Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

ColorDoku3D: puzle 3D para desarrollar tu percepción del color

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Irene Cebrián Onsurbe

Tutores: David de Andrés Martínez

Juan Serra Lluch

Curso 2019-2020

Resumen

El presente Trabajo Fin de Grado consiste en el diseño y desarrollo de un juego con fines educativos para los alumnos del Máster Habilitante en Arquitectura. La finalidad del juego es mejorar la percepción sobre el color y sus cualidades en los alumnos, usando recursos propios de su ámbito como el espacio tridimensional, la geometría descriptiva y el espacio de color.

El juego consiste en la resolución de puzles de color entramados sobre las caras de figuras geométricas representadas en tres dimensiones. El objetivo que se ha de perseguir para completar los puzles es seguir la relación que se observa entre los colores y colorear las caras de la figura de manera que cada cara esté pintada de un color relacionado con el de sus caras colindantes. El concepto de espacio de color estará ligado a esta forma de resolución del puzle y es otro objetivo de aprendizaje del juego.

El juego está implementado como una aplicación móvil para facilitar el acceso a los alumnos y su práctica en cualquier momento. Se estudian los distintos tipos de aplicación móvil, optando por un desarrollo híbrido para asegurar la accesibilidad al alumno independientemente de cuál sea el sistema operativo de su teléfono móvil.

A nivel técnico, el desarrollo se ha realizado con el *framework* Ionic junto a las tecnologías web. Además, los gráficos 3D se han implementado mediante la librería Three.js de JavaScript y se hace uso del espacio de color HSL.

Palabras clave: puzle, color, juego, aplicación móvil, Ionic, 3D, Three.js, espacio de color, HSL, ColorDoku3D

Abstract

This final project involves the design and development of a game for educational purposes which is designed for use by students of the Enabling Architecture Master. The purpose of the game is to improve their perception of colour as well as its qualities, using resources from their area of expertise, such as three-dimensional space, descriptive geometry and the colour space.

The game consists of solving colour puzzles placed on the faces of geometrical figures, which are represented in three dimensions. In order to complete the puzzles, it's necessary to follow the relationship between the colours and to colour the faces of the figures so that every face is painted with a colour that is related to the colours of the adjoining faces. The concept of colour space is associated with solving the puzzle and, therefore, is another learning goal of the game.

The game is deployed as a mobile application to facilitate access for students and make it available at any time. After studying the different types of mobile applications, a

hybrid development was chosen in order to guarantee accessibility for students no matter which mobile operating system it runs on.

At the technical level, the development has been completed using the Ionic Framework together with other web technologies. Besides, 3D graphics have been implemented using the Three.js Javascript library and using the HSL colour space.

Keywords: puzzle, colour, game, mobile application, Ionic, 3D, Three.js, colour space, HSL, ColorDoku3D

Resum

El present Treball de Fi de Grau consisteix en el disseny i desenvolupament d'un joc amb finalitats educatives per als alumnes del Màster Habilitant en Arquitectura. La finalitat del joc és millorar la percepció sobre el color i les seues qualitats en els alumnes, usant recursos propis del seu àmbit com l'espai tridimensional, la geometria descriptiva i l'espai de color.

El joc consisteix en la resolució de puzles de color entramats sobre les cares de figures geomètriques representades en tres dimensions. L'objectiu que s'ha de perseguir per a completar els puzles és seguir la relació que s'observa entre els colors i acolorir les cares de la figura de manera que cada cara estiga pintada d'un color relacionat amb el de les seues cares confrontants. El concepte d'espai de color estarà lligat a la forma de resolució del puzle i és un altre objectiu d'aprenentatge del joc.

El joc està implementat com una aplicació mòbil per a facilitar l'accés als alumnes i la seua pràctica en qualsevol moment. S'estudien els diferents tipus d'aplicació mòbil, optant per un desenvolupament híbrid per a assegurar l'accessibilitat a l'alumne independentment de quin siga el sistema operatiu del seu telèfon mòbil.

A nivell tècnic, el desenvolupament s'ha realitzat amb el framework Ionic juntament amb tecnologies web. A més, els gràfics 3D s'han implementat mitjançant la llibreria Three.js de Javascript i es fa ús de l'espai de color HSL.

Paraules clau: puzle, color, joc, aplicació mòbil, Ionic, 3D, Three.js, espai de color, HSL, ColorDoku3D

Agradecimientos

A mi familia, porque siempre han creído en mí y me han animado a formarme tanto personalmente como profesionalmente.

A mis amigos, por todo el cariño y apoyo recibido en este proceso final. En concreto, a los formados en la universidad, los cuales significan un gran regalo de la vida unido a estos estudios.

A mis compañeros de piso, los cuales entran en las anteriores categorías, pero merecen mención especial porque han sido mi pilar afectivo durante este trabajo.

A la comunidad de educación pública, por brindarme la oportunidad de formarme y ayudarme en el camino.

A mis tutores, por su ayuda y compresión durante la realización de este trabajo.

Índice general

Índice de tablas

Índice de figuras

1. Introducción

La gamificación es un recurso cada vez más popular, el cual emplea elementos propios de un juego para presentar un enfoque distinto a la hora de solucionar un problema, creando atracción y motivación en las personas implicadas [1]. En el espectro del mundo de los juegos se encuentran los Serious Games, o juegos serios, como juegos con una clara finalidad educativa en lugar del objetivo convencional de diversión. Que se denominen serios no va relacionado con su grado de entretenimiento, si no con su propósito principal de formación. Entre los objetivos que se han perseguido por los Serious Games se encuentran la docencia de asignaturas como matemáticas dentro del ámbito educativo, la formación de personal en empresas para promover habilidades de negociación o liderazgo, e incluso la práctica de coordinación dentro de una sala de emergencias de un hospital para que los enfermeros ganen experiencia [2]. Los Serious Games se unen con la gamificación para incluir dinámicas de recompensa como sistemas de puntos, clasificaciones, niveles y desafíos. Estos recursos fomentan el uso de los Serious Games y los convierten atractivos, aunque su intención principal sea educacional.

En este caso, la intención es promover el aprendizaje sobre el color a través de la gamificación, desarrollando una aplicación móvil que consiste en un juego. Estará dirigido a alumnos del Máster Habilitante en Arquitectura de la Universidad Politécnica de Valencia, que están matriculados en la asignatura 'Graphic and Chromatic Design for Architecture and Urban Space'. Se trata de una asignatura optativa específica sobre color y arquitectura.

Una de las habilidades más importantes que un arquitecto debe adquirir para manejar adecuadamente el color es conocer, comprender y manipular con facilidad las tres variables psicométricas que describen un color: tono, saturación y luminosidad [3]. Aunque el entendimiento teórico de estos tres conceptos es relativamente fácil, su manejo requiere haber dedicado horas de entrenamiento para discriminar colores y saber ordenarlos adecuadamente. El empleo de la gamificación en el aula permite adquirir estas destrezas de una manera lúdica y operativa. Además, el alumno entiende que la ordenación de los colores dibuja un espacio tridimensional susceptible de ser manejado para la composición tridimensional de los espacios arquitectónicos.

A diferencia de otros juegos disponibles como 'Blendoku'[1] o 'I love hue'[2] en los que el usuario debe ordenar colores en una interfaz bidimensional, el empleo de sólidos tridimensionales redunda en el entendimiento de que son tres las variables que describen un color, como se ha señalado: tono, saturación y luminosidad. Por tanto, esta aplicación tiene la intencionalidad de reproducir este tipo de juegos traduciendo su superficie bidimensional a una tridimensional sobre figuras geométricas, uniendo los tres ejes de coordenadas a las tres variables del color haciendo uso del espacio de color HSL.

[1] http://www.blendoku.com/
[2] http://i-love-hue.com/

Por otro lado, el empleo de sólidos regulares de múltiples caras como soporte para la ordenación de los colores, permite que el alumno repase aspectos de geometría descriptiva. No es nuevo el empleo del color como vehículo para la comprensión de la geometría. Así, uno de los textos fundacionales de la geometría descriptiva "Los Elementos" de Euclides, fue publicado en Londres en 1847 por Oliver Byrne, el cual utilizaba un código gráfico basado en colores puros que sustituye al texto, logrando de este modo un producto atractivo, no solo desde el punto de vista matemático, sino desde el punto de vista plástico [5].

1.1. Motivación

La elección de un trabajo sobre la planificación y el desarrollo de un sistema software se fundamenta por la rama de conocimiento de software en la cual me he formado dentro de la Ingeniería Informática. Este campo de la informática engloba todas las etapas de vida de un proyecto de desarrollo software y se encarga de identificar, planificar, desarrollar y resolver cualquier aspecto relacionado con él, tal y como se pretende en este trabajo [6].

Concretamente, este sistema software se trata de una aplicación móvil, un área de interés personal debido a la experiencia formada en el ámbito profesional. El uso de nuevas herramientas y su estudio me concede una oportunidad para ampliar mis conocimientos sobre este tipo de desarrollo. Las aplicaciones móviles son programas altamente accesibles para cualquier usuario en la actualidad, y es una razón muy interesante para tener en cuenta a la hora de plantear su desarrollo en lugar de un programa destinado a ejecutarse en el ordenador.

Además, este proyecto plantea un desarrollo 3D, un campo que no conozco personalmente. Me atrae aprender las propiedades y el funcionamiento que lo definen para adquirir unos conocimientos básicos de cara al futuro profesional, incluso para proyectos propios. Las tecnologías 3D son un recurso muy potente que está presente actualmente en herramientas visuales y desarrollo de videojuegos.

Por último, es gratificante saber que este proyecto tiene una finalidad educativa y que ayudará al aprendizaje sobre la percepción del color a los alumnos de esta materia, al igual que al repaso de la geometría descriptiva.

1.2. Objetivos

La misión de este trabajo es desarrollar una aplicación móvil que funcione como instrumento de aprendizaje para mejorar la compresión del color, entender y manipular sus características principales a través del espacio de color y entrenar la percepción sobre su distinción y su orden.

Para desarrollar esta aplicación, primero, se debe hacer un estudio de las aplicaciones móviles en la actualidad que sirva como marco de referencia sobre qué tipos de

desarrollo existen y cuáles son sus similitudes y diferencias, además de sus límites. Se pretende adquirir un conocimiento general sobre las aplicaciones móviles que sirva para reflexionar y decidir cuál es la opción más conveniente.

Antes de comenzar a desarrollar la aplicación móvil se tiene que haber realizado un diseño, tanto interno como externo. Teniendo en cuenta la finalidad esperada, se ha de plantear la estructura arquitectónica de la aplicación y detallar su comportamiento funcional. Este diseño previo servirá como base y guía cuando comience la implementación del juego.

Antes de comenzar a desarrollar la aplicación móvil se debe diseñar su estructura arquitectónica y plantear su comportamiento específico para definir los requisitos que han de cumplir respecto a la finalidad esperada.

Una vez se hayan elegido las herramientas que den soporte al desarrollo de la aplicación, será necesario aprender su funcionamiento básico para empezar a implementar las funcionalidades requeridas. Junto a ello, se estudiará el espacio tridimensional y sus principales propiedades porque son conceptos imprescindibles para el desarrollo de la aplicación.

El resultado de todo el desarrollo debe ser una aplicación móvil ejecutable en cualquier sistema operativo, para que cualquier alumno pueda acceder a ella sin depender del móvil que posea.

1.3. Metodología

Para la realización de este trabajo se pretende utilizar una metodología ágil de desarrollo software, basada en un desarrollo iterativo e incremental que permite ajustar los requisitos a lo largo de todo el proceso y abordar los cambios que se pueden producir durante el desarrollo. Al adentrarse en un campo desconocido personalmente como es la programación de elementos 3D, al igual que los recursos que proporcionan sus herramientas de desarrollo, quizás surjan conflictos con las funcionalidades deseadas y sea necesario redefinir los objetivos durante la implementación. Una metodología ágil proporciona este tipo de flexibilidad y adecuación a los problemas que puedan aparecer.

Scrum, la metodología ágil elegida, consiste en entregar y revisar el trabajo hecho cada cierto tiempo para examinar si se han conseguido los objetivos planificados o, en caso contrario, redefinirlos y así adaptarse durante el proceso a las complicaciones que puedan ocurrir [7]. Nos basaremos en su funcionamiento para definir varios objetivos, proceder al estudio de su implementación y desarrollo, y seguidamente, a su revisión y evaluación, y así sacar conclusiones para definir los siguientes objetivos a implementar.

1.4. Estructura de la memoria

El presente documento está compuesto por ocho apartados en total, tras un breve resumen que explica de qué trata el trabajo.

El primer apartado es la introducción y sirve para presentar de dónde nace la idea de este trabajo, aclarar los objetivos que se persiguen y explicar la metodología que se seguirá para desarrollarlo.

