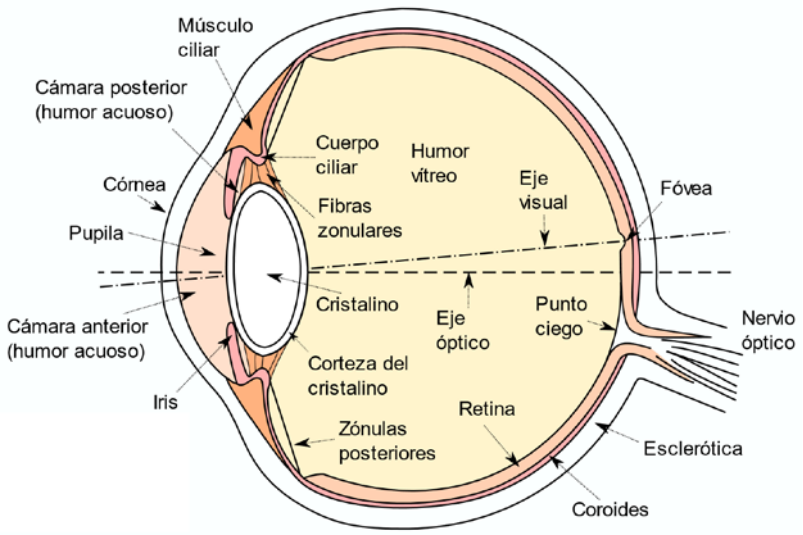


Ilustración 1: Sección del ojo humano

El ojo humano posee una lente, llamada cristalino, que permite enfocar objetos situados a diferentes distancias; un "diafragma" o pupila cuyo diámetro está regulado por el iris y un tejido sensible a la luz que es la retina. La luz penetra a través de la pupila, atraviesa el cristalino

[1] (Dalton, 1794)

y se proyecta sobre la retina, donde se transforma gracias a unas células llamadas fotorreceptoras. Éstas pueden ser de dos tipos:

- Conos: son los responsables de la visión del color, y funcionan mejor de día o en ambientes bien iluminados. Esta es la razón por la que de noche no somos capaces de apreciar los colores.
- Bastones: se activan en niveles bajos de iluminación, y sólo permiten distinguir el negro, el blanco y los distintos grises. Con ellos se percibe el contraste.

Existen tres tipos de conos, cada uno de ellos sensible a una longitud de onda diferente:

- Rojos: sensibles a longitudes de onda larga, con un pico de sensibilidad en 560nm.
- Verdes: sensibles a longitudes de onda media, con un pico de sensibilidad en 530nm.
- Azules: sensibles a longitudes de onda corta, con un pico de sensibilidad en 420nm.

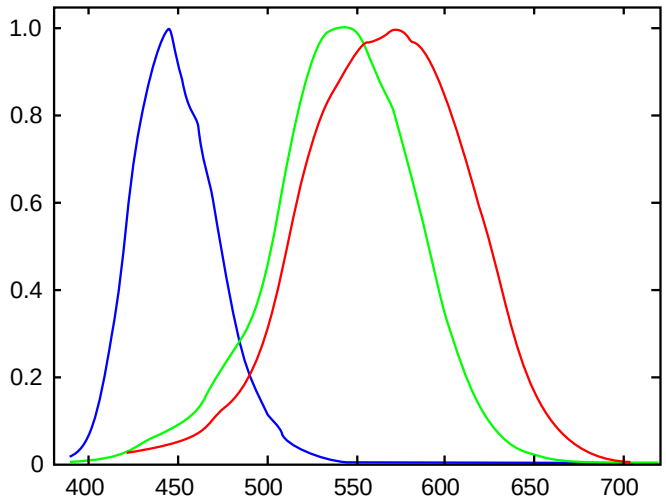


Ilustración 2: Curvas de sensibilidad de los tres conos humanos, asociados a los tres colores primarios.
Autor: Vanessa Ezekowitz / en.wikipedia (CC BY-SA 3.0)

3.3. Deficiencias visuales del color

El daltonismo, por tanto, ocurre cuando uno o más tipos de conos están ausentes, no funcionan correctamente, o detectan un color diferente al normal. Puede ser leve si sólo un tipo de cono está afectado, o más severo cuando hay una ausencia de los tres tipos, por lo que todo se ve en tonos grises.

De mayor a menor gravedad, se distinguen los siguientes tipos de deficiencias:

Acromático

Una persona con daltonismo acromático ve todo en blanco y negro. No percibe color alguno, ya sea porque no tiene ninguno de los tres tipos de conos o por razones neurológicas.

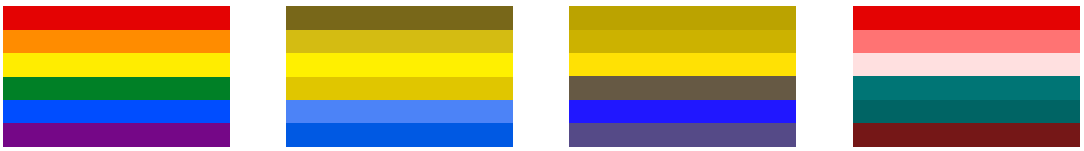
Monocromático

Se presenta cuando únicamente existe uno de los tres tipos de conos, quedando la visión de la luz y el color reducida a una dimensión.

Dicromático

El dicromatismo es un defecto moderadamente grave en el cual hay una disfunción de uno de los tres conos. Puede ser de tres tipos diferentes:

- Protanopía: consiste en la ausencia total de los conos rojos.
- Deuteranopía: debida a la ausencia de conos verdes.
- Tritanopía: situación muy poco frecuente en la que están ausentes los conos azules.



Visión normal | Protanopía | Deuteranopía | Tritanopía

Tricromático anómalo

Es el grupo más abundante y común de daltónicos. Poseen los tres tipos de conos, pero con modificaciones funcionales, por lo que se confunde un color con otro. Suelen tener percepciones similares a los daltónicos dicromáticos, pero menos notables. Se dan tres tipos:

- Protanomalía: existe una reducción en el pico de sensibilidad de los conos rojos.
- Deuteranomalía: es la más usual, y se da cuando el pico de sensibilidad de los conos verdes varía hacía el espectro del rojo.
- Tritanomalía: es muy infrecuente, y ocurre cuando el pico de sensibilidad de los conos azules varía hacía el espectro del verde.

Visión normal | Protanomalía | Deuteranomalía | Tritanomalía

3.4. Genética del daltonismo

El daltonismo es hereditario y su transmisión está ligada al cromosoma X. Si un varón hereda un cromosoma X alterado será daltónico. En cambio, las mujeres, al poseer dos cromosomas X, sólo serán daltónicas si sus dos cromosomas X están alterados. Esta es la razón por la que el daltonismo afecta aproximadamente al 8% de los hombres y solo al 0,5% de las mujeres.

El daltonismo afecta normalmente a ambos ojos por igual, y se mantiene estable a lo largo de la vida. El diagnóstico se suele realizar mediante con el test de color de Ishihara (ver ilustración 2), y actualmente no existe tratamiento.

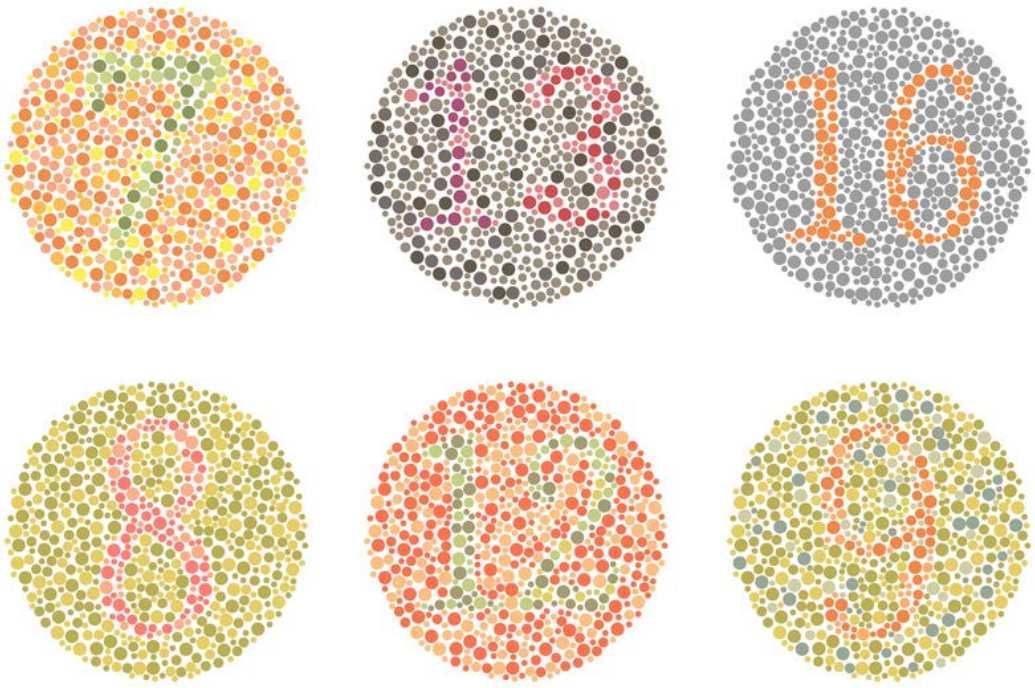


Ilustración 3: El test de color de Ishihara consta de 38 cartas, cada una de las cuales contiene círculos formados por puntos de colores y tamaños aleatorios. En el patrón de puntos se forma un número visible para aquellos con visión normal e invisible o difícil de ver para aquellos con daltonismo. (Cartas de Ishihara, s.f.)

4. Selección de plataforma de desarrollo

En la actualidad, entre los sistemas operativos para móviles destacan tres: Android, (desarrollado por Google), iOS (creado por Apple) y Windows Phone (de Microsoft). La elección de la plataforma podría hacerse únicamente en función de la cuota de mercado de cada sistema, pero resulta más coherente tener en cuenta otras variables. Las más importantes se muestran en la siguiente tabla:

	Android	**iOS**	**Windows 10**
Coste licencia desarrollo[2]	25$	99$ /año	19$
Coste equipo desarrollo	Cualquier ordenador	Sólo ordenadores con SO Mac	Sólo ordenadores con SO Windows
Lenguaje programación	Java	Objective-C	C++, C# o VB.NET
Cuota mercado (móvil, Tablet y sobremesa)[3]	41,37%	13,24%	13,00%
Aplicaciones disponibles	Más de 100	Más de 100	4

El no disponer de un ordenador Mac implica que el desarrollo para iOS queda descartado. Parece, pues, que Android debería ser la plataforma elegida. Sin embargo, el elevado número de aplicaciones disponibles hace que la utilidad de la que vamos a crear quede reducida. Esto, unido a la preferencia del autor por C#, hace que también Android quede descartado. Por tanto, la aplicación se desarrollará para **Windows 10**.

[2] Datos obtenidos de las respectivas páginas de los fabricantes.
[3] (StatCounter, 2017)

5. Elección del entorno de programación

El desarrollo de aplicaciones para Windows 10 se realiza típicamente en *Microsoft Visual Studio*, un entorno integrado de desarrollo (IDE, por sus siglas en inglés). Usaremos la versión gratuita del mismo, llamado *Community Edition 2017*, orientada a desarrolladores individuales o pequeños equipos.

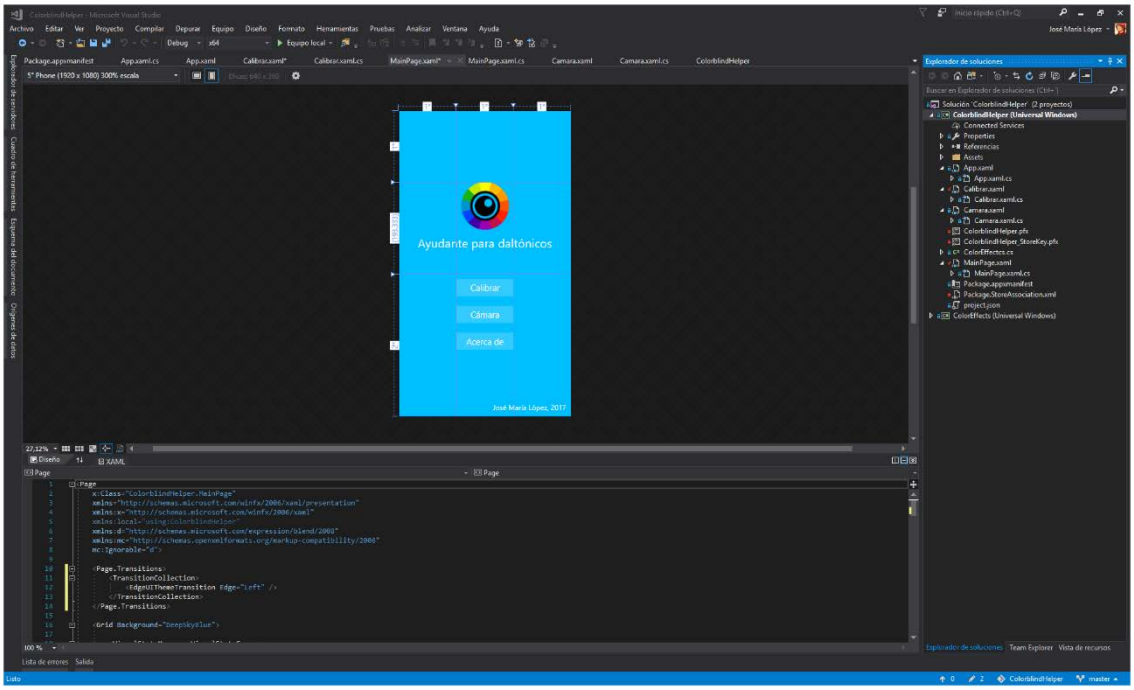


Ilustración 4: Pantalla del proyecto desarrollado en el entorno de Microsoft Visual Studio 2017

Para el desarrollo de la aplicación se seguirán las directrices de la Plataforma Universal de Windows creada por Microsoft (UWP, por sus siglas en inglés), una arquitectura que permite desarrollar aplicaciones universales que funcionan en Windows 10, Windows 10 Mobile, Xbox One y Microsoft HoloLens sin que sea necesario realizar modificaciones.

Desde el comienzo hemos considerado que los dispositivos móviles son los más adecuados para utilizar la aplicación que vamos a desarrollar. Sin embargo, y dado el escaso trabajo que hay que realizar para que funcione también en otro tipo de dispositivos, permitiremos utilizar la aplicación en ordenadores con Windows 10.

6. Determinación de las funcionalidades

Antes de desarrollar la aplicación, es necesario definir qué es lo que tiene que hacer y cómo tiene que hacerlo. Esta información de las funcionalidades de la aplicación se recogerá en el "pre-análisis funcional", un documento que nos servirá de base para los siguientes pasos del desarrollo.

En nuestro caso, se contemplan cuatro funcionalidades básicas:

1) Modificar en tiempo real el flujo de imágenes capturado por la cámara del dispositivo móvil, de manera que se alteren los colores para permitir al usuario diferenciar unos colores de otros.

2) Dada la existencia de distintas clases de daltonismo, la aplicación tendrá que ser capaz de cubrir las tipologías más comunes. Por tanto, será posible calibrarla en función del tipo de daltonismo del usuario.

3) Guardar la calibración del usuario, de modo en los siguientes usos no haga falta volver a realizarla.

4) La aplicación se tendrá que visualizar correctamente independientemente del dispositivo en el que se ejecute.

Cada una de estas funcionalidades lleva consigo una serie de necesidades técnicas que ha de cumplir la aplicación:

- Acceso a la cámara del dispositivo, y gestión de los errores derivados de que no haya una disponible.
- Aplicación de efectos de modificación de color en tiempo real, tanto para imágenes estáticas como para vídeo.
- Acceso al sistema de almacenamiento del dispositivo para guardar información de la configuración.
- Gestión de la información sobre el tamaño de la pantalla y la orientación del dispositivo, de modo que el diseño de las diferentes pantallas de la aplicación sea correcto en todo momento.

7. Diseño de la aplicación

Una vez elaborado el pre-análisis funcional, es necesario elaborar un prototipo inicial de la aplicación. Este prototipo contendrá una descripción visual de cada una de las pantallas que compondrán la aplicación, lo que nos permitirá estimar y organizar de antemano todo el código que será necesario desarrollar.

Existen multitud de aplicaciones para realizar este prototipado, y en nuestro caso se ha utilizado una llamada *moqups* (http://moqups.com/), que posee una versión gratuita con todas las funcionalidades y alguna limitación (que a nosotros no nos afecta). El único detalle que cabe mencionar es que *moqups* no está pensado para diseñar aplicaciones para Windows 10, puesto que carece de recursos gráficos de este sistema operativo.