En el segundo apartado, se realiza un estudio sobre los distintos desarrollos de aplicaciones móviles que se pueden seguir con una comparativa final para decantarse por uno de ellos en este trabajo. Después, se examinan qué herramientas son usadas actualmente para el desarrollo de aplicaciones móviles en el ámbito híbrido y, después de un análisis, se elige una de ellas para desarrollar ColorDoku3D.

En el tercer apartado se analiza la idea principal de la aplicación y se desarrollan sus requisitos a cumplir, especificando qué funcionalidades se han de implementar y cuáles son las interacciones que ha de poder realizar el usuario con la aplicación móvil para jugar.

En el cuarto apartado, se presenta el diseño pensado para la aplicación y se explica cómo se estructura internamente. También se muestran los bocetos iniciales para su interfaz gráfica, resultado del análisis de requisitos y la especificación de casos de uso.

El quinto apartado pertenece a la implementación de la aplicación móvil. Primero, se comentan las tecnologías y las herramientas empleadas en todo el trabajo. Después, se explica en orden de desarrollo cómo se ha abarcado la implementación del juego. Además, se describen en detalle las funcionalidades y los elementos más importantes, detallando cómo se han implementado en la aplicación. Por último, se describe el despliegue de la aplicación en las distintas plataformas.

El sexto apartado presenta el resultado final de la aplicación, explicando las acciones que realiza un alumno ficticio usándola. El proceso descrito se puede visualizar a la par con las capturas realizadas de la aplicación final replicando las mismas acciones.

El séptimo apartado presenta las conclusiones del trabajo, exponiendo los desafíos que se han llevado a cabo y las complicaciones que han surgido en el desarrollo, junto con los objetivos conseguidos con éxito. También se detalla la relación del trabajo realizado con los estudios cursados. Después, se realiza un análisis de las proyecciones futuras de este trabajo.

Por último, el apartado de bibliografía lista las referencias que se han incluido en todo el trabajo para fundamentar los temas tratados y ofrecer su consulta al lector.

2. Estado del arte

Es primordial conocer qué opciones de desarrollo hay disponibles para elaborar el juego propuesto y considerar qué nos ofrecen cada una de ellas. Con esta intención se analizan los distintos tipos de desarrollo de aplicaciones móviles, posteriormente se profundizará en la elección que resulte más conveniente.

2.1. Desarrollo de aplicaciones móviles

Durante los últimos años los móviles, concretamente los smartphones, se han popularizado como dispositivo tecnológico de fácil disponibilidad. Su mayor poder de cómputo brinda multitud de aplicaciones de diversa índole, y el desarrollo de éstas cada vez dispone de un mayor abanico de herramientas para llevarse a cabo.

El desarrollo específico de aplicaciones móviles se diferencia notablemente del desarrollo de software tradicional. Previamente se precisa deliberar cuáles son las características de interés y así valorar qué tipo de aplicación es la adecuada para nuestro propósito, de esta manera se delimitan las herramientas disponibles para su desarrollo [8]. A continuación, se describen los distintos tipos de aplicación móvil para tener en cuenta, según su desarrollo.

2.1.1. Aplicaciones nativas

Son específicas para cada sistema operativo y es necesario programarlas en el lenguaje nativo de la plataforma elegida. Por lo tanto, no son portables, se precisa un desarrollo distinto para cada plataforma. Hoy en día, esta distinción se reparte mayoritariamente entre iOS [3] y Android [4]. Los lenguajes utilizados son Swift [5] u Objective-C [6] para el sistema operativo de iOS y Java [7] o Kotlin [8] para el sistema operativo de Android.

La gran ventaja es el acceso a las particularidades de cada plataforma. Trabajar con el sistema operativo específico permite utilizar el hardware y disponer de herramientas para utilizar sensores como el GPS, el giroscopio y la cámara del teléfono móvil. También afecta a su rendimiento, en general es más optimizado.

[3] https://www.apple.com/es/ios/ios-14/
[4] https://www.android.com/intl/es_es/
[5] https://www.apple.com/es/swift/
[6] https://es.wikipedia.org/wiki/Objective-C
[7] https://www.java.com/es/
[8] https://kotlinlang.org/

Estas aplicaciones tienden a tener una mejor experiencia de usuario, ya que sus interfaces siguen guías de estilo del propio sistema operativo y homogeneizan el aspecto con las demás aplicaciones. Al usuario le resulta más intuitivo y sencillo el uso de cualquiera de ellas.

El gran inconveniente es el coste de desarrollo porque es necesario crear una aplicación por cada plataforma para que pueda utilizarla cualquier usuario, independientemente de cuál es su teléfono móvil. Para ello se necesita personal especializado en cada lenguaje nativo, y el proceso de desarrollo de cada aplicación va por separado, siendo realmente una sola aplicación en cuanto a funcionalidad. Este coste también es mayor en el mantenimiento de la aplicación, siendo necesario implementar los cambios en cada plataforma independientemente.

2.1.2. Aplicaciones web

Son aplicaciones que pueden ejecutarse en cualquier navegador móvil, independientemente de su sistema operativo. Se llevan a cabo con las principales tecnologías web: HTML[9], CSS[10] y JavaScript[11]. En 2018, se contabilizaron 35,8 millones de usuarios conectados a Internet desde su dispositivo móvil en España [9], en consecuencia, el requisito de acceso a la red de las aplicaciones web hace de éstas una opción multiplataforma real. Además, no ocupan memoria de almacenamiento porque no es necesario instalarlas en el dispositivo, simplemente se ejecuta en el navegador y su consumo de recursos es mínimo. En consecuencia, el usuario siempre accede a la última versión, sin necesidad de descargar y actualizar la aplicación. Por otro lado, no están capacitadas para acceder a las características propias del teléfono.

Dentro de esta categoría, se encuentran las *Progressive Web Apps* (PWA) que cuentan con una serie de mejoras para que la apariencia y la funcionalidad de una aplicación web se asemeje a una aplicación nativa del teléfono. Son descargables para tenerlas almacenadas como otra aplicación móvil y pueden funcionar sin acceso a internet. Además, cuentan cada vez con más recursos que les permiten funcionalidades que requieren acceso al dispositivo, propias de las nativas, como notificaciones *push*[12] o el uso de *service workers*[13] para la ejecución en segundo plano [10].

2.1.3. Aplicaciones híbridas

Combinan las ventajas de las aplicaciones nativas y web. Su desarrollo es independiente de la plataforma en la que se ejecutará la aplicación, es decir, la misma aplicación se podrá ejecutar en cualquier sistema operativo. Esto es posible porque la aplicación es embebida en un contenedor nativo que ejecuta dentro de él un navegador. El resultado final es una aplicación instalable, en lugar de una aplicación

[9] https://developer.mozilla.org/es/docs/Web/HTML
[10] https://developer.mozilla.org/es/docs/Web/CSS
[11] https://developer.mozilla.org/es/docs/Web/JavaScript
[12] Notificaciones del móvil con mensajes de aplicaciones recibidos de servidores remotos
[13] Servicios que ejecutan código en segundo plano en el navegador

ejecutada en el navegador, como son las aplicaciones web. Esta capa intermedia la ofrecen herramientas que no son propias del sistema operativo, si no de terceros.

Se desarrolla con los lenguajes de programación HTML, CSS y JavaScript, propios del desarrollo web, junto con librerías y *frameworks* que permite el acceso a las principales características propias de la plataforma en la que se está ejecutando, por ejemplo, la cámara del dispositivo. Aun así, dependen de las tecnologías disponibles para acceder a las características nativas, y puede no ser posible implementar funcionalidades más específicas del hardware.

En cuanto al diseño de la interfaz, ya no siguen las guías recomendadas por cada plataforma, en cambio, tienen las que proporciona cada *framework*. Esto puede afectar a la experiencia del usuario. Además, el rendimiento general de la aplicación se puede ver perjudicado por la ejecución del contenedor web como capa intermedia, en lugar de la nativa directamente.

La mayor ventaja es el desarrollo de una sola aplicación para su instalación y ejecución en cualquier sistema operativo de teléfono móvil, significando un menor coste de desarrollo y mantenimiento.

2.1.4. Aplicaciones generadas

El término ‘generadas’ se debe al proceso de tener un solo código para desarrollar la aplicación que posteriormente se compila al lenguaje nativo de la plataforma destino, es decir, se genera una aplicación nativa para cada plataforma desde el mismo código. Es una ventaja ya que las aplicaciones nativas son más recomendables para el buen rendimiento de la aplicación.

Las tecnologías que permiten este desarrollo en su mayoría son de pago. Cada una de ellas utiliza un lenguaje de programación específico. Las más conocidas son Xamarin[14] o Genexus[15].

En su contra, se debe conocer el lenguaje y la API particular de la herramienta que se use para desarrollar la aplicación. Normalmente, aunque accedan a las características propias de lo nativo, no es tan fácil con los gráficos y las interfaces [11].

2.1.5. Análisis y conclusiones

La Tabla 2.1 compara las principales características de los tipos de aplicaciones móviles estudiados para facilitar el análisis y la elección de uno de ellos.

Entre las características elegidas, se encuentran varias relacionadas con las particularidades deseadas para la aplicación. Nos puede interesar la ejecución nativa por la importancia del rendimiento, así como la integración del hardware si requerimos del uso de los sensores incorporados en el móvil. El acceso a Internet puede ser

[14] https://dotnet.microsoft.com/apps/xamarin
[15] https://www.genexus.com/es/

opcional e interesar o no según las funcionalidades, hay aplicaciones que no necesitan Internet y otras que precisan de ello para descargar todos los datos e interactuar con ellos.

Otras características se enfocan más en el desarrollo. Es importante saber qué lenguajes de programación están implicados para considerar lo que pueden aportar y si facilitan la implementación de la aplicación. Al mismo tiempo, el coste de desarrollo es crucial para conocer la cantidad de recursos que se está dispuesto a consumir.

Tanto la instalación de la aplicación en el móvil como la distribución en las tiendas de aplicaciones de cada plataforma son aspectos decisivos para elegir un tipo de aplicación. En este caso, como se puede comprobar en la Tabla 2.1, ayuda para descartar la opción de aplicaciones web, visto que las demás permiten estás opciones. Esta característica es interesante si se quiere llegar a los usuarios fácilmente y mostrarse como otra opción más dentro del mercado de aplicaciones.

Tabla 2.1 *- Comparativa entre los distintos tipos de aplicaciones*

	NATIVA	**WEB**	**HÍBRIDA**	**GENERADA**
Ejecución nativa	Sí	No	Limitada	Sí
Distribución en tiendas de aplicaciones	Sí	No	Sí	Sí
Lenguaje de programación	Swift, Java	HTML, CSS, JavaScript	HTML, CSS, JavaScript	Específico de la plataforma
Necesidad de acceso a la red	No	Sí	No	No
Instalación en el móvil	Sí	No	Sí	Sí
Coste de desarrollo	Alto	Bajo	Bajo	Medio
Integración hardware	Completa	Muy limitada	Limitada	Completa

En este caso, las aplicaciones nativas quedan descartadas debido a la necesidad de crear diferentes aplicaciones con diferentes lenguajes para ser ejecutada tanto en Android como en iOS, implicando un alto coste de desarrollo y posterior mantenimiento de la aplicación. Igualmente, el privilegio que tienen las aplicaciones nativas sobre el hardware del dispositivo no es crucial, ya que no es necesario el uso de sensores o la cámara del teléfono. En definitiva, no es necesaria una alta integración del hardware.

El objetivo principal que se quiere satisfacer con esta decisión es que los alumnos puedan utilizar la aplicación independientemente del sistema operativo de su teléfono móvil. O en su defecto, se pueda ejecutar en el ordenador haciendo uso del navegador, debido a la importancia de los gráficos y su relevante perceptibilidad. Teniendo lo último en cuenta, rechazamos la opción de las aplicaciones generadas, sumándose a la razón de que se necesitan herramientas de terceros con tecnologías muy específicas.

Finalmente, nos decantamos por las aplicaciones híbridas debido al uso de los lenguajes de desarrollo web porque su uso es generalizado y más conocido, y los *frameworks* capaces de ejecutar la aplicación teniendo en cuenta las características de la plataforma destino, un proceso más fiable que las aplicaciones web o PWA. Esta opción también nos proporciona la incorporación de la aplicación en el mercado de aplicaciones de las distintas plataformas móviles como son Play Store[16] para Android y App Store[17] para iOS y así facilita el acceso de los alumnos para su descarga.

2.2. Herramientas de desarrollo híbrido

Hasta ahora las empresas tendían a apostar por el desarrollo nativo de aplicaciones móviles por la fiabilidad que proporcionaban en cuanto a la adecuación con el dispositivo destino. Las opciones de tecnologías híbridas eran escasas y nuevas, por lo que causaban rechazo a la hora de apostar por ellas, pese a su principal ventaja de escribir un solo código independiente de las plataformas destino. La duda recaía principalmente en su soporte en el futuro, según la popularidad y aceptación que ganasen, al igual que en sus capacidades para implementar cualquier funcionalidad, que hasta el momento era posible en el desarrollo nativo.