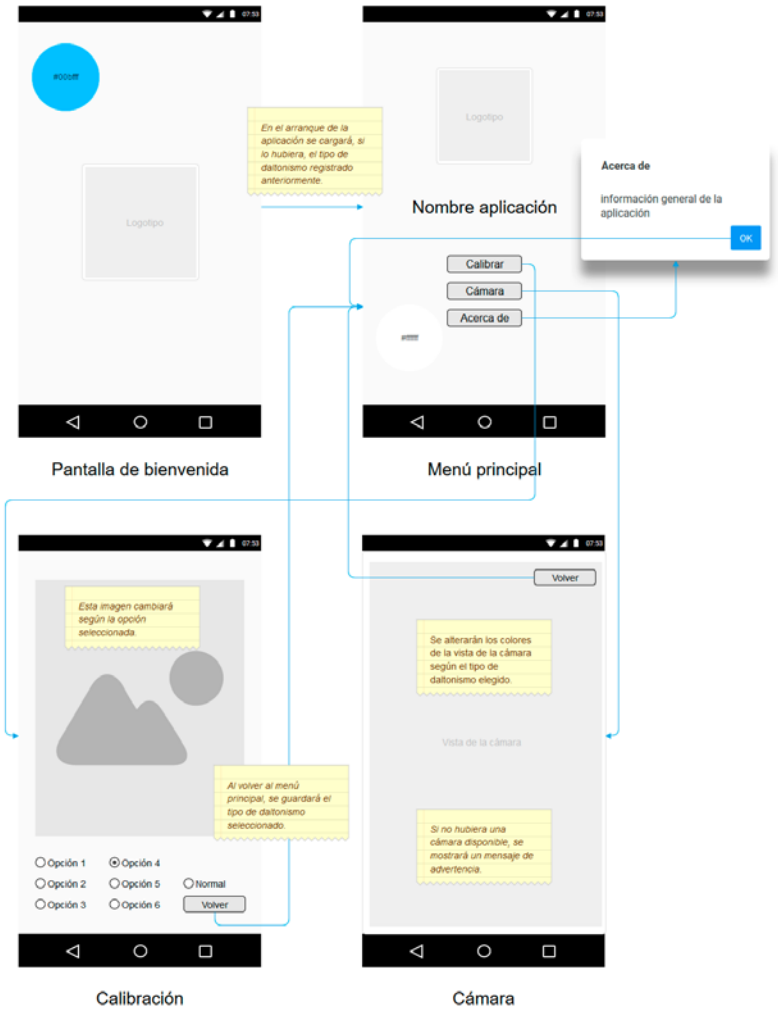
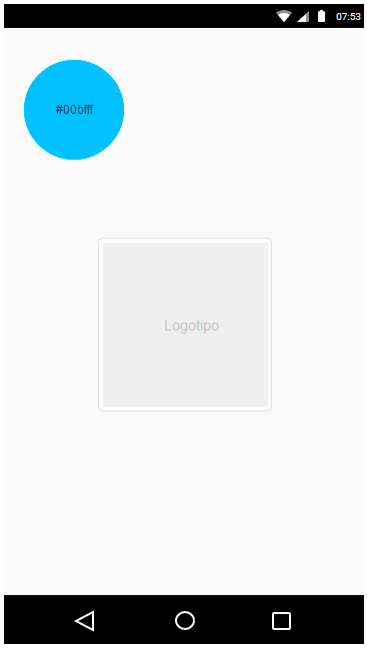


Ilustración 5: Prototipo de nuestra aplicación diseñado en moqups*. Nótese que los elementos gráficos (pantallas, botones, mensajes…) corresponden al sistema operativo Android.*

Describiremos a continuación con detalle cada una de las pantallas de la aplicación:



7.1. Pantalla de bienvenida

Es la primera pantalla que se mostrará nada más ejecutar la aplicación.

Aspecto:

La pantalla de bienvenida estará compuesta únicamente por el logotipo de la aplicación sobre fondo azul (*deepBlueSky* en Visual Studio, o #00bbff en RGB).

Programación necesaria:

La pantalla de bienvenida no necesita código, pues se genera automáticamente a partir del logotipo de la aplicación y el color de fondo que hayamos elegido para toda la aplicación.

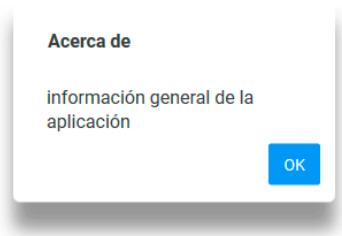
7.2. Menú principal

Desde aquí se gestionará el uso de la aplicación.

Aspecto:

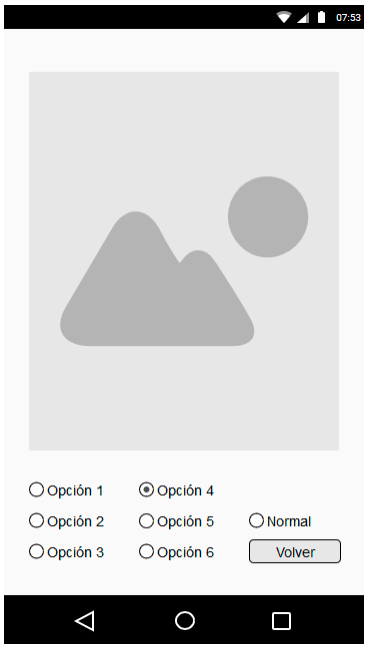
El icono de la aplicación aparecerá en la parte superior, acompañado del nombre. Debajo, tres botones (en color blanco, o #ffffff RGB) nos darán acceso a las principales funcionalidades: calibrar, cámara y acerca de. Estos botones se mostrarán apilados si la orientación del dispositivo es vertical, y alineados si es horizontal. De esta manera se aprovechará de la forma más eficiente el espacio disponible.

Programación necesaria:



Primero, comprobaremos si hay algún dato de calibración previo, en cuyo caso lo cargaremos. Si no existiera, lo inicializaremos como visión “normal”. El botón “calibrar” y “cámara” nos llevarán a las siguientes pantallas, mientras que el de “acerca de” mostrará información general de la aplicación.

7.3. Calibración



Desde esta pantalla podremos determinar el tipo de daltonismo que más se ajusta a nuestro caso.

Aspecto:

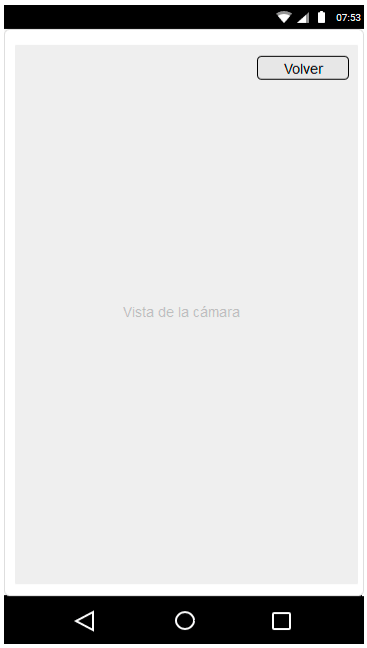
En la parte superior de la pantalla se mostrarán diferentes cartas de Ishihara, correspondientes a los tipos de daltonismo más comunes. En la parte inferior (si la orientación del dispositivo es vertical) o a la derecha (si la orientación es horizontal) se mostrarán en formato botón radial (*radio button* en inglés) varias opciones disponibles, y sólo podremos elegir una. Cada vez que elijamos una, la imagen superior modificará sus colores para facilitar la identificación de las cartas de Ishihara. Existirá también la opción de seleccionar la visión "normal" y volver al menú principal con el botón correspondiente.

Programación necesaria:

Habrá que capturar la pulsación de cada uno de los botones para modificar el color de la imagen superior en consecuencia. El procedimiento de modificación será similar en todos los casos, y sólo cambiarán los valores del cambio de color.

Grabaremos en el sistema de almacenamiento del dispositivo la opción seleccionada únicamente en el momento que el usuario pulse el botón "volver" para regresar al menú principal. De este modo reduciremos el número de accesos al almacenamiento que, aunque no suponga un menor consumo de recursos, es preferible desde el punto de vista del desarrollo.

7.4. Cámara



Esta es la pantalla más importante de la aplicación, la que modifica los colores de lo que ve la cámara para que el usuario pueda diferenciarlos con mayor facilidad.

Aspecto:

Toda la pantalla será ocupada por lo que ve la cámara, y sólo existirá un botón en la parte superior para regresar al menú principal.

Programación necesaria:

Lo primero que habrá que comprobar al cargar esta pantalla es si existe una cámara disponible. Aunque lo explicaremos con más detalle en secciones posteriores, configuraremos la aplicación para que no sea posible instalarla en dispositivos sin cámara. Sin embargo, aun habiendo una, puede que el usuario no dé permiso para utilizarla, o se encuentre desconectada en ese momento (como puede ser el caso, por ejemplo, de ordenadores de sobremesa con cámara externa). En tal caso se mostrará un mensaje de advertencia.

Lo siguiente que haremos será cargar el tipo de daltonismo configurado, y aplicaremos a la cámara la modificación de colores correspondiente.

El botón “volver”, una vez más, nos llevará de nuevo a la pantalla principal.

7.5. Icono de la aplicación

Adicionalmente, es necesario establecer un icono para la aplicación, que se mostrará tanto en los dispositivos donde esté instalado como en la Tienda de Microsoft. El diseño partió de una imagen algo compleja que representa un ojo humano cuyo iris se mezcla con un círculo cromático (ver ilustración 6), que buscaba representar la visión a todo color. Su estilo, no obstante, contrastaba con la idea de sencillez que recomienda Microsoft. Además, la gran gama cromática de esta imagen implicaba que su tamaño tenía que ser menor para no superar el peso máximo permitido.

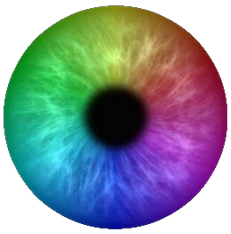


Ilustración 6: Versión inicial del icono de la aplicación

Ilustración 7: Versión final del icono de la aplicación

Pasamos de este modo a icono más sencillo, de colores planos pero que transmite una idea similar (ver ilustración 7). Tiene un peso mucho menor comparado con la versión inicial a igual tamaño, pues hemos pasado de 2.390 KB a 119 KB.