En la actualidad, el desarrollo de aplicaciones híbridas está en auge. Su mayor conocimiento y uso por parte de empresas y desarrolladores permite que este tipo de tecnologías estén evolucionando rápidamente mediante la solución de errores y la incorporación de nuevas características. Simultáneamente, los programadores crean una mayor comunidad a través de los foros de resolución de dudas y errores, facilitando la iniciación y el aprendizaje.

Entre las herramientas disponibles, presentamos las tres opciones más conocidas y valoradas en la actualidad por los desarrolladores de aplicaciones móviles.

2.2.1. React Native[18]

Fue lanzado por Facebook[19] en 2015 y desde entonces ha ganado mucha popularidad en el desarrollo de aplicaciones móviles.

[16] https://play.google.com/store
[17] https://www.apple.com/es/ios/app-store/
[18] https://reactnative.dev/
[19] https://es-es.facebook.com/

Se emplea JavaScript como lenguaje de programación junto con la librería React.js[20], utilizada para la construcción de interfaces basada en componentes interactivos. Todo ello se une con controladores UI[21] nativos para generar aplicaciones que se ejecuten directamente en la plataforma nativa y no sobre un navegador. No siempre es posible conseguir esta ejecución nativa, pero se puede añadir código específico de la plataforma si es necesario, cubriendo todas las necesidades que requiera la aplicación [12]. Destaca su característica de *hot reloading* que facilita la ejecución y visualización casi inmediata de cambios del código en el dispositivo o emulador.

Algunos de las aplicaciones más conocidas creadas con este *framework* son Facebook, Instagram[22] o Airbnb[23].

2.2.2. Ionic[24]

Framework de código abierto basado en la programación web con tecnologías estándar JavaScript, HTML y CSS, lanzado en 2013. Posteriormente, se sustituyó el lenguaje de programación por TypeScript[25], más recomendado en proyectos grandes para suplir las debilidades de JavaScript. Dispone de AngularJS[26] para la creación de interfaces dinámicas que facilitan la sincronización automática entre los datos y la vista. También proporciona una amplia gama de componentes propios para la interacción con la interfaz.

Su singularidad reside en el uso del plugin Cordova[27], que crea el contenedor nativo para la ejecución del código web en las plataformas destino. Este *framework* es de código abierto y permite acceder a las características propias de la plataforma nativa. Para ello también tiene la capacidad de usar *plugins*, tanto propios como de terceras partes, que se pueden añadir para conseguir funcionalidades nativas.

Además, aporta la herramienta Ionic CLI[28] (*Command Line Interface*) para el uso de comandos vía terminal. Los comandos pueden ser de Ionic, Cordova, ejecución y *testing*, entre otros. Simplifica notablemente la creación de elementos de Ionic, como proyectos o páginas preestablecidas, las cuales se crean con todos sus elementos preparados para implementarlas y reducen el tiempo de preparación [13].

Algunas de las aplicaciones con mayor número de descargas hechas con Ionic son McDonald's[29] (Turquía), JustWatch[30] y Sanvello[31] [14].

---

[20] https://es.reactjs.org/
[21] *User Interface* o interfaz de usuario
[22] https://www.instagram.com/?hl=es
[23] https://www.airbnb.es/
[24] https://ionicframework.com/
[25] https://www.typescriptlang.org/
[26] https://angularjs.org/
[27] https://cordova.apache.org/
[28] https://ionicframework.com/docs/cli
[29] https://www.mcdonalds.com.tr/
[30] https://www.justwatch.com/es
[31] https://www.sanvello.com/

2.2.3. Flutter[32]

Tecnología desarrollada por Google[33] y lanzada en 2016 para el desarrollo de aplicaciones nativas en Android y iOS en base a un solo código.

El lenguaje de programación utilizado es Dart[34], también creado por Google, desvinculándose así de la tecnología web estándar. La sintaxis de Dart es parecida a la de JavaScript, pero su principal diferencia es que es un lenguaje tipado.

La creación de interfaces se hace a través de *widgets* propios creados con Dart y que van desde lo más básicos a otros más complejos resultado de combinar los básicos. Los *widgets* son altamente personalizables y uno de los elementos más importantes de Flutter. Además, proporciona paquetes especializados para ajustarnos más al diseño de la plataforma destino, Material Design[35] para Android y Cupertino[36] para iOS.

Dispone de la opción de *hot reloading* para aligerar la ejecución de la aplicación mientras se desarrolla. Se destaca su rendimiento debido a que se compila utilizando bibliotecas C/C++, más cercanas al lenguaje máquina y que se traduce en un mejor rendimiento en el ámbito nativo [15].

Algunas aplicaciones desarrolladas con Flutter son Alibaba[37] y Google Ads[38] [16].

2.2.4. Análisis y conclusiones

La Tabla 2.2 nos muestra las características más importantes de las herramientas propuestas para el desarrollo híbrido.

Entre ellas se encuentra el lenguaje y la curva de aprendizaje, lo cual es importante porque se puede tener experiencia con algunos de los lenguajes y con otros no, aplicándolo igual para las herramientas. Esto afecta al tiempo de desarrollo y a la facilidad de implementación, al igual que el soporte de cada herramienta, que también puede marcar una gran diferencia en la resolución de errores.

Particularmente, se indican las opciones de apariencia de cada herramienta porque puede ser algo decisivo según los requisitos. En ocasiones, el diseño visual de una aplicación se quiere seguir según las normas de diseño de la plataforma destino para que la aplicación no se diferencie de las demás en el aspecto de usabilidad. Otras veces, esto no se requiere o se prefiere elementos más personalizables como pueden ser los componentes web.

[32] https://flutter.dev/
[33] https://www.google.es/
[34] https://dart.dev/
[35] https://flutter.dev/docs/development/ui/widgets/material
[36] https://flutter.dev/docs/development/ui/widgets/cupertino
[37] https://www.alibaba.com/
[38] https://ads.google.com/

Tabla 2.2 *-Comparativa entre las distintas herramientas de desarrollo híbrido*

	REACT NATIVE	**IONIC**	**FLUTTER**
Lenguaje	JavaScript + (Java, Objective C)	TypeScript, HTML, CSS	Dart
Curva aprendizaje	Media	Mínima	Media
Interfaz / Apariencia	Componentes nativos	Componentes web (HTML, CSS)	Componentes propios
Soporte	Alto	Alto	Bajo

Entre los requisitos de esta aplicación se encuentra la necesidad de recurrir a una o varias soluciones externas 3D para los gráficos. Para esta condición es fácil encontrar librerías JavaScript acordes para integrar en el proyecto, siendo posible en React Native e Ionic.

El acceso a las propiedades del teléfono no es de gran interés en este caso, porque no se necesitan elementos hardware específicos para ejecutar el juego. Por el contrario, precisamos de una alta flexibilidad a la hora de implementar la interfaz, debido a la importancia de los controles y los componentes interactivos para jugar. Cualquiera de las soluciones propuestas nos ofrece recursos para la interfaz y su control como botones, listas o pestañas, pero para crear interfaces más personalizables, es conveniente usar HTML con CSS puesto que sus elementos poseen muchos atributos modificables.

En conclusión, Ionic es la solución que mejor se adapta a nuestras necesidades. Además, tanto los lenguajes web como Ionic, brindan un alto soporte por parte de los usuarios para consultar dudas o errores, algo muy importante en este trabajo dado que no se tiene experiencia en ninguna de las tres tecnologías.

3. Especificación

Ahora que ya se ha estudiado qué tipo de aplicación se va a desarrollar y cuál es la herramienta que ayudará a la implementación, se ha de delimitar cuales van a ser las funcionalidades de ColorDoku3D. Para ello, se va a realizar un análisis de requisitos describiendo el comportamiento del juego y proporcionando una idea más concisa sobre las funcionalidades que se deben de desarrollar posteriormente. Después, se compondrá una serie de casos de uso derivados de los requisitos planteados, y así se visualizará claramente cada una de las interacciones del usuario con la aplicación.

3.1. Análisis de requisitos

La Ingeniería de requisitos es una rama de la Ingeniería de software que ayuda en el proceso de desarrollo y mantenimiento de un sistema software. Se concibe como una especialidad en sí misma por la importancia que tiene la identificación, definición y actualización de los requisitos en el desarrollo. Este estudio puede tener un gran impacto en la calidad del producto final y el coste variable de desarrollo debido a la satisfacción del cliente o usuario final, que además va ligado inevitablemente al tiempo de desarrollo.

A continuación, se van a enumerar los requisitos precisados para esta aplicación de manera que se cumpla conjuntamente la idea general del juego que se propone y su intención didáctica. Estos requisitos están basados en la dinámica de juego Bledoku para la resolución de puzles, junto con la intención de incorporar el espacio de color como elemento fundamental. Se han recogido gracias al diálogo con los tutores de este trabajo dónde han expresado sus propósitos y los objetivos principales que se han de cumplir.

1. El usuario elegirá una figura entre una lista formada por las figuras geométricas propuestas para realizar el puzle sobre ella.

2. El usuario configurará el puzle a realizar con la geometría seleccionada modificando su relación respecto al espacio de color.

3. La figura se visualizará dentro de un espacio tridimensional representando un espacio de color. Junto a este espacio, se proporcionarán varios controles que modificarán la relación entre la figura y el espacio de color.

4. La figura podrá moverse respecto a los tres ejes de coordenadas sin salir del espacio de color representado.

5. Se podrá aumentar o reducir el tamaño de la figura dentro del espacio de color, sin superar sus límites.

6. Las caras de la figura asociarán como color a resolver el color representado por el punto que se corresponde con su centro en el espacio de color.

7. La figura donde se resuelve el puzle siempre tendrá un mínimo de caras pintadas. Es necesario como base para su resolución.

8. Se podrá elegir el nivel de dificultad del puzle escogiendo el número de caras a colorear por defecto al inicio del juego, para servir como guía en la resolución de éste.

9. Las caras coloreadas por defecto al inicio del juego no se podrán borrar, de manera que se eviten confusiones y estén como punto de referencia durante todo el juego.

10. El puzle configurado se mostrará antes de su pantalla de resolución, representado por la lista de colores que se han de pintar posteriormente en las caras de la figura. Así el usuario tendrá una representación previa de cómo van a ser los elementos de ese puzle en concreto.

11. El usuario tendrá que aceptar la paleta de colores mostrada para proceder al juego, o volver a configurar el puzle de nuevo si no le convence.

12. La figura geométrica donde haya que resolver el puzle se podrá girar para visualizar cualquiera de sus caras.

13. Se podrá modificar el color de cada una de las caras de la figura, individualmente.

14. Cada cara tendrá un color asociado como el color correcto, de manera que el puzle estará completado cuando todas las caras estén coloreadas con el color correcto asociado. Si el puzle se completa correctamente se notificará al usuario.

15. Se podrá elegir el color a pintar en una paleta de colores mostrada junto a la figura a resolver.

16. Cada color solo puede pintarse sobre una cara de la figura simultáneamente.

17. Se podrá borrar un color pintado en la figura gracias a la herramienta de borrado que se mostrará junto con la lista de colores.

18. La figura del puzle a resolver se podrá ampliar para concentrarse mejor en cada una de sus caras.

19. La figura del puzle a resolver se podrá reducir para facilitar su visión completa y facilitar la percepción de los colores pintados.

3.2. Casos de uso

Los requisitos definidos nos proporcionan la información para reconocer los casos de uso implicados y proceder a su descripción. Con el siguiente diagrama de casos de uso, mostrado en la Figura 3.1, se refleja la representación en conjunto del comportamiento de la aplicación y sus interacciones.

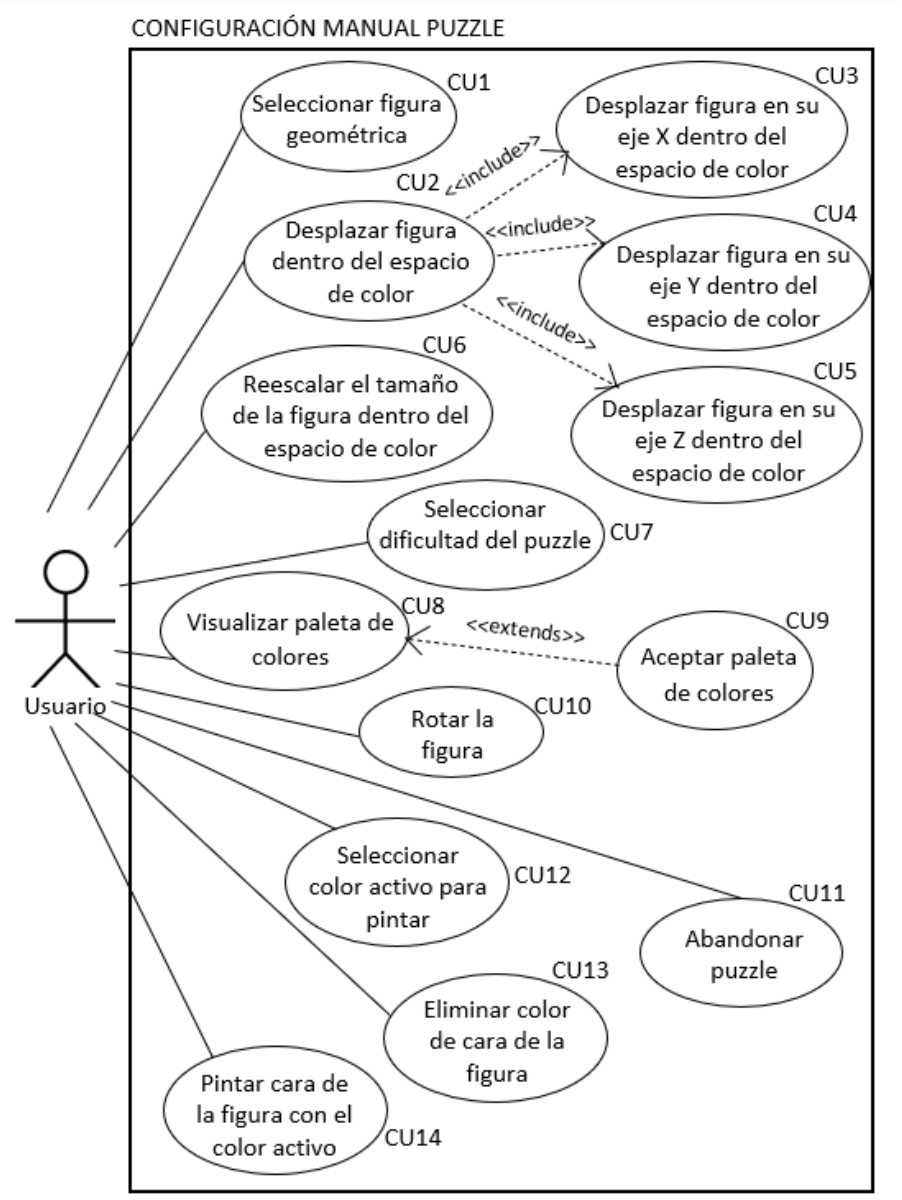


Figura 3.1 *- Diagrama de casos de uso*

La funcionalidad de la aplicación se captura en detalle en los siguientes casos de uso extraídos del diagrama de la Figura 3.1:

Tabla 3.1 *- CU1-Seleccionar figura geométrica*

Actor	Usuario
Caso de Uso	Seleccionar figura geométrica
Precondiciones	Ninguna
Postcondiciones	Figura geométrica seleccionada para jugar
Descripción	El usuario elige entre un listado de figuras geométricas para realizar el puzle sobre ella.

Tabla 3.2 *- CU2-Desplazar figura dentro del espacio de color*

Actor	Usuario
Caso de Uso	Desplazar figura dentro del espacio de color
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Ninguna
Descripción	El usuario mueve la figura geométrica elegida dentro del espacio de color representado para que cada una de sus caras se asocie con un color concreto.

Tabla 3.3 *- CU3-Desplazar figura en su eje X dentro del espacio de color*

Actor	Usuario
Caso de Uso	Desplazar figura en su eje X dentro del espacio de color
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Ninguna
Descripción	El usuario modifica la posición del eje X de la figura dentro del espacio de color.

Tabla 3.4 *- CU4-Desplazar figura en su eje Y dentro del espacio de color*

Actor	Usuario
Caso de Uso	Desplazar figura en su eje Y dentro del espacio de color
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Ninguna
Descripción	El usuario modifica la posición del eje Y de la figura dentro del espacio de color.

Tabla 3.5 *- CU5-Desplazar figura en su eje Z dentro del espacio de color*

Actor	Usuario
Caso de Uso	Desplazar figura en su eje Z dentro del espacio de color
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Ninguna
Descripción	El usuario modifica la posición del eje Z de la figura dentro del espacio de color.

Tabla 3.6 *- CU6-Escalar el tamaño de la figura dentro del espacio de color*

Actor	Usuario
Caso de Uso	Escalar el tamaño de la figura dentro del espacio de color
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Ninguna
Descripción	El usuario escala el tamaño de la figura dentro del espacio de color. Puede aumentarlo o reducirlo sin sobresalir los límites que conforman el espacio de color.

Tabla 3.7 *- CU7-Seleccionar dificultad del puzle*

Actor	Usuario
Caso de Uso	Seleccionar dificultad del puzle
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Ninguna
Descripción	El usuario elije el número de caras que estarán coloreadas por defecto al inicio del puzle, para servir como guía en su resolución. Así modifica el nivel de dificultad del puzle que quiere realizar.

Tabla 3.8 *- CU8-Visualizar paleta de colores*

Actor	Usuario
Caso de Uso	Visualizar paleta de colores
Precondiciones	Figura geométrica seleccionada para jugar
Postcondiciones	Figura geométrica configurada dentro del espacio de color. Calculados los colores del puzle.
Descripción	Se muestra la paleta de colores con los valores configurados por la figura dentro del espacio de color, según su tamaño y ubicación dentro de éste. De esta manera el usuario conoce con qué colores puede resolver el puzle configurado.

Tabla 3.9 *- CU9-Aceptar paleta de colores*

Actor	Usuario
Caso de Uso	Aceptar paleta de colores
Precondiciones	Figura geométrica configurada dentro del espacio de color. Visualizada la paleta de colores para resolver el puzle.
Postcondiciones	Puzle configurado y aceptado por el usuario.
Descripción	El usuario acepta los colores que ha visualizado en el CU8 para que sea la paleta con la que jugar sobre la figura elegida. Se navega a la pantalla de juego o resolución del puzle.

Tabla 3.10 *- CU10-Rotar la figura*

Actor	Usuario
Caso de Uso	Rotar la figura
Precondiciones	Puzle configurado y aceptado por el usuario.
Postcondiciones	Ninguna
Descripción	El usuario rota la figura a resolver de manera que sean visibles todas sus caras y sea accesible interactuar con ellas.

Tabla 3.11 *- CU11-Abandonar puzle*

Actor	Usuario
Caso de Uso	Abandonar puzle
Precondiciones	Puzle configurado y aceptado por el usuario.
Postcondiciones	Estado inicial de la aplicación
Descripción	El usuario decide no seguir resolviendo el puzle configurado y se vuelve a la pantalla principal de la aplicación.

Tabla 3.12 *- CU12-Seleccionar color activo para pintar*

Actor	Usuario
Caso de Uso	Seleccionar color activo para pintar
Precondiciones	Puzle configurado y aceptado por el usuario.
Postcondiciones	El color de la paleta seleccionado es el color activo.
Descripción	El usuario elige entre la paleta de colores uno de ellos y, en consecuencia, este color es el color activo que puede pintar en una de las caras de la figura.

Tabla 3.13 *- CU13-Eliminar color de la cara de la figura*

Actor	Usuario
Caso de Uso	Eliminar color de cara de la figura
Precondiciones	Puzle configurado y aceptado por el usuario.
Postcondiciones	Ninguna
Descripción	El usuario selecciona la herramienta de borrado y, consecutivamente, elige la cara a la que quiere quitar su color. La cara se decolora.

Tabla 3.14 *- CU14-Pintar cara de la figura con el color activo*

Actor	Usuario
Caso de Uso	Pintar cara de la figura con el color activo
Precondiciones	Puzle configurado y aceptado por el usuario. El color de la paleta seleccionado es el color activo.
Postcondiciones	Ninguna
Descripción	El usuario elige la cara a la que quiere colorear con el color activo seleccionado previamente. La cara se pinta de ese color.

4. Diseño

Una vez conocidos los requisitos a cumplir y las funcionalidades a implementar, se procede al diseño de la aplicación. Este apartado servirá como guía para estructurar la aplicación y esbozar su implementación.

4.1. Arquitectura de tres capas

La aplicación estará basada en un modelo de tres capas, muy común en el desarrollo software. Este modelo consiste en dividir la aplicación en tres niveles: negocio, persistencia y presentación. Los niveles son independientes y ayudan a estructurar internamente un programa, de forma que cualquier modificación afecte a la capa correspondiente y no al conjunto de la implementación porque complicaría todo el procedimiento [17]. En los siguientes apartados se detalla cada una de las capas o niveles.

4.1.1. Capa de presentación

La capa de presentación está compuesta por la parte gráfica que llega al usuario mediante la interfaz visual y la representación de datos en ella. En este caso, es crucial este nivel por la idiosincrasia de la aplicación referida al color y los elementos 3D. A continuación, se presentan los bocetos diseñados para el conjunto de pantallas que se han pensado como solución a los requisitos analizados y los casos de uso inferidos.

La aplicación constará inicialmente de un listado compuesto por varias figuras geométricas disponibles para su selección, como se puede ver en la Figura 4.1. Esta pantalla está relacionada con el CU1 reflejado en la Tabla 3.1. Todas las figuras deben de ser poliedros de caras planas, para la determinación de los colores en cada una de ellas y su uniforme relación. La elección de qué figuras aparecen en el listado se hará más adelante, para estudiar cuáles tienen sentido, es decir, hay que comprobar si son fáciles de visualizar y manipular en el espacio 3D, su nivel de dificultad y su viabilidad una vez se combinen en el espacio de color, entre otros aspectos.

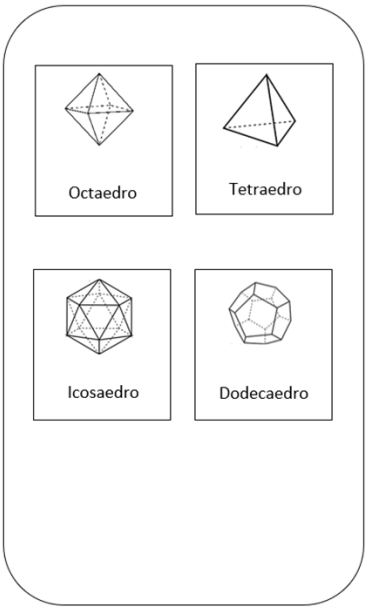


Figura 4.1 *- Pantalla listado de figuras*

Una vez se ha seleccionado una figura, se debe modificar su posición y tamaño dentro del espacio de color, representados conjuntamente en el espacio 3D. En la Figura 4.2 el espacio de color está representado por un cubo y contiene a la figura seleccionada dentro, en este caso representada por un tetraedro. Así se determina su relación, es decir, qué colores del espacio de color son los que corresponden a cada cara. Esto permitirá que la misma figura, según su tamaño y posición dentro del espacio de color, obtenga un conjunto particular de colores para sus caras.

La pantalla tiene que contener unos controles para la modificación del tamaño de la figura y su posición dentro del espacio de color. La Figura 4.2 presenta estos controles como *sliders* porque es una forma fácil e intuitiva de modificar cada uno de los valores. Al escalar se mostrará en pantalla cómo se redimensiona la figura siempre dentro del espacio de color, esto nos permite visualizar cuál es su tamaño y si los colores resultantes están más o menos relacionados entre ellos. Cuando la figura sea más grande, los colores serán más distintos entre ellos porque las caras de la figura estarán más separadas entre ellas y eso significa que colindan con colores con menos relación en el espacio de color. Se aplica el mismo sentido cuando la figura es más pequeña, de manera que los colores colindantes con las caras tienen más relación y son más parecidos. Los *sliders* para trasladar la figura dentro del espacio de color, cumplen con los casos de uso definidos, diferenciándose entre los tres ejes de coordenadas. Por lo tanto, cada uno de ellos moverá la figura según su eje, siempre dentro del espacio de color y esto condiciona qué tipo de tonalidad, saturación y luminosidad van a tener los colores resultantes para cada cara. Por último, se selecciona el número de caras con otro *slider* para unificar el aspecto de todos los controles de configuración y que así resulte fácil su uso para el usuario.

Todo este proceso se representa visualmente para saber cómo estamos manipulando el poliedro y, en consecuencia, qué tipo de puzle vamos a realizar. La biblioteca que utilicemos de 3D nos dará las herramientas necesarias para crear estos elementos gráficos.

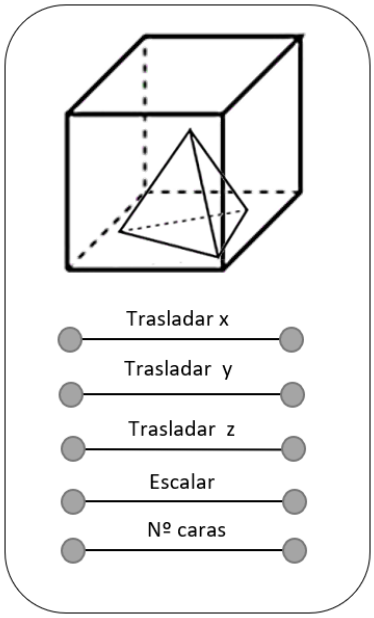


Figura 4.2 *- Pantalla de configuración*

Cuando se ha configurado el puzle, el conjunto de colores resultantes se debe mostrar al usuario para decidir si quiere resolver un puzle con esos colores dentro de la figura. Esta pantalla es necesaria para saber qué tipo de puzle se va a tener que resolver y si se está de acuerdo con esa lista de colores, antes de proceder al juego. El botón de aceptar que se ve en la Figura 4.3 procede a la pantalla de juego.