8. Desarrollo del código

8.1. Consideraciones generales

El desarrollo de la aplicación se estructurará siguiendo las pantallas del prototipo. En cada una habrá que desarrollar el aspecto visual y el código. En la parte visual, que se programará en XAML (*eXtensible Application Markup Language*, Lenguaje Extensible de Formato para Aplicaciones en español) nos centraremos en la composición de las características gráficas de cada pantalla, haciendo hincapié en su correcta visualización independientemente del tipo de dispositivo o la orientación del mismo. El código, que se realizará en C#, recogerá el funcionamiento de cada pantalla, siguiendo el ciclo de vida de la aplicación.

Aunque el autor tenga algo de experiencia como programador, esta es la primera aplicación UWP que desarrolla. Por tanto, antes de comenzar con el desarrollo de la aplicación será necesario aprender las bases de XAML y C#.

Microsoft ofrece a través de la *Microsoft Virtual Academy*[4] multitud de cursos gratuitos para, entre otras cosas, aprender a programar y desarrollar aplicaciones UWP para Windows 10. Estos cursos aunaban pequeños vídeos explicativos, evaluaciones periódicas y desafíos personales en forma de aplicaciones a desarrollar.

En nuestro caso, hemos empleado los siguientes:

- Fundamentos de C# para absolutos principiantes.
- Desarrollo en Windows 10 para absolutos principiantes.

El primero nos sirvió para conocer las bases de C#, mientras que con el segundo descubrimos XAML y cómo se conjuga con C# para crear aplicaciones de Windows 10.

Adicionalmente, se han empleado otros recursos disponibles en Internet:

- Centro de desarrollo de Windows[5], con toda la información necesaria para desarrollar aplicaciones de Windows 10, así como ejemplos prácticos.

[4] https://mva.microsoft.com
[5] https://docs.microsoft.com

- *Stack Overflow*[6], una comunidad de desarrolladores donde se pueden plantear y encontrar soluciones a problemas de programación en diferentes lenguajes.

Por último, la gestión del código se hizo a través de *Visual Studio Team Services*[7], un servicio de Microsoft (con opción gratuita para equipos de hasta 5 personas) que permite, entre otras cosas, realizar el control de versiones.

8.2. Implementación

Aunque no se considera necesario explicar todo el código que se desarrolló, sí que es importante comentar algunos aspectos del mismo:

8.2.1. Representación del color

En informática, el color de cada pixel de una imagen se representa a través de sus valores RGB, que corresponden a los colores primarios rojo, azul y verde (RGB viene de sus iniciales en inglés: Red, Green, Blue). El modelo RGB está basado en la síntesis aditiva, por lo que cada color se representa como la suma de los tres colores primarios.

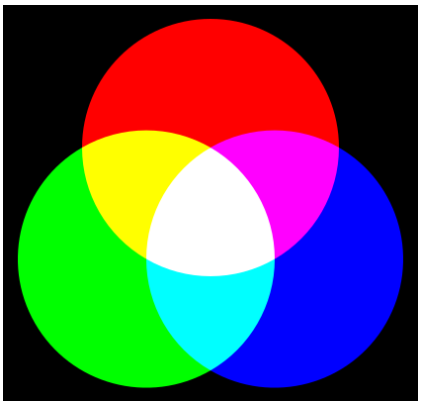


Ilustración 8: Modelo aditivo de colores rojo, verde y azul

Es habitual codificar estos valores RGB con un byte (8 bits), por lo que oscilarán entre 0 (ausencia del color) y 255 (intensidad máxima del color). Así, por ejemplo, color negro se representa como la ausencia de color (0, 0, 0), mientras que el blanco es el valor máximo de los tres colores primarios (255, 255, 255).

Es habitual añadir a este modelo de color una canal alfa de transparencia, lo que nos permite indicar no sólo el color de cada pixel, sino su opacidad. Un canal alfa con valor 0 implicará que ese pixel es completamente transparente, mientras que un valor de 25 hará que

[6] https://stackoverflow.com/
[7] https://www.visualstudio.com/es/team-services/

el pixel sea completamente opaco. La representación usando estos 4 canales se denomina RGBA, y es la que usaremos internamente en nuestra aplicación.

Ilustración 9: Modelo de color RGBA sobre fondo blanco para apreciar el canal de transparencia

8.2.2. Corrección del color

La principal funcionalidad de la aplicación se realiza a través de la modificación de los canales de color RGB (rojo, verde y azul, por sus iniciales en inglés) de la imagen / vídeo a través de una matriz de transformación. Se obtiene el nuevo vector de color empleando una matriz de transformación:

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Esta matriz está basada en la librería *ColorMatrix*[8], creada de forma colaborativa para realizar todo tipo de modificaciones de color. En concreto, usaremos las siguientes para cada tipo de daltonismo:

$$\begin{bmatrix} 0{,}5667 & 0{,}4333 & 0 \\ 0{,}5583 & 0{,}4417 & 0 \\ 0 & 0{,}2417 & 0{,}7583 \end{bmatrix}$$

Protanopía

$$\begin{bmatrix} 0{,}6250 & 0{,}3750 & 0 \\ 0{,}7000 & 0{,}3000 & 0 \\ 0 & 0{,}3000 & 0{,}7000 \end{bmatrix}$$

Deuteranopía

$$\begin{bmatrix} 0{,}9500 & 0{,}0500 & 0 \\ 0 & 0{,}4333 & 0{,}5667 \\ 0 & 0{,}4750 & 0{,}5250 \end{bmatrix}$$

Tritanopía

$$\begin{bmatrix} 0{,}8167 & 0{,}1833 & 0 \\ 0{,}3333 & 0{,}6667 & 0 \\ 0 & 0{,}1250 & 0{,}8750 \end{bmatrix}$$

Protanomalía

$$\begin{bmatrix} 0{,}8000 & 0{,}2000 & 0 \\ 0{,}2583 & 0{,}7417 & 0 \\ 0 & 0{,}1417 & 0{,}8583 \end{bmatrix}$$

Deuteranomalía

$$\begin{bmatrix} 0{,}9667 & 0{,}0333 & 0 \\ 0 & 0{,}7333 & 0{,}2667 \\ 0 & 0{,}1833 & 0{,}8167 \end{bmatrix}$$

Tritanomalía

[8] http://web.archive.org/web/20081104142003/http://www.nofunc.com:80/Color_Matrix_Library/

Esta conversión del color hará que la imagen se modifique de la siguiente manera:

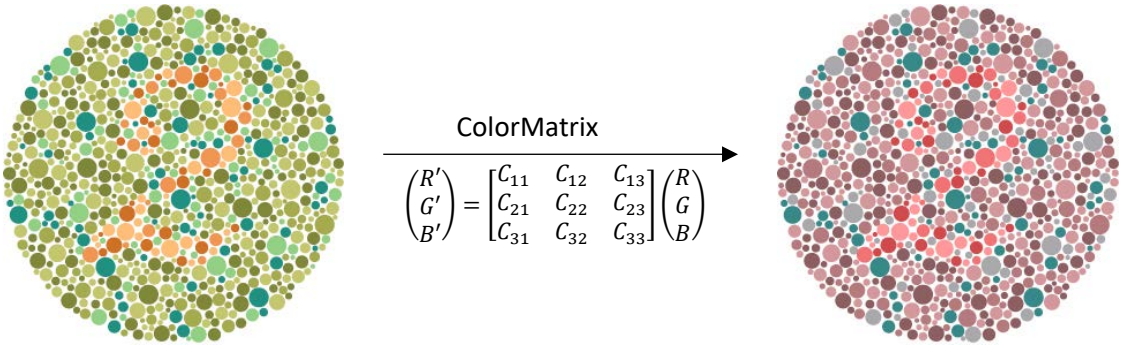


Ilustración 10: Simulación de la matriz de transformación ColorMatrix *para corregir la tritanopía. Se observa que el número 2 se muestra más claramente en la imagen de la derecha.*

8.2.3. Modificación del color en imágenes

La corrección del color se realiza empleando *Win2D*[9], una librería de Microsoft muy fácil de utilizar que permite el renderizado de gráficos en 2D empleando la aceleración gráfica del dispositivo.

Esta librería tiene una función llamada *ColorMatrixEffect*, que modifica los colores de una imagen mediante una matriz de transformación de 5x4. Nuestra matriz de transformación es de 3x3, pues sólo modifica los valores correspondientes a los canales de color primarios RGB. Así que tendremos que ampliarla a 4x4 considerando también el canal de transparencia, al que le asignaremos el valor 1, pues no se va a modificar. Luego sólo queda añadir una fila más (vacía en nuestro caso, pues tampoco se utilizará) hasta obtener la matriz 5x4 deseada.