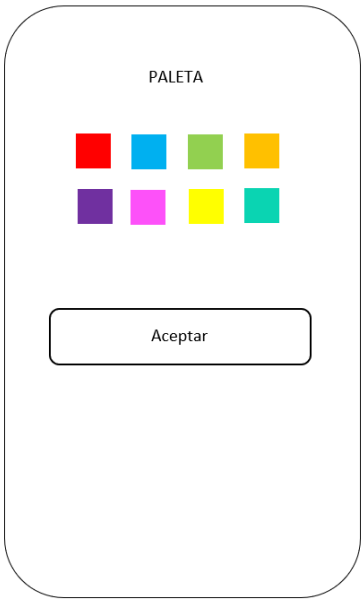


Figura 4.3 *- Pantalla paleta de colores*

Finalmente, la pantalla de juego representada en la Figura 4.4 nos muestra la paleta de colores a resolver y el poliedro seleccionado, para así empezar a jugar y colorear las caras de forma correcta. Los colores son seleccionables y se activan tal y como se explica en el CU12, detallado en la Tabla 3.12, y el botón de ‘Borrar color’ permite

decolorar la cara del poliedro que se seleccione, como se explica en el CU13, detallado en la Tabla 3.13, para que el usuario rectifique sus decisiones a la hora de resolver el puzle. También se encuentra el botón 'Cancelar' para salir de la pantalla si el usuario no quiere terminar de resolver el puzle, mencionado en el CU11, especificado en la Tabla 3.11.

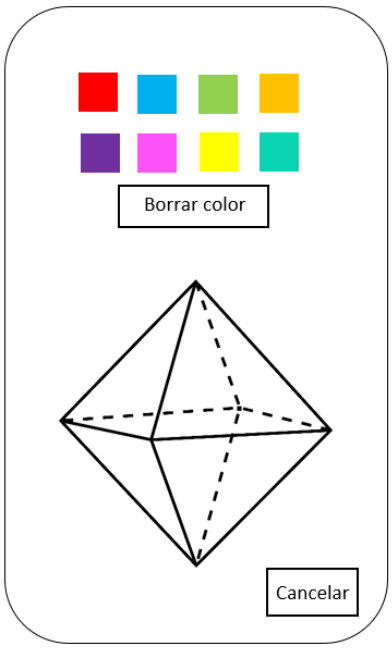


Figura 4.4 *- Pantalla de juego*

4.1.2. Capa de persistencia

La capa de persistencia se encarga de almacenar los datos necesarios para el funcionamiento de la aplicación. En esta aplicación en concreto, los datos son la información sobre las figuras geométricas que vamos a dibujar en el plano tridimensional. Son los únicos datos necesarios debido a que, a partir de una figura geométrica, el usuario configura toda la información relacionada con ella para dar paso al puzle resultante a jugar.

Los datos necesarios en el dibujo de cada figura y la resolución del puzle se explican a continuación.

- Número de vértices del poliedro. Este dato implica una figura más o menos compleja a la hora de calcular sus modificaciones dentro del espacio de color.

- Conjunto de vértices que forman el poliedro. Cada vértice se detallará como un punto en el espacio tridimensional incluyendo su valor en el eje X, Y y Z.

- Número de caras de la figura. Implica un puzle más o menos complicado debido a la cantidad de incógnitas a resolver con su color asociado. También se relaciona con la complejidad de cálculo de los colores dentro del espacio de color, cuantas más caras, más colores que calcular.

- Conjunto de caras que forman la figura. Cada cara estará expresada como el conjunto de vértices que la componen. Se necesitan las caras de la geometría para delimitar la superficie de cada una de ellas y entenderlas como unidades singulares, ya que se requiere pintarlas individualmente.

- El número mínimo de caras coloreadas por defecto al empezar el juego. Este dato se necesita porque necesitamos unos colores base en el puzle para entenderlo y resolverlo, además, es complejo si solo se pinta una cara porque no se muestra la relación total de la figura y sus colores asociados. El hecho de que todas las figuras y su color estén relacionados entre sí requiere más de un color como base y guía del puzle, y para determinar este número se tendrá en cuenta el número de caras de la figura.

4.1.3. Capa de lógica

La capa de lógica o negocio contiene la implementación de las funcionalidades de la aplicación. Esta capa se comunica, por una parte, con la capa de presentación para mostrar el resultado de los procesos que lleva a cabo y, por otra parte, con la capa de persistencia para recoger o guardar los datos necesarios en esos procesos. De manera que dirige todo el flujo de la aplicación para su correcta ejecución, según las implementaciones de todas las funcionalidades incluidas.

En el apartado de especificación de este documento se han descrito los requisitos que debe cumplir la aplicación y, a partir de ellos, las funcionalidades a implementar expresadas como casos de uso. La lógica de la aplicación se compondrá del desarrollo de estas características y, para ello, necesitamos herramientas especiales que asistan nuestras necesidades y que serán descritas en el apartado de implementación.

5. Implementación

En este apartado se conocen las tecnologías y herramientas que se utilizarán para la implementación. A continuación, se explicará con detalle qué es el espacio de color HSL, el cual será necesario comprender para aplicarlo en todo el juego. Entonces, se procederá al desarrollo de la aplicación, que irá inevitablemente unido al aprendizaje de las herramientas necesarias. Por ello, se irán abordando las funcionalidades a implementar desde lo más básico y fundamental para cumplir la idea del juego, hacia lo más complejo.

5.1. Tecnologías

A continuación, se van a exponer las tecnologías implicadas en este desarrollo. En el inicio de este trabajo se ha elegido Ionic para la implementación de esta aplicación, un *framework* que tiene varias tecnologías implicadas en su uso como son HTML, CSS, TypeScript y Angular, relacionadas con el desarrollo web.

5.1.1. HTML

Es un lenguaje de marcado de hipertexto (*HyperText Markup Language)*. En 2014 se publicó su versión actual de HTML 5. Este lenguaje se escribe mediante elementos compuestos de etiquetas, contenido y atributos. Las etiquetas delimitan el inicio y final cada elemento, dentro reside el contenido y en la primera etiqueta se determinan los atributos del elemento, descritos con un nombre referido al atributo y su valor. Con esa sencilla estructura se componen los amplios contenidos que visualizamos en las páginas web y se establece su orden.

5.1.2. CSS

Las hojas de estilo en cascada (*Cascading Style Sheets)* es un lenguaje de estilos aplicado a desarrollo web para definir la representación del documento HTML y modificar sus atributos referidos a la renderización en pantalla. Cada documento HTML está relacionado con un documento CSS que lo diseña, modificando la fuente de letra, el color y el tamaño de los distintos elementos, el espaciado y la estructuración, entre otras propiedades disponibles.

5.1.3. TypeScript[39]

Publicado por Microsoft en 2012 como una solución para el desarrollo web a gran escala con JavaScript, debido a las carencias que presentaba en ese ámbito. Los dos lenguajes son totalmente compatibles porque el compilador de TypeScript se traduce a código JavaScript, proporcionando acceso a todas las librerías y *frameworks* que existen para JavaScript. Su principal característica distintiva es el tipado estático, es decir, cada variable tiene un tipo asociado y se le pueden asignar solo valores de ese tipo.

5.1.4. Angular[40]

Desarrollado y mantenido por Google desde 2010. Se ofrece como solución para el desarrollo *front end* de una página web, basado en la estructura de aplicaciones web de una sola página. Permite que cada página sea una aplicación web con sus componentes HTML, JavaScript (o TypeScript) y CSS asociados. Hace uso del patrón MVC[41] (Modelo Vista Controlador) que permite tener separados los elementos de la lógica de la aplicación, la vista y los datos, proporcionando un *data binding* bidireccional entre el modelo y la vista.

5.1.5. Three.js[42]

Librería de JavaScript para mostrar animaciones 3D en el navegador web con la ayuda de WebGL[43], una API en JavaScript para la renderización de gráficos en páginas web a través del elemento ‘canvas’ de HTML. WebGL se destaca por su estandarización en todos los navegadores web.

Incorpora una amplia gama de recursos para crear escenas tridimensionales con elementos como luces, animación de objetos, cámaras, importación de archivos 3D, perspectivas y recursos para desarrollar aplicaciones para realidad virtual o *VR (Virtual Reality)*.

En este proyecto se empleará para crear los espacios tridimensionales y con ellos las figuras geométricas con sus elementos primordiales como caras, aristas y vértices. Su elección se basa en la cantidad de ejemplos y ayuda que se puede encontrar con respecto a su uso, lo cual ayudará a su aprendizaje y planteamiento a lo largo de la implementación.

[39] https://www.typescriptlang.org/
[40] https://angular.io/
[41] https://developer.mozilla.org/es/docs/Glossary/MVC
[42] https://threejs.org/
[43] https://www.khronos.org/webgl/

5.2. Herramientas

Junto con las tecnologías mencionadas, se necesitan herramientas que hagan posible su uso y que asistan el desarrollo de este trabajo.

5.2.1. Visual Studio Code[44]

Microsoft publicó este editor de código fuente gratuito y de código abierto, el cual es compatible con los lenguajes de programación más utilizados. Se ha popularizado por su diseño básico y sencillo puesto que muestra pocos controles en pantalla, aunque dispone de consola de comandos para proporcionar acceso a todas sus características, lo cual resulta muy útil.

Permite depurar código, resaltar la sintaxis para mejorar su comprensión, contiene funciones de autocompletado y dispone de una tienda de *plugins* para añadir funcionalidades extra. También se destaca su fácil integración con Git para el control de versiones del código desarrollado.

Se hace uso de este entorno de desarrollo debido a su sencillez y la incorporación de la terminal en el entorno, ya que se usará notablemente a lo largo del desarrollo.

5.2.2. Git[45]

Control de versiones para registrar los cambios en el código y asegurarlos subiéndolos a la nube. Es distribuido, diferenciándose de otros centralizados como Subversion[46], esto quiere decir que se replica todo el repositorio donde se almacena el código, en lugar de depender de la conexión a la red para acceder al repositorio centralizado con todo el código y versiones almacenadas. Mediante comandos se pueden realizar todas las operaciones.

Es muy conveniente en proyectos en los que participan varias personas para gestionar todas las modificaciones, e igualmente para almacenar el código y garantizar su accesibilidad.

[44] https://code.visualstudio.com/
[45] https://git-scm.com/
[46] http://subversion.apache.org/

5.2.3. Ionic CLI[47]

CLI o *command-line Interface*, es un intérprete por línea de comandos que ofrece Ionic como herramienta para facilitar procedimientos habituales en el desarrollo de aplicaciones con Ionic. Algunas de estas tareas son crear un proyecto Ionic, añadir una página con sus correspondientes archivos para implementarla de funcionalidad o ejecutar la aplicación que se está desarrollando.

5.2.4. Google Chrome[48]

Google creó este navegador web en 2008 y desde entonces es uno de los más usados. A parte de navegar por sitios web, es conocido su uso para el desarrollo web porque proporciona herramientas muy útiles para los desarrolladores. En este caso servirá como medio de ejecución de la aplicación para ver el resultado de lo que se va implementando. La consola de salida será de ayuda cuando ocurran errores y servirá como recurso de testeo de nuestras funcionalidades. A parte, a la hora de ejecutar la aplicación en el teléfono móvil, Google Chrome también proporciona herramientas de utilidad como la visualización de bases de datos.

5.2.5. Node.js[49]

Entorno de ejecución basado en JavaScript cuyo principal fin es gestionar la capa del servidor, pero aporta más herramientas al desarrollador. Es necesario para instalar Ionic.

5.2.6. Npm[50]

Sistema de gestión de paquetes de JavaScript que va unido a Node.js. Permite instalar paquetes fácilmente a través de la línea de comandos y gestionar sus dependencias en la aplicación.

Necesario para preparar el entorno con Ionic e instalar herramientas adicionales como Three.js, las cuales se expondrán en el apartado de implementación.

[47] https://ionicframework.com/docs/cli
[48] https://www.google.com/chrome/
[49] https://nodejs.org/es/
[50] https://www.npmjs.com/

5.2.7. Google Drive[51]

Servicio para el almacenamiento de archivos proporcionado por Google. Incorpora editores de archivos de texto y hojas de cálculo, entre otros. Su utilidad en este caso será guardar este trabajo para asegurar sus distintas versiones y hacer copias de seguridad de los recursos necesarios durante el desarrollo.