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \xrightarrow{Canal\ alfa} \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 \\ C_{21} & C_{22} & C_{23} & 0 \\ C_{31} & C_{32} & C_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{Fila\ adicional} \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 \\ C_{21} & C_{22} & C_{23} & 0 \\ C_{31} & C_{32} & C_{33} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

El vector resultante tendrá 5 componentes, y sólo habrá que descartar el quinto para obtener el nuevo valor RGBA de cada pixel.

8.3. Creación del manual del usuario

La aplicación se ha diseñado para ser utilizada tanto por usuarios nóveles como expertos en el manejo de dispositivos móviles. Sin embargo, y para que todos ellos puedan sacarle el máximo partido, se elaborará un manual de funcionamiento. Dicho manual se mostrará al

[9] https://microsoft.github.io/Win2D/html/Introduction.htm

arrancar por primera vez la aplicación, aunque podrá ser consultado en todo momento desde el menú principal.

En el anexo 1 del presente informe se recoge el manual completo.

9. Pruebas de funcionamiento

Una vez se hayan implementado las distintas pantallas de la aplicación, se evaluará una versión preliminar por terceras partes para comprobar su correcto funcionamiento y depurar posibles errores. Para ello, usaremos los llamados “paquetes piloto”, una característica de la Tienda de Windows que nos permiten distribuir paquetes diferentes sólo para un grupo de prueba limitado. Los paquetes piloto han de pasar también el proceso de certificación, exactamente igual que una versión “normal”. Por seguir un orden en el informe, el proceso de certificación se tratará más adelante, cuando ya tengamos lista la aplicación.

El grupo de prueba se crea directamente desde la web de desarrolladores, y sólo se necesitan las direcciones de correo de los participantes. Estas direcciones han de corresponder a cuentas de Microsoft, pues es necesario identificarse debidamente en la Tienda para descargar las versiones de evaluación. Las personas incluidas en el grupo piloto recibirán automáticamente las nuevas actualizaciones en sus dispositivos, exactamente igual que sucede con el resto de aplicaciones de la Tienda de Microsoft.

Nuestra aplicación no es muy compleja ni posee excesivas funcionalidades, por lo que no se necesitaba un grupo de pruebas excesivamente grande. Estuvo compuesto por amigos y conocidos que poseyeran equipos con Windows 10, y su tarea consistió en verificar que el funcionamiento de la aplicación fuera el previsto y que se visualizara adecuadamente. También se les pidió que sugirieran modificaciones o nuevas funcionalidades.

Dos fueron los paquetes piloto que se lanzaron: el primero para verificar el funcionamiento de la calibración y la cámara y el segundo para analizar la versión final de la aplicación. Se indica a continuación los problemas detectados y la solución desarrollada:

Problema	Solución
Incorrecta visualización del texto de algunos botones en función de la orientación del dispositivo.	Se modificó el tamaño de los botones y de la fuente de texto empleada.
Ausencia de botones de navegación en la versión de escritorio.	Al tratarse de una aplicación móvil en su concepción, no se tuvo en cuenta que los móviles cuentan con botones de navegación y los equipos de sobremesa no. La solución pasó por implementar un botón para retroceder en la parte superior izquierda de la ventana, que sólo aparece en este tipo de equipos. Este funcionamiento es el empleado por Microsoft en sus aplicaciones UWP.
Imagen de la cámara borrosa al acercar mucho el móvil al objeto.	Se programó la opción enfoque automático (autofocus) marcando la opción correspondiente en la vista de la cámara. Como hay cámaras que no permiten este enfoque automático, esta opción no se muestra en dispositivos no compatibles.

Se propusieron también nuevas funcionalidades, que se recogen más adelante en el epígrafe “Conclusiones”.

10. Publicación de la aplicación

Una vez lista nuestra aplicación, el paso final es publicarla en la Tienda de Windows, para que cualquier persona pueda descargarla y utilizarla en su dispositivo con Windows 10. Como hemos comentado anteriormente, los paquetes piloto necesarios para la realización de pruebas de la aplicación también están sujetos al proceso de certificación. Como el proceso es exactamente el mismo, vamos a incidir únicamente en la publicación de la versión final de la aplicación.

El proceso de publicación es algo laborioso, y se gestiona a través de la página web de desarrolladores de Microsoft[10]. Consta de los siguientes pasos:

10.1. Creación de una cuenta de desarrollador

Sólo los desarrolladores registrados pueden enviar sus aplicaciones a la Tienda de Windows, por lo que registrarse será lo primero que hagamos. Existen dos tipos de cuentas: individuales (con un coste de 19$) y empresariales (con un coste de 99$). El pago es único y no es necesario renovarlo. En nuestro caso, lógicamente, crearemos una cuenta individual.

Entre todos los datos que hay que cumplimentar, son especialmente críticos los detalles financieros. Según el IRS[11] (Sistema de Impuestos Internos, por sus siglas en inglés), una suerte de Agencia Tributaria de Estados Unidos, al ser Microsoft una empresa con sede en EE. UU. (y ser la que gestiona todas las transacciones de la Tienda de Windows), todos los posibles ingresos que tuviéramos se considerarían Ingresos Efectivamente Conectados (ECI, por sus siglas en inglés), y tendrían que ser declarados y gravados en función del número de transacciones realizadas. Esta declaración no aplica para aplicaciones gratuitas, así que no incidiremos más en este aspecto dado que no pretendemos cobrar por nuestra aplicación.

10.2. Reserva del nombre de la aplicación

Reservar un nombre es el primer paso para crear una aplicación. La reserva tiene una duración de 1 año, e incluso es posible hacerla antes de tenerla lista para asegurarnos cuanto antes de que nadie más pueda usar el nombre. Es posible incluso reservar varios nombres para utilizarlos en distintos idiomas de la aplicación. En nuestro hemos reservado “Colorblind Helper” y “Ayudante para daltónicos”, en previsión de futuras versiones en castellano e inglés.

[10] https://developer.microsoft.com/es-es/
[11] (IRS, 2017)

10.3. Envío de la aplicación

Además del paquete de la aplicación, se necesita información adicional sobre la aplicación para poder publicarla. Entre otros, estos son los detalles que hay que cumplimentar:

- Precios y disponibilidad: indicaremos que la aplicación será gratuita o que se publicará en los 242 mercados disponibles en el mundo.
- Propiedades: aquí se recogerá que la categoría de nuestra aplicación es "Salud y bienestar" y que no se podrá instalar en dispositivos sin cámara.
- Clasificación por edades: a través de una serie de preguntas sobre el contenido y funcionamiento de la aplicación se asignará una clasificación por edades. En nuestro caso obtuvimos la clasificación +3 de PEGI[12], que es el sistema que se utiliza en toda Europa, aunque se generaron clasificaciones para todos los mercados disponibles.
- Descripciones de la Tienda: toda la información que se mostrará en la página de nuestra aplicación se indicará aquí, como por ejemplo la descripción, notas de la versión o capturas de pantalla.
- Paquetes: aquí cargaremos como tal nuestra aplicación e indicaremos las familias de dispositivos en las que estará disponible. Tal y como la hemos desarrollado, nuestra aplicación estará disponible en ordenadores y móviles con Windows 10. También podría estar disponible en dispositivos holográficos, como las Microsoft HoloLens, pero el hecho de no disponer de unas para realizar las pruebas correspondientes nos ha echado atrás.

10.4. Certificación de la aplicación

Una vez cumplimentada toda la información necesaria, llega el momento de enviar nuestra aplicación a Microsoft para su análisis, validación y certificación. Este proceso puede durar un máximo de 72 horas, y su estado se puede seguir a través de la web de desarrolladores.

Enviamos nuestra aplicación por primera vez el martes 8 de agosto de 2017, y el día 10 se nos notificó de un problema con la certificación. Puesto que nuestra aplicación tiene acceso a la cámara del dispositivo, era obligatorio incluir en las descripciones de la Tienda una URL a nuestra política de privacidad. Había que especificara qué se iba a hacer con los datos que se

[12] (PEGI, 2017)

pudieran recopilar, aunque en nuestro caso fueran nulos. Aun así, tuvimos a realizar un pequeño sitio web[13] para la aplicación con toda esta información.

Ilustración 11: Captura de la política de privacidad disponible en la web de la aplicación

La aplicación se volvió a enviar el mismo 10 de agosto, y la aplicación pasó el proceso de certificación el día 12.

10.5. Distribución de la aplicación

Una vez certificada la aplicación, se procede a su distribución por todos los mercados disponibles. En nuestro caso, la aplicación estuvo disponible para descargar el martes 16 de agosto a las 20:27, tanto en equipos de sobremesa como en móviles.