5.3. Espacio de color HSL

El espacio de color juega un gran papel en esta aplicación. Es necesario entender este concepto y comprender el espacio de color que se vaya a aplicar. Para este trabajo se propuso el espacio de color HSL porque es estudiado por los alumnos a los que va dirigido y se caracteriza por ser muy intuitivo.

Un espacio de color recoge los colores en un modelo espacial aplicando un orden concreto que sirva para generalizar la especificación del color. El espacio tridimensional representará en cada uno de sus puntos un color, especificado por el valor de sus coordenadas. Su representación como cilindro se presenta en la Figura 5.1.

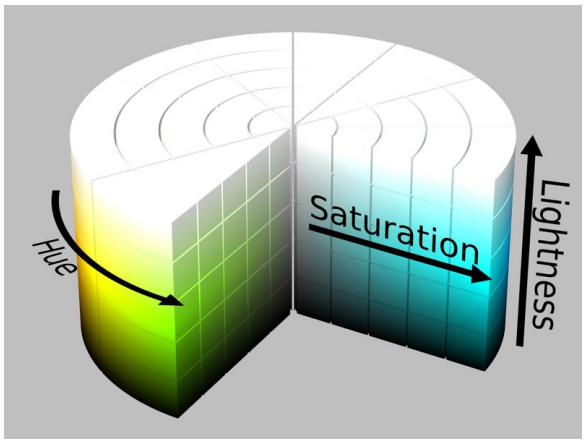


Figura 5.1 *- Representación del espacio de color HSL*

HSL es un espacio de color cuyas siglas denominan sus tres propiedades para describir un color: *Hue, Saturation, Lightness* o Tono, Saturación y Luminosidad [17]. Se puede representar gráficamente de distintas formas, pero sus características son las mismas. El tono es determinado por el ángulo formado respecto a un punto de referencia desde el eje central del espacio, el valor de este ángulo está entre 0º y 360º, lo que conforma un círculo con todos los tonos, como se muestra en la Figura 5.2.

[51] https://drive.google.com/drive/

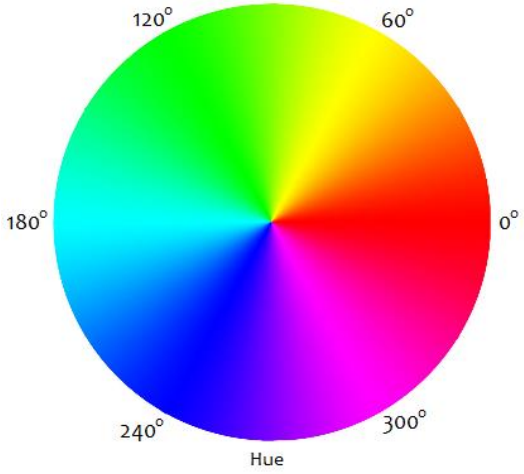


Figura 5.2 *- Representación del tono del espacio de color HSL*

La saturación es la distancia horizontal desde el eje vertical del espacio representado hasta el punto del color, o la posición en el radio del cilindro, normalmente se representa desde 0% hasta el 100%, y se entiende como la cantidad de color en proporción al tono, tendiendo a gris cuando disminuye, como muestra la Figura 5.3.

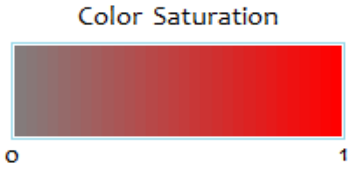


Figura 5.3 *- Representación de la saturación del espacio de color HSL*

La luminosidad está representada por el eje vertical del espacio representado, normalmente se especifica desde el 0% al 100% y representa la oscuridad relativa del color, siendo 0% negro y 100% blanco, tal como se ve en la Figura 5.4.

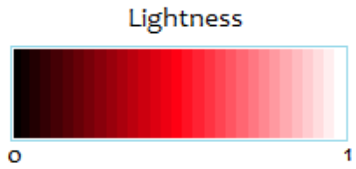


Figura 5.4 *- Representación de la luminosidad del espacio de color HSL*

5.4. Preparación del proyecto

Para empezar la implementación de ColorDoku3D, se deben instalar todas las herramientas necesarias y asegurar un entorno listo para comenzar a programar.

5.4.1. Instalación Ionic

Antes de instalar Ionic, hay que asegurarse de preparar el entorno instalando Node.js y npm, dos herramientas que se utilizan en la mayoría de los proyectos de JavaScript y son indispensables para proceder a esta instalación. Se debe descargar la opción conveniente en cada caso desde la página principal de Node.js, en este caso se ha elegido la destinada a Windows, y se procede a su instalación. Esta instalación deberá incluir las dos herramientas nombradas, una vez instaladas, se verifica que las herramientas están disponibles para el desarrollo con los comandos de la Figura 5.5.

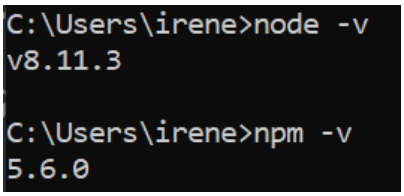


Figura 5.5 *- Verificación de la instalación de node y npm*

A continuación, se instala Cordova e Ionic con el instalador de paquetes npm. Después se comprueba que están instalados utilizando la terminal e indicando los comandos que se muestran a continuación.

$ npm install -g cordova

$ npm install -g @ionic/cli

Este último comando muestra la instalación de Ionic junto a su herramienta Ionic CLI, la cual va a resultar muy útil para generar todo lo necesario en el proyecto.

5.4.2. Control de versiones

Procedemos a la instalación de Git, el control de versiones elegido. Para este desarrollo se descarga su opción para Windows[52] y se efectúa la instalación.

Se hará uso de él en todo el desarrollo para gestionar las versiones de código que se vayan implementando y asegurando el proyecto.

[52] https://git-scm.com/download/win

5.4.3. Creación del proyecto y puesta en marcha

Después de preparar todo el entorno, se procede a crear el proyecto para empezar con su desarrollo. Se usan las herramientas de Ionic CLI para crear un nuevo proyecto llamado 'ColorDoku3D' como se puede ver a continuación. La herramienta de Ionic ofrece diferentes plantillas para generar proyectos con un modelo preestablecido como *tabs* o *side menu* para crear cada tipo de aplicación rápidamente, en este caso se elige *blank* porque no se requiere ningún modelo previo.

$ ionic start ColorDoku3D blank

Después de generarse el proyecto, se puede comprobar en la Figura 5.6 cómo se ha generado una página, cuyo contenido está vacío, con sus respectivos archivos para el código HTML, su diseño CSS y la lógica de la página en el archivo 'home.ts' de TypeScript. Todas las páginas se compondrán de estos tres archivos para su desarrollo propio y se enlazarán a través de la navegación que se implementará más adelante.

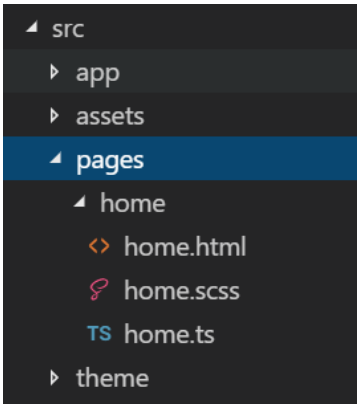


Figura 5.6 *- Estructura de páginas en Ionic*

Para ejecutar el proyecto que se ha creado en el navegador web, y posteriormente, en todo su desarrollo, se ejecutará un comando de Ionic esencial. Una vez ejecutado se abrirá una pestaña de Google Chrome ejecutando la aplicación y se podrán utilizar sus herramientas para desarrolladores para comprobar el funcionamiento deseado de las implementaciones que se vayan añadiendo. El comando que hay que ejecutar para la ejecución es el siguiente.

$ ionic serve

5.4.4. Instalación de Three.js

Para este proyecto en necesario el uso de una biblioteca JavaScript que asista la necesidad de generar espacios 3D y dibujar geometrías en ellos. Se ha escogido Three.js porque es una librería popular, la cual disponible de mucho material para aprender su uso y aportar soporte en los momentos que no se tenga clara su

aplicación. Particularmente en este proyecto, en el que no se tienen conocimientos previos sobre proyección tridimensional y se requiere el aprendizaje de los principales elementos en el diseño 3D, es necesaria una buena documentación de la herramienta.

Su instalación se lleva a cabo a través de npm, ejecutando el siguiente comando desde la carpeta del proyecto con ayuda de la terminal.

$ npm install --save three

Después de su instalación, se debe importar la librería en la página de Ionic que requiera su uso a través de la referencia de la Figura 5.7. Mediante la palabra 'THREE' se obtendrán sus funcionalidades en el código.

```
import * as THREE from 'three';
```

Figura 5.7 *- Importación de Three.js*

5.5. Desarrollo de la aplicación

En este apartado se aborda el desarrollo de la aplicación a nivel técnico, en el orden en el que se acuerda con los tutores, proponiendo objetivos a cumplir basados en las funcionalidades requeridas y expresadas en el apartado de especificación.

5.5.1. Pantalla de juego

Lo primero que se va a desarrollar es la pantalla principal del juego para introducirse en el uso de la librería Three.js. Se va a investigar y aprender cómo implementar el dibujo de las figuras en el espacio tridimensional, la selección de sus caras individualmente y su cambio de color.

Se comienza por realizar los pasos básicos para utilizar las herramientas de Three.js e introducir una figura geométrica básica. La primera geometría para implementar es un icosaedro. Los datos de esta figura han sido proporcionados por el tutor Juan Serra Lluch exportados desde un programa de modelado tridimensional. A continuación, se va a explicar cómo se configura toda la escena 3D con Three.js y servirá como ejemplo de la implementación de las distintas figuras geométricas que se incluirán más adelante.

Preparación de Three.js

En el apartado de preparación se indica cómo importar la librería Three.js para permitir su uso, esto se debe realizar en cada página que queramos añadir una escena 3D, en este caso se trata de la pantalla de juego. Los pasos que seguir en cualquier aplicación de Three.js son los siguientes.

1. Crear escena. Lo primero de todo es crear la escena que representará el espacio tridimensional en el código, a ella se irán añadiendo los elementos que se necesiten como pueden ser los objetos 3D, cámaras para visualizarla, luces y sonidos, entre otros.

```
this.scene = new THREE.Scene();
```

Figura 5.8 *- Creación de escena Three.js*

2. Crear cámara. Este elemento permite visualizar la escena en la que se encuentra. Se pueden añadir varias cámaras en una misma escena para visualizarla desde distintos puntos, pero solo podrá estar activa una de ellas en cada momento.

   Existen varios tipos de cámaras con distintas proyecciones para visualizar el espacio 3D. En este proyecto se va a utilizar PerspectiveCamera[53], la más común debido a que simula la visión humana, es decir, deforma los objetos según la posición y distancia en la que se encuentren respecto a la cámara. PerspectiveCamera tiene los siguientes parámetros disponibles para su configuración.

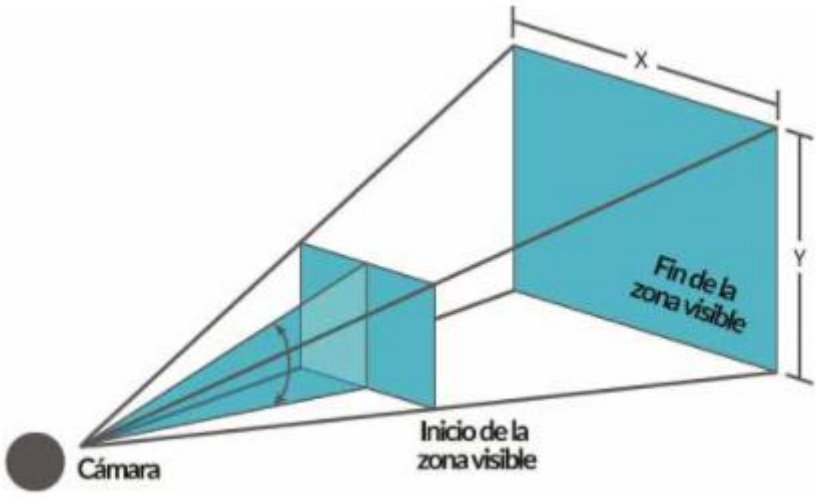


Figura 5.9 *- Representación de PerspectiveCamera*

[53] https://threejs.org/docs/#api/en/cameras/PerspectiveCamera

- Fov (*Camera frustum vertical field of view*): ángulo especificado en grados que define el campo visual vertical que se visualiza con la cámara, de abajo hacia arriba. El campo visual horizontal se calcula a partir del vertical.
- Aspect (*Camera frustum aspect ratio*): el ratio del *fov* o la relación de aspecto de la cámara. Normalmente es el ancho de la ventana donde se proyecta la escena dividida entre su alto, para que se visualice como es esperado y no se produzcan deformaciones.

- Near: indica el inicio de la zona visible, situada entre la cámara y el fin de la zona visible.