La dirección permanente de la aplicación en la Tienda de Microsoft es: https://www.microsoft.com/store/apps/9MT0ZT0PS0SH

[13] https://colorblindhelper.wordpress.com/

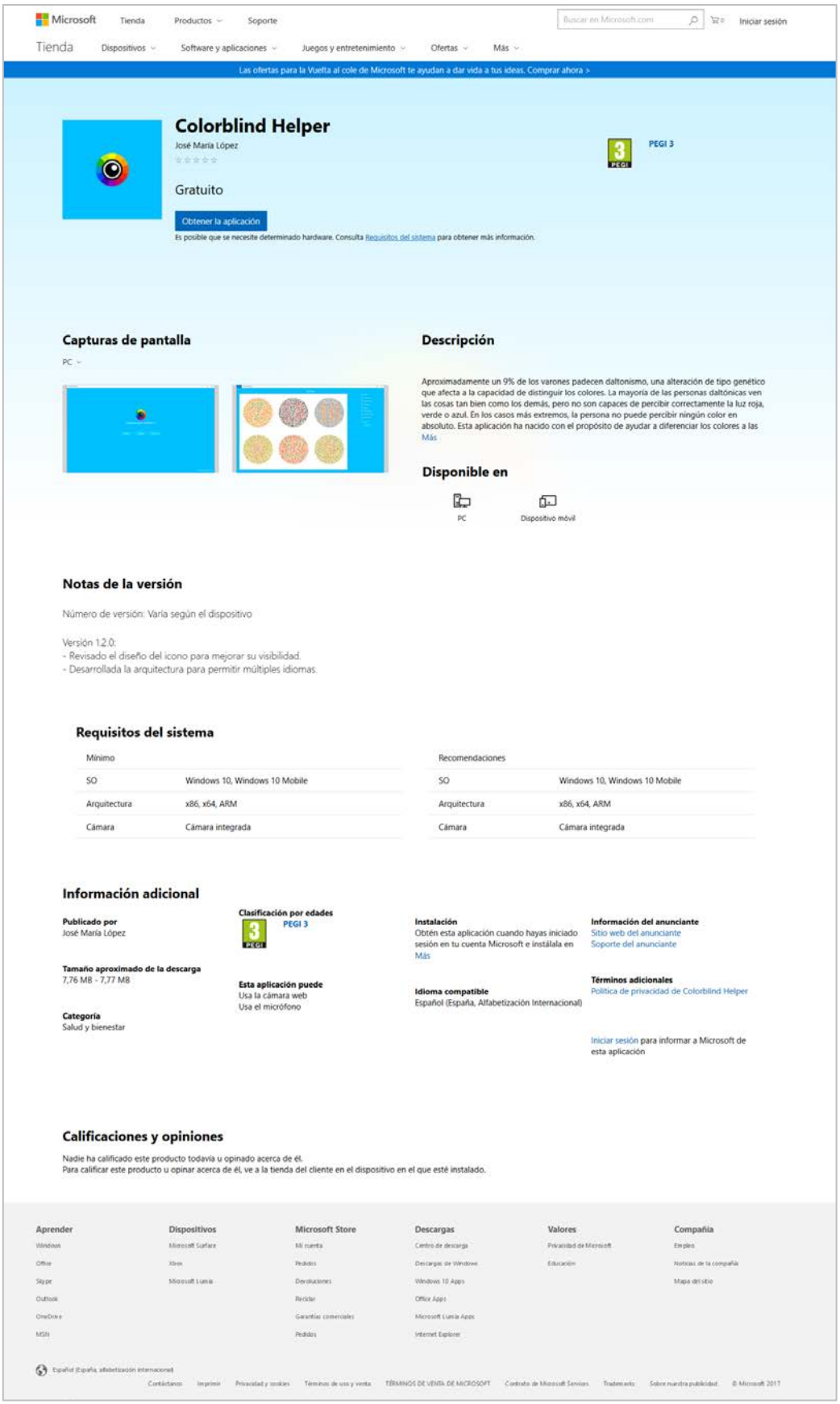


Ilustración 12: Captura de la página de la aplicación en la Tienda de Microsoft

11. Conclusiones

Finalizado el proyecto, se ha realizado con éxito la aplicación propuesta. El resultado es una aplicación disponible en la Tienda de Microsoft que permite a una persona daltónica visualizar imágenes a través de la cámara de su dispositivo, modificando los colores de tal manera que pueda diferenciarlos con mayor facilidad.

Sin embargo, han surgido ideas nuevas a lo largo del proceso que se podrían implementar para mejorar la aplicación:

Idiomas adicionales:

No hay demasiados textos en la aplicación, así que la inclusión de nuevos idiomas sería bastante factible.

Nuevo icono de la aplicación:

Es posible que, en función de la configuración de pantalla (fondo de escritorio y color personalizado), el icono no se vea correctamente en equipos de sobremesa. Se analizará la modificación del mismo para que sus colores contrasten más.

Inclusión del manual del usuario:

Aunque su funcionamiento es bastante sencillo, se podría mostrar una pequeña guía la primera vez que se ejecutara la aplicación. Por supuesto, el usuario también podría leerla en cualquier momento desde el menú principal.

Calibración con fotografías:

En lugar de utilizar las cartas de Ishihara para identificar el tipo de daltonismo del usuario, sería posible incluir la opción de que el usuario hiciera una fotografía (por ejemplo, al objeto cuyos colores no logra diferenciar) y la empleara en la calibración.

Reconocimiento de colores:

Además de la funcionalidad básica de ayudar a diferenciar colores, sería interesante que la aplicación pudiera indicarle al usuario el color que se está mostrando a través de la cámara del dispositivo.

12. Bibliografía

Campbell, T. G. (2016). iPad colour vision apps for dyschromatopsia screening. *Journal of Clinical Neuroscience, 29*, 92-94.

Cartas de Ishihara. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Cartas_de_Ishihara

Dalton, J. (1794). Extraordinary facts relating to the vision of colours. En *Memoirs of the Literary Philosophical Society of Manchester* (págs. 28-45). Londres: Cadell and Davins.

Daltonize. (s.f.). Obtenido de http://www.daltonize.org/

Documentación Win2D. (s.f.). Obtenido de https://microsoft.github.io/Win2D/html/Introduction.htm

Farnsworth, D. (1943). The Farnsworth-Munsell 100-hue and dichotomous tests for color vision. *Journal of the Optical Society of America; Vol. 33, Issue 10*, 568-578.

French, A. R. (2008). The evolution of colour vision testing. *Australian orthoptic journal, 40*, 7-15.

IRS. (9 de Marzo de 2017). Obtenido de Effectively Connected Income (ECI): https://www.irs.gov/individuals/international-taxpayers/effectively-connected-income-eci

Ishihara, S. (1960). *Tests for colour-blindness.* Tokyo: Kanehara Shuppan Company.

Microsoft Developer. (s.f.). Obtenido de https://developer.microsoft.com/es-es/

PEGI. (2017). Obtenido de ¿Cuál es el significado de las marcas?: http://www.pegi.info/es/index/id/96/

Stack Overflow. (s.f.). Obtenido de https://stackoverflow.com/

StatCounter. (Julio de 2017). Obtenido de http://gs.statcounter.com/os-market-share/desktop-mobile-tablet/worldwide/#monthly-201607-201707

Anexo 1: Manual del usuario

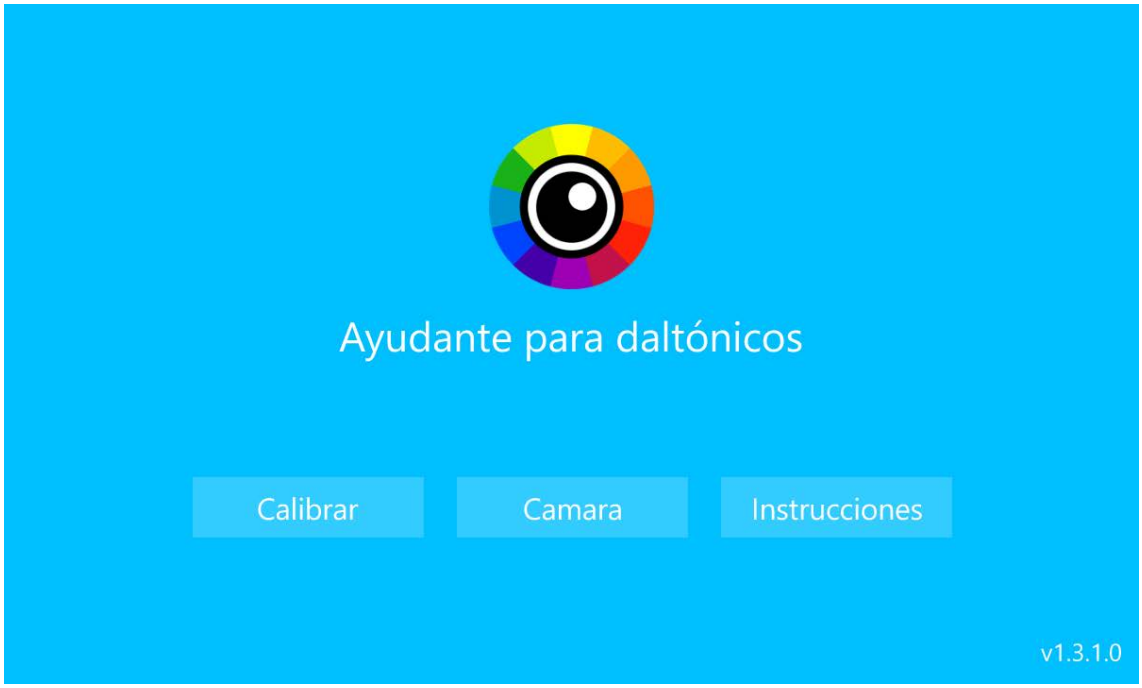
A continuación, se recoge el manual del usuario. Este manual se muestra la primera que se inicia la aplicación, y puede ser consultado en todo momento desde el menú principal.