- Far: define el fin de la zona visible. Si algún objeto se posiciona más lejos, no será visible para la cámara y no se podrá visualizar al mostrar la escena en pantalla.

```
this.camera = new THREE.PerspectiveCamera( 75, this.divWidth/this.divHeight, 0.1, 1000 );
```

Figura 5.10 *- Implementación de PerspectiveCamera*

En la Figura 5.10 se muestra el código con los parámetros que se han pasado para crear la PespectiveCamera. Un ángulo de 75º para un campo visual amplio. La relación de aspecto es el ancho y el alto del elemento HTML en el que se representará la escena tridimensional. La zona visible es amplia para no tener problemas al visualizar los elementos 3D que se añadan a la escena.

3. Añadir objeto. El propósito del espacio tridimensional que se está creando es representar una figura geométrica concreta. La primera figura que se va a representar es un icosaedro, un poliedro formado por 12 vértices y 20 caras triangulares. Los datos disponibles inicialmente para su dibujado son los valores de sus vértices, calculados con una herramienta de modelado 3D. En Three.js son imprescindibles dos elementos para crear una figura tridimensional, detallados a continuación.

   - Geometría

   Instancias de la clase Geometry[54] formadas por los vértices y las caras de la geometría. Los vértices son puntos en el espacio representados con instancias de Vector3[55] y las caras representan triángulos que unen tres puntos definidos a través de los vértices, instanciadas por Face3[56].

   Three.js dispone de geometrías definidas para agilizar el proceso, por ejemplo, BoxGeometry para representar cubos. En este proyecto los datos de las figuras son particulares, por lo tanto, no se puede hacer uso de estas geometrías definidas genéricamente. A partir de los vértices, se deben calcular las caras de cada figura

[54] https://threejs.org/docs/#api/en/core/Geometry
[55] https://threejs.org/docs/#api/en/math/Vector3
[56] https://threejs.org/docs/#api/en/core/Face3

manualmente con el siguiente proceso, ya que es muy importante el orden en el que especificamos los vértices de cada cara. Conocido el orden de los vértices a dibujar, éste se utiliza para establecer los tres vértices que forman cada cara en el sentido contrario a las agujas del reloj cuando esa cara esté enfrente de la cámara.

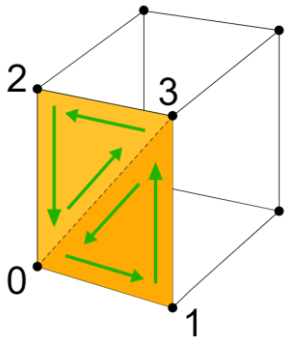


Figura 5.11 *- Definición de los vértices de cada cara*

Por ejemplo, en la Figura 5.11 se reflejan dos caras que se definirían con Face3 indicando los vértices de cada una de ellas en el orden que muestran las flechas, el cual sigue el sentido contrario a las agujas del reloj cuando se posiciona enfrente de nuestro campo visual, lo que corresponde a la cámara que tengamos en la escena. Como se ha comentado es un proceso manual debido a que las figuras son personalizadas con datos propios para sus vértices, y requiere hacer un ejercicio de percepción espacial para imaginar la figura definidos sus vértices e ir determinando cada una de sus caras.

- Material

La geometría define la figura y el material la viste. Three.js proporciona una gran colección de materiales para modificar características como la visibilidad de las caras, el color, el comportamiento de la luz en la figura, las aristas y el grado de transparencia, entre otras. En este proyecto se utilizará MeshBasicMaterial[57], el cual es un material no afectado por la luz y muestra los colores tal y como se definen, una característica imprescindible para facilitar la percepción de los distintos colores que se mostrarán.

Finalmente, la geometría y el material se unen definiendo un objeto Mesh[58]. Este objeto es el que se añadirá a la escena representando la figura geométrica en su conjunto y su comportamiento dentro del espacio tridimensional.

[57] https://threejs.org/docs/#api/en/materials/MeshBasicMaterial
[58] https://threejs.org/docs/#api/en/objects/Mesh

```
this.geometry = new THREE.Geometry()
this.geometry.vertices = this.vertices
this.geometry.faces = this.faces
this.material = new THREE.MeshBasicMaterial( { color: 0xffffff, vertexColors: THREE.FaceColors } );
this.mesh = new THREE.Mesh( this.geometry, this.material );
this.scene.add(this.mesh);
```

Figura 5.12 *- Implementación de la figura geométrica*

4. Renderizado. Por último, falta indicar dónde deberá mostrarse la composición definida dentro de la página web. Para ello es necesario el uso de *renders*, en este caso se utiliza WebGL para crear el elemento dónde se pintarán los gráficos y añadirlo a la página.

```
this.renderer = new THREE.WebGLRenderer({canvas: this.canvasThree.nativeElement});
this.renderer.render(this.scene, this.camera);
```

Figura 5.13 *- Implementación renderizado*

Controles para manipular la geometría

Lo primero, es conseguir que la figura se pueda ver en su totalidad, es decir, permitir su rotación para que sea posible seleccionar cualquiera de sus caras y así completar el puzle. Para ello, se importa la librería externa OrbitControls.js, la cual permite a la cámara orbitar alrededor de un objetivo, en este caso la figura. Este tipo de movimiento de cámara va a permitir seleccionar la figura y moverla. En realidad, nos permite mover la cámara que ayuda a ver la escena, alrededor de la geometría, pero da la sensación de que estamos girando la figura y de este modo se pueden visualizar cualquiera de sus caras.

```
this.orbitControls = new OrbitControls(this.camera, this.divThree.nativeElement);
this.orbitControls.addEventListener( 'change', ((event)=>{
  this.render();
}) );
```

Figura 5.14 *- Implementación de OrbitControls.js*

En la Figura 5.14 se muestra cómo se ha de implementar esta librería. En su creación se indica como primer parámetro la cámara de la escena y como segundo parámetro el elemento HTML al que afecta. Seguidamente, se añade un *listener* para renderizar los cambios que se producen en la figura con la rotación que se aplica con esta librería, y así hacerlos visibles.

Adicionalmente, este control nos ayuda a redimensionar la figura geométrica. Así se puede modificar el tamaño de la figura, según la necesidad del usuario para visualizarla mejor y ayudar a completar el puzle.

Detectar las caras de la geometría

Dado que la figura está dibujada en la pantalla y es manipulable, el siguiente objetivo es detectar cada una de sus caras para que el usuario pueda seleccionarlas individualmente y colorearlas una a una con ayuda de la paleta de colores. Three.js provee de la clase Raycaster[59] para este fin.

```
this.mouse = new THREE.Vector2();
this.mouse.x = ( ( event.clientX - this.renderer.domElement.offsetLeft ) / this.renderer.domElement.width ) * 2 - 1;
this.mouse.y = - ( ( event.clientY - this.renderer.domElement.offsetTop ) / this.renderer.domElement.height ) * 2 + 1;

this.raycaster = new THREE.Raycaster();
this.raycaster.setFromCamera( this.mouse, this.camera )
var intersects = this.raycaster.intersectObject( this.mesh)
```

Figura 5.15 *- Implementación control Raycaster*

Raycaster opera interceptando el ratón sobre el objeto 3D que se indique. En este caso, interceptará los gestos en la pantalla una vez se ejecute la aplicación en el móvil. Como se puede ver en la Figura 5.15, Raycaster se configura con el cálculo del ratón y la cámara de la escena, aplicándose después sobre un objeto de la escena con el método 'intersectObject()'. Este último método da como resultado las intersecciones que se han producido entre la selección con el ratón y el objeto indicado, puede contener descendentes. Las intersecciones resultantes permitirán conocer qué cara se ha seleccionado para poder modificarla, ya sea pintándola de un color o borrándolo.

```
this.mesh.geometry.faces[intersects[0].faceIndex].color.setHSL(this.color[0],this.color[1],this.color[2]);
```

Figura 5.16 *- Implementación del coloreado de la cara seleccionada*

Creación de la paleta de colores

La paleta mostrará todos los colores que compongan cada puzle en la pantalla de juego, para seleccionarlos y poder pintar las caras de la figura. Para su implementación se ha estudiado el uso de la librería Three.js con este fin, ya que puede mostrar objetos de dos dimensiones, pero se descartó esta opción porque complicaba la representación junto a la figura geométrica dentro del espacio tridimensional. En consecuencia, se ha optado por usar los elementos gráficos SVG de HTML para representar rectángulos con las etiquetas 'rect' y modificar sus atributos para representar cada color dentro de una paleta, haciendo uso de CSS para su implementación. Junto a estos colores se añade otro elemento gráfico para realizar la función de borrado de color.

Una vez la paleta es mostrada en la pantalla, se ha de implementar su funcionalidad cumpliendo los requisitos definidos en el apartado de especificación. Hay que implementar estos requisitos en la capa de negocio y en la capa de presentación. La

[59] https://threejs.org/docs/#api/en/core/Raycaster

representación del color activo se muestra como un borde blanco sobre el color activo, como se muestra en la Figura 5.17.

Figura 5.17 *- Representación de color activo*

Este color será el que esté activo y se pintará en la cara que se presione de la figura. Una vez se haya pintado en una cara, no podrá pintarse en las demás. Los colores que están pintados en la geometría se representan tachados por una cruz para señalar que no están disponibles, como en la Figura 5.18.

Figura 5.18 *- Representación de color no disponible*

Se ha añadido la funcionalidad de poder borrar el color activo de la cara en la que está pintado para facilitar al usuario probar ese color en varios sitios seguidamente, sin tener que utilizar la herramienta de borrado de color cada vez que quiere probarlo en otra cara distinta.

Finalmente, en la Figura 5.19 se muestra la pantalla de juego en su totalidad, con la paleta de colores, la herramienta de borrado junto a los colores y la figura geométrica para resolver a la derecha de la pantalla.

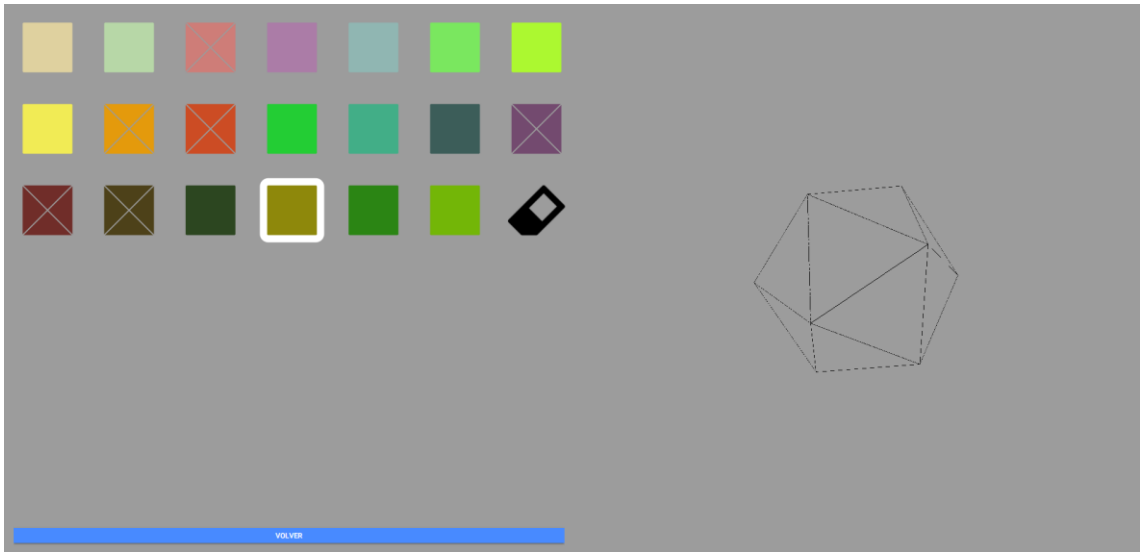


Figura 5.19 *- Pantalla de juego*

5.5.2. Pantalla de configuración del juego

La pantalla de configuración presenta un desafío matemático a la hora de posicionar y modificar la posición de la geometría que elijamos, dentro del espacio de color. Esta combinación nos ayudará a componer el puzle que se presentará como juego posteriormente con la pantalla de juego ya implementada.

Representación del espacio de color

El espacio de color HSL se representará en el espacio tridimensional como un cubo de aristas de color púrpura. El motivo de su representación como un cubo es debido a que Three.js ofrece facilidades para colorear las caras de las geometrías a través de la especificación HSL con su clase Color[60], como se puede ver en la siguiente Figura 5.20. Esta clase ayudará en la pantalla de resolución del puzle como herramienta para implementar toda la dinámica de pintado y borrado de colores.