Introducción

El daltonismo es una alteración de tipo genético que afecta a la capacidad de distinguir los colores. Se produce por uno o más tipos de conos (células sensoriales responsables de la visión del color) están ausentes, no funcionan correctamente, o detectan un color diferente al normal. Con *Colorblind Helper*, las personas daltónicas pueden diferenciar los colores empleando la cámara de sus dispositivos.

¿Cómo funciona?

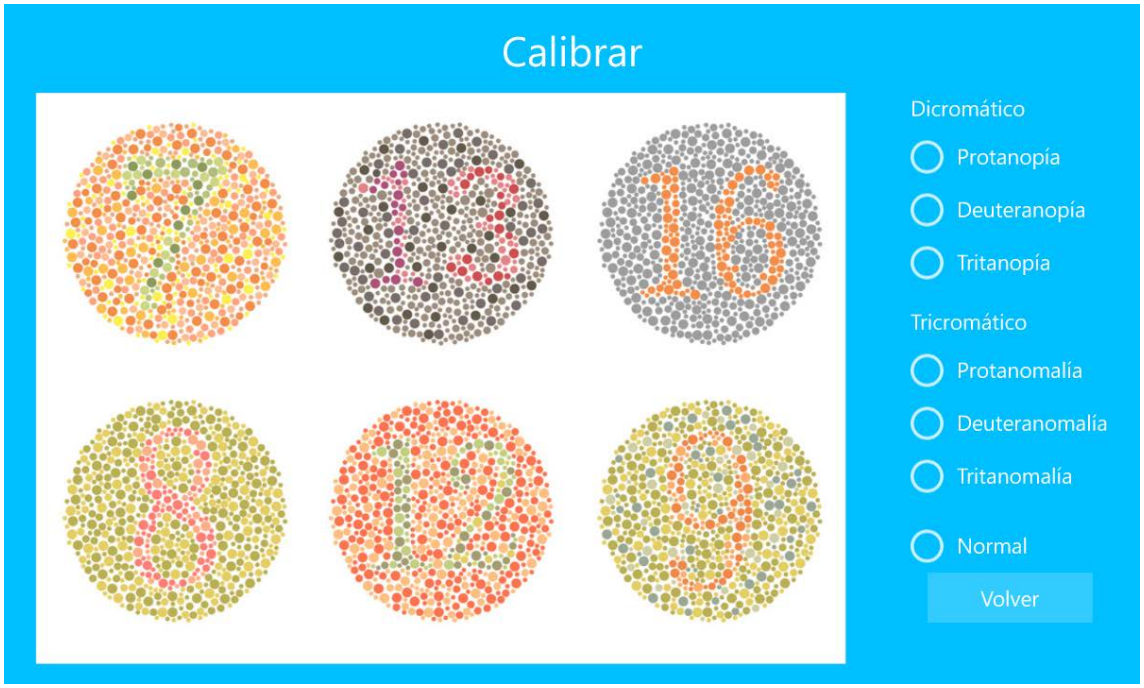
Desde el menú principal podrás acceder a todas las funcionalidades de la aplicación.



Lo primero que tendrías que hacer sería calibrar la aplicación para que se adapte al tipo de daltonismo que tienes. Podrás hacerlo con sólo pulsar el botón 

Calibración

En esta nueva pantalla verás varios círculos de colores con un número en su interior. Estos círculos, llamados cartas de Ishihara, fueron diseñados por este doctor japonés para diagnósticar el daltonismo. Al lado de la imagen se encuentran los tipos de daltonismo más comunes: dicromatismo (deficiencia grave en uno de los conos que detectan el color, creando una ausencia de dicho color) y tricromatismo (deficiencia ligera en uno de los conos, por lo que confunden un color con otro).



Al pulsar cada uno de los tipos de daltonismo disponibles, la imagen cambiará sus colores para facilitar el reconocimiento del número en el interior de los círculos. Puede devolver los colores a su estado original seleccionando la opción “Normal”.

La calibración se guarda automáticamente, así que no será necesario que repitas este paso la próxima vez que utilices la aplicación.

Una vez haya identificado su tipo, la aplicación quedará configurada, y ya podrá volver al menú principal pulsando el botón 

Cámara

Una vez regrese al menú principal, podrá acceder a la vista de la cámara pulsando en el botón Cámara

En la nueva pantalla se mostrará lo que ve la cámara de tu dispositivo, con las modificaciones de color correspondientes al tipo de daltonismo elegido.

Si tu dispositivo lo permite, en la parte superior derecha podrás activar el autoenfoque de la cámara. De nuevo, para volver al menú principal sólo has de pulsar el botón Volver

Instrucciones

Desde el menú principal podemos obtener información adicional de la aplicación si pulsamos el botón Instrucciones

Las instrucciones se muestran al iniciar la aplicación, pero se puede evitar que suceda si marcamos la opción "No volver a mostrar". Para volver al menú principal sólo tendremos que pulsar el botón Cerrar

Anexo 2: Descripción del código

A continuación, se recogen secciones del código que se ha desarrollado y que ilustran diferentes partes de la aplicación.

Página de bienvenida

Se ilustra el aspecto gráfico de la página de bienvenida. El lenguaje utilizado es XAML:

```
<Page
    x:Class="ColorblindHelper.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:ColorblindHelper"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
```

Cabecera del bloque, generada automáticamente por Visual Studio

```
    <Page.Transitions>
        <TransitionCollection>
            <EdgeUIThemeTransition Edge="Left" />
        </TransitionCollection>
    </Page.Transitions>
```

Definición de la transición que se realizará al acceder a esta pantalla

```
   <Grid Background="DeepSkyBlue">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="VisualStateGroup">
                <VisualState x:Name="Horizontal">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="500" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="Titulo.(Grid.Row)"
                                Value="1" />
                        <Setter Target="Titulo.(Grid.Column)"
                                Value="0" />
                        <Setter Target="Botones.(Grid.Row)"
                                Value="2" />
                        <Setter Target="Botones.(Grid.Column)"
                                Value="0" />
                        <Setter Target="Copyright.(Grid.Row)"
                                Value="1" />
                        <Setter Target="Copyright.(Grid.Column)"
                                Value="2" />

                        <Setter Target="Titulo.(Grid.ColumnSpan)"
                                Value="3" />
                        <Setter Target="Botones.(Grid.ColumnSpan)"
                                Value="3" />
                        <Setter Target="Copyright.(Grid.RowSpan)"
                                Value="3" />

                        <Setter
Target="InicioCamara.(RelativePanel.AlignHorizontalCenterWithPanel)"
                                Value="True" />
                        <Setter
Target="InicioCalibrar.(RelativePanel.LeftOf)"
                                Value="InicioCamara" />
                        <Setter
Target="InicioAcercaDe.(RelativePanel.RightOf)"
                                Value="InicioCamara" />

                        <Setter Target="InicioCalibrar.(Margin)"
                                Value="10,20,10,20" />
                        <Setter Target="InicioCamara.(Margin)"
```

Configuración de los distintos elementos de la pantalla cuando la orientación del dispositivo es horizontal.

Se ha omitido la parte del código correspondiente a la orientación vertical por ser muy similar.

```
                        Value="10,20,10,20" />
                    <Setter Target="InicioAcercaDe.(Margin)"
                        Value="10,20,10,20" />

                </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>

    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="2*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <StackPanel Grid.Row="1"
                Name="Titulo">
        <Image Source="Assets/StoreLogo.scale-400.png"
               Height="100"/>
        <TextBlock Text="Ayudante para daltónicos"
                   Style="{StaticResource TituloTextBlockEstilo}"
                   Margin="0,10,0,50"/>
    </StackPanel>

    <RelativePanel Grid.Row="2"
                Name="Botones"
                HorizontalAlignment="Center">
        <Button Name="InicioCalibrar"
                Content="Calibrar"
                Style="{StaticResource InicioBotonEstilo}"
                Click="InicioCalibrar_Click"/>
        <Button Name="InicioCamara"
                Content="Cámara"
                Style="{StaticResource InicioBotonEstilo}"
                Click="InicioCamara_Click"/>
        <Button Name="InicioAcercaDe"
                Content="Acerca de"
                Style="{StaticResource InicioBotonEstilo}"
                Click="InicioAcercaDe_Click"/>
    </RelativePanel>

    <StackPanel Grid.Row="2"
                Name="Copyright"
                VerticalAlignment="Bottom"
                HorizontalAlignment="Right">
        <TextBlock Name="Firma"
                   Text="José María López, 2017"
                   Style="{StaticResource InicioTextBlockEstilo}"/>
    </StackPanel>
  </Grid>
</Page>
```

Configuración de la pantalla, que constará de 3 filas de elementos y 3 columnas.

Definición de los diferentes elementos (imágenes, botones y textos) que componen la pantalla.

Calibración

Mostraremos ahora parte del código que gestiona el funcionamiento de la calibración de la visión daltónica. En esta ocasión el lenguaje utilizado es C#:

```
using Microsoft.Graphics.Canvas;
using Microsoft.Graphics.Canvas.Effects;
using Microsoft.Graphics.Canvas.UI;
using Microsoft.Graphics.Canvas.UI.Xaml;
using System;
using System.Threading.Tasks;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
```

Declaración de las clases que se van a utilizar en esta página.

```
namespace ColorblindHelper
{
    public sealed partial class Calibrar : Page
    {
        CanvasBitmap bitmapResultado;
        ColorMatrixEffect colorEffect;
        public static float C11, C22, C33;
        public static float C12, C13, C21, C23, C31, C32;
        public static int codigoColor;

        public Calibrar()
        {
            this.InitializeComponent();

            // Inicializo los colores
            Calibrar.C11 = 1;
            Calibrar.C12 = 0;
            Calibrar.C13 = 0;
            Calibrar.C21 = 0;
            Calibrar.C22 = 1;
            Calibrar.C23 = 0;
            Calibrar.C31 = 0;
            Calibrar.C32 = 0;
            Calibrar.C33 = 1;

            // Inicializo el código del daltonismo
            codigoColor = 0;
        }
```

Creación de la clase principal dentro de espacio de la aplicación, declaración de variables principales e inicialización.

```
        void imgResultado_CreateResources(CanvasControl sender,
CanvasCreateResourcesEventArgs args)
        {
args.TrackAsyncAction(imgResultado_CreateResourcesAsync(sender).AsAsyncAction());
        }

        async Task imgResultado_CreateResourcesAsync(CanvasControl sender)
        {
            bitmapResultado = await CanvasBitmap.LoadAsync(sender,
"Assets/colorblindness.jpg");
            colorEffect = new ColorMatrixEffect { Source = bitmapResultado };
        }

        private void imgResultado_Draw(CanvasControl sender, CanvasDrawEventArgs
args)
        {
            SetEffectProperties();

            args.DrawingSession.DrawImage(colorEffect);
        }

        void SetEffectProperties()
        {
            var matrix = new Matrix5x4();

            matrix.M11 = (float)(Calibrar.C11);
            matrix.M12 = (float)(Calibrar.C12);
```

Declaración de las funciones que gestionan el dibujado de la imagen de calibración en función del efecto de color aplicado.

Destacar que la tarea de la creación de la imagen a dibujar es asíncrona, pues se produce sólo ante interacciones del usuario.

```
            matrix.M13 = (float)(Calibrar.C13);

            matrix.M21 = (float)(Calibrar.C21);
            matrix.M22 = (float)(Calibrar.C22);
            matrix.M23 = (float)(Calibrar.C23);

            matrix.M31 = (float)(Calibrar.C31);
            matrix.M32 = (float)(Calibrar.C32);
            matrix.M33 = (float)(Calibrar.C33);

            matrix.M44 = 1;

            colorEffect.ColorMatrix = matrix;
        }

        private void buttonProtanopia_Click(object sender, RoutedEventArgs e)
        {
            Calibrar.C11 = 0.56667f;
            Calibrar.C12 = 0.43333f;
            Calibrar.C13 = 0;

            Calibrar.C21 = 0.55833f;
            Calibrar.C22 = 0.44167f;
            Calibrar.C23 = 0;

            Calibrar.C31 = 0;
            Calibrar.C32 = 0.24167f;
            Calibrar.C33 = 0.75833f;

            if (imgResultado != null)
                imgResultado.Invalidate();

            codigoColor = 1;
        }

        void Page_Unloaded(object sender, RoutedEventArgs e)
        {
            this.imgResultado.RemoveFromVisualTree();
            this.imgResultado = null;
        }

        private void buttonVolver_Click(object sender, RoutedEventArgs e)
        {
            var localSettings =
Windows.Storage.ApplicationData.Current.LocalSettings;
            localSettings.Values["tipoDaltonismo"] =
Convert.ToInt16(codigoColor);
            App.tipoDaltonismo = Convert.ToInt16(codigoColor);

            Frame.GoBack();
        }
    }
}
```

Definición de la función que gestiona el botón "Protanopía", y que modifica los valores de la matriz de transformación según corresponde.

El código del resto de botones se omite por ser reiterativo.

Declaración de la función que elimina el dibujado de la imagen cuando se abandona la página, así como la que gestiona el botón para volver a la pantalla principal.

Nótese que es en este punto cuando guardamos en el almacenamiento del dispositivo la calibración elegida.

Cámara

Por último, recogemos parte del código que gestiona la aplicación del efecto a la vista de la cámara. De nuevo, el lenguaje utilizado es C#:

```
using ColorEffects;
using System;
using System.Linq;
using Windows.Devices.Enumeration;
using Windows.Graphics.Display;
using Windows.Media.Capture;
using Windows.Media.Devices;
using Windows.Media.Effects;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
```

Declaración de las clases que se van a utilizar en esta página.

```
namespace ColorblindHelper
{
    public sealed partial class Camara : Page
    {
        private MediaCapture _mediaCapture;
        DeviceInformation _cameraDevice;

        public Camara()
        {
            this.InitializeComponent();
        }
```

Creación de la clase principal dentro de espacio de la aplicación, declaración de variables principales.

```
        protected async override void OnNavigatedTo(NavigationEventArgs e)
        {
            base.OnNavigatedTo(e);

            // Selecciono la primera cámara existente, y capturo el error de
que no haya una
            var allVideoDevices = await
DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);
            DeviceInformation desiredDevice = allVideoDevices.FirstOrDefault(x
=> x.EnclosureLocation != null
                && x.EnclosureLocation.Panel ==
Windows.Devices.Enumeration.Panel.Front);
            _cameraDevice = desiredDevice ?? allVideoDevices.FirstOrDefault();

            if (_cameraDevice == null)
            {
                var dialog = new Windows.UI.Popups.MessageDialog("Necesita una
cámara poder utilizar correctamente esta aplicación.",
                    "No se ha encontrado una cámara.");

                dialog.Commands.Add(new Windows.UI.Popups.UICommand("Aceptar")
{ Id = 0 });

                dialog.DefaultCommandIndex = 0;

                var result = await dialog.ShowAsync();
                return;
            }
```

Lo primero que hacemos al cargar la página es ver si el dispositivo dispone de cámara. Si no es así, capturamos el error y mostramos un mensaje de advertencia.

```
            _mediaCapture = new MediaCapture();
            var settings = new MediaCaptureInitializationSettings()
            {
                StreamingCaptureMode = StreamingCaptureMode.Video,
            };
            await _mediaCapture.InitializeAsync(settings);
            captureElement.Source = _mediaCapture;
            await _mediaCapture.StartPreviewAsync();

            var focusControl =
_mediaCapture.VideoDeviceController.FocusControl;

            if (focusControl.Supported)
```

Una vez seleccionada la cámara, la inicializamos y comprobamos si permite realizar autoenfoque automático. Si no es así, ocultamos esa opción en la vista de la página.

```
            {
                CafFocusRadioButton.Visibility =
focusControl.SupportedFocusModes.Contains(FocusMode.Continuous)
                    ? Windows.UI.Xaml.Visibility.Visible :
Windows.UI.Xaml.Visibility.Collapsed;
            }
            else
            {
                CafFocusRadioButton.Visibility =
Windows.UI.Xaml.Visibility.Collapsed;
            }


            // Fijar la pantalla en modo horizontal
            DisplayInformation.AutoRotationPreferences =
DisplayOrientations.Landscape;


            switch (App.tipoDaltonismo)
            {
                case 1:
                    var effect = await _mediaCapture.AddVideoEffectAsync(
                        new VideoEffectDefinition(typeof(Vision01).FullName),
                        MediaStreamType.VideoPreview);
                    break;

                default:
                    effect = await _mediaCapture.AddVideoEffectAsync(
                        new VideoEffectDefinition(typeof(Vision00).FullName),
                        MediaStreamType.VideoPreview);
                    break;
            }
```

Bloqueamos la orientación del dispositivo y aplicamos a la vista el efecto correspondiente.

Por simplificar, sólo se muestra uno de los casos y la opción por defecto (visión normal).

```
        private async void CafFocusRadioButton_Checked(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
        {
            var focusControl =
_mediaCapture.VideoDeviceController.FocusControl;
            await focusControl.UnlockAsync();
            var settings = new FocusSettings { Mode = FocusMode.Continuous,
AutoFocusRange = AutoFocusRange.FullRange };
            focusControl.Configure(settings);
            await focusControl.FocusAsync();
        }
```

Declaración de la función que gestiona el autoenfoque de la cámara, a la que se accede al pulsar la opción correspondiente.

```
        private void Button_Click(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
        {
            Frame.GoBack();
            DisplayInformation.AutoRotationPreferences =
DisplayOrientations.None;
        }
    }
}
```

Declaración de la función que gestiona el botón para volver a la pantalla principal, en la que desbloqueamos la orientación del dispositivo.