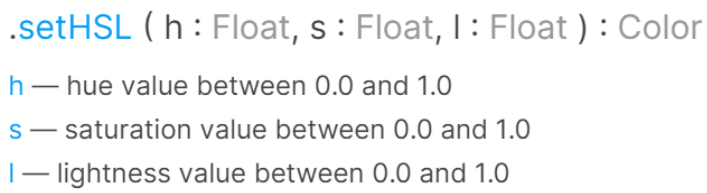


Figura 5.20 *- Método modificación color HSL*

El tono, la saturación y la luminosidad son representados por valores comprendidos entre 0 y 1. En el anterior apartado, dedicado a la comprensión del espacio de color, se puede comprobar que tanto la saturación y la luminosidad son fácilmente traducibles a estos valores, ya que normalmente se especifican desde el 0% hasta el 100%. La diferencia está en el tono porque su valor angular comprendido entre 0º y 360º se interpretará como un valor entre 0 y 1, de manera que se pueden representar los tres ejes iguales, formando un cubo que represente el espacio de color en su totalidad como en la Figura 5.21.

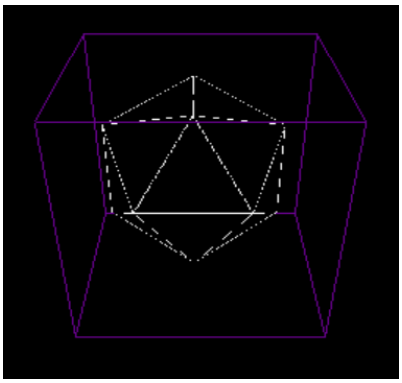


Figura 5.21 *- Representación del espacio de color HSL en cubo*

[60] https://threejs.org/docs/#api/en/math/Color

Controles de configuración

Lo primero que hay que averiguar es cómo representar los controles descritos en los casos de uso CU2, CU3, CU4, CU5 y CU6, los cuales modifican la posición y el tamaño de la figura geométrica dentro del espacio de color. Para desempeñar estas funciones se escoge el uso de *sliders* porque permiten seleccionar fácilmente un valor a través del control deslizante entre un rango de valores definido, resultando más intuitivo para el usuario. Su representación se muestra en la Figura 5.22.

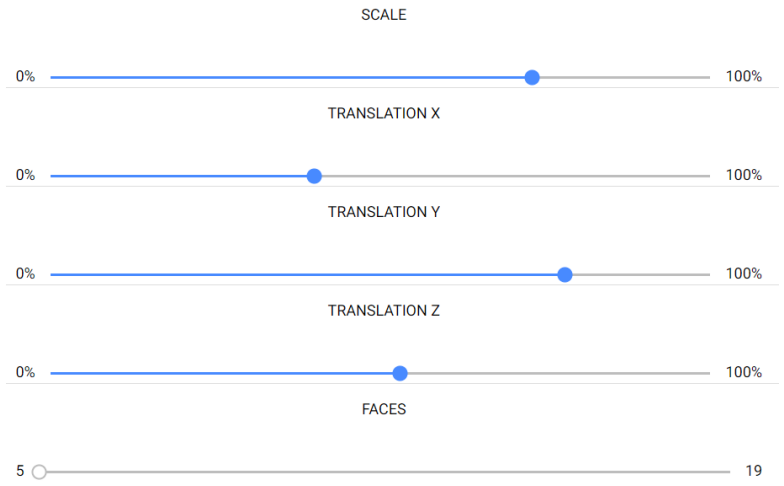


Figura 5.22 *- Representación controles de configuración*

Escalado de la geometría

El escalado de la figura dentro del espacio de color se selecciona con un valor entre el 0% hasta el 100%, esto quiere decir que la figura se puede redimensionar desde un tamaño nulo o 0% hasta su tamaño más grande o 100%, sin sobrepasar los límites del espacio de color. Por esta razón, los datos origen de las geometrías están calculados como el máximo tamaño que la figura puede adoptar dentro del espacio de color, así se tiene un punto de referencia máximo con el que se puede calcular el escalado de la figura a través de la multiplicación de los vértices por valores menores.

Cada vez que se escala la figura se posiciona en un punto de origen definido, reiniciando los valores que se hayan configurado sobre el desplazamiento de la figura, ya que es necesario tener dos puntos de referencia para calcular los vértices redimensionados de la figura y poder modificar después su posición, un origen y su tamaño máximo dentro del espacio de color. En la siguiente Figura 5.23, se puede comprobar estos puntos de referencia y cómo la figura tiene una posición fijada como origen después de cada cambio en el valor de escalado.

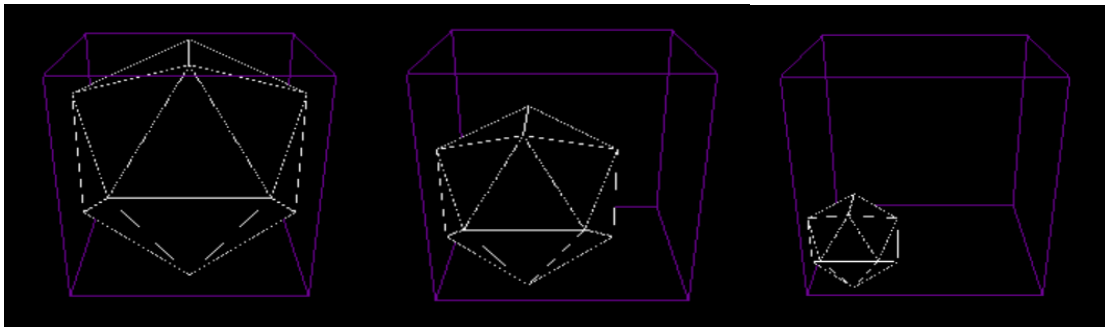


Figura 5.23 *- Escalado al 100%, 72% y 38% respectivamente*

Desplazamiento de la geometría

El cambio de posición de la figura geométrica dentro del espacio de color se realiza a través de los tres ejes que conforman el espacio tridimensional. Cada uno de estos ejes tiene un control específico para el desplazamiento de la figura sobre él, sin sobrepasar los límites del espacio de color. El valor que adoptan mediante el *slider* está comprendido entre el 0% y el 100%, es decir, tomando como referencia el punto origen que se ha elegido en el escalado, la figura se desplazará en su eje desde ese punto de origen o 0%, hasta lo máximo que le permita el espacio de color o 100% del mismo eje. Este cálculo se realiza con los 3 ejes por igual a través de la comprobación del espacio disponible en el eje, el espacio ocupado por la figura según su tamaño y su posición a través del valor elegido por el *slider*.

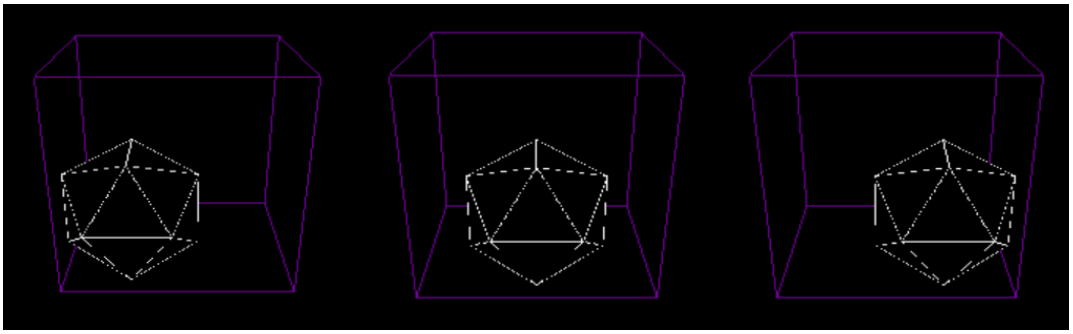


Figura 5.24 *- Desplazamiento de X al 0%, 50% y 100% respectivamente*

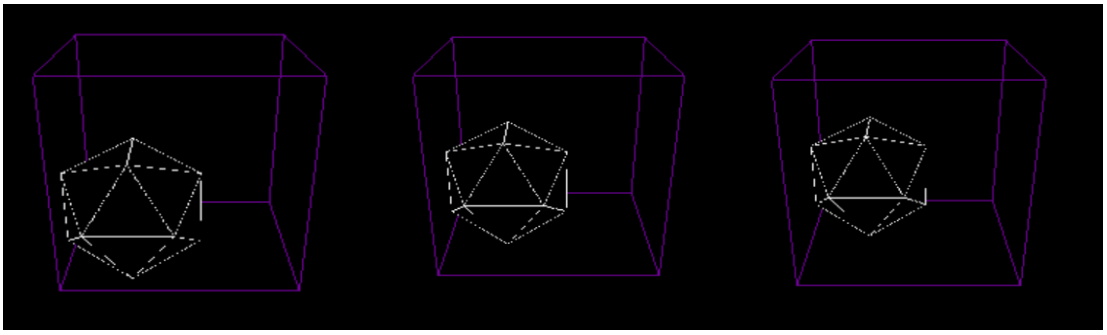


Figura 5.25 *- Desplazamiento de Y al 0%, 50% y 100% respectivamente*

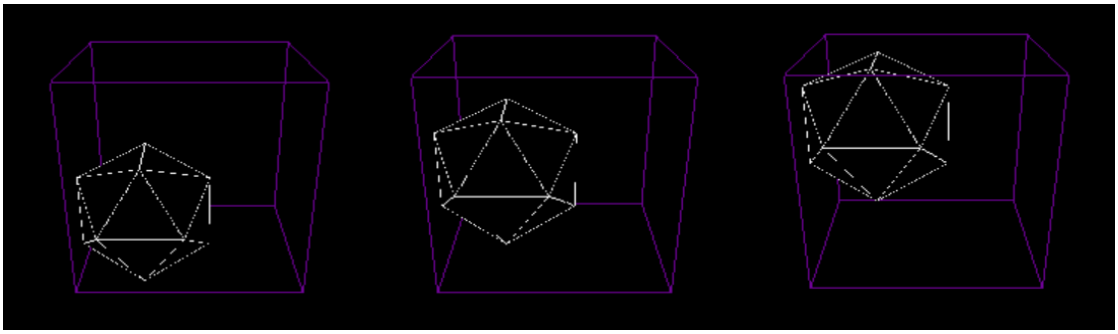


Figura 5.26 *- Desplazamiento de Z al 0%, 50% y 100% respectivamente*

Cálculo de los colores del puzle en el espacio de color

Cuando se han configurado los controles disponibles y la geometría tiene una posición concreta en el espacio de color, se procede a calcular cuáles son los colores que se asocian a cada cara de la figura.

Lo primero que se debe hacer es calcular los baricentros de cada cara, es decir, su punto central. Ese punto será el color que se considera colindante respecto al espacio de color. Los baricentros son el punto donde concurren las tres medianas de cada triángulo. Las medianas se calculan como la media aritmética de las tres coordenadas y su resultado es el baricentro de cada triángulo.

$$Baricentro = \left(\frac{Xa + Xb + Xc}{3}, \frac{Ya + Yb + Yc}{3}, \frac{Za + Zb + Zc}{3}\right)$$

Figura 5.27 *- Fórmula de cálculo del baricentro*

Calculados los baricentros se procede a calcular cada color con sus coordenadas. El cálculo de los colores es equivalente a calcular cada propiedad del espacio de color HSL. Estos cálculos están condicionados por nuestro espacio de color representado como un cubo y los valores que se requieren para representar los colores con Three.js, como se describía en la Figura 5.20.

El cálculo del tono o *Hue* en el espacio de color HSL representado debe dar como resultado un ángulo en grados, y se dividirá siempre por 360 debido a que lo pasaremos como parámetro al método que se ha mencionado de Three.js como un parámetro cuyo valor está comprendido entre 0 y 1. Las sumas o restas que se aplican al valor calculado al principio como H*, tienen su relación con el espacio de color.

$$H^* = arctan\ \frac{y}{x}$$

$$Hue = \begin{cases} \frac{180\ -\ H^*}{360} & si\ x < 0 \quad y \geq 0 \\ \frac{180\ +\ H^*}{360} & si\ x < 0 \quad y < 0 \\ \frac{360\ -\ H^*}{360} & si\ x \geq 0 \quad y < 0 \end{cases}$$

Figura 5.28 *- Fórmula de cálculo del tono*

```
getHColorHSL(x, y, z){
  var alfa = this.degrees( Math.atan( y / x ) )

  if(y>=0 && x<0) alfa =  (180 - Math.abs(alfa))
  else if(y<0 && x<0) alfa =  (180 + Math.abs(alfa))
  else if(y<0 && x>=0) alfa = (360 - Math.abs(alfa))

  return alfa /360
}
```

Figura 5.29 *- Implementación del cálculo del tono*

La saturación o *Saturation* se calcula con los valores de las coordenadas X e Y del punto en el espacio de color. Después se divide entre 100 porque su valor debe estar comprendido entre 0 y 1.

$$Saturation = \frac{\sqrt{\frac{x^2}{y^2}}}{100}$$

Figura 5.30 *- Fórmula de cálculo de la saturación*

```
getSColorHSL(x, y, z){
  var auxS = Math.sqrt( Math.pow(x, 2) + Math.pow(y, 2) )

  return (auxS / 100);
}
```

Figura 5.31 *- Implementación del cálculo de la saturación*

La iluminación o *Lightness* está directamente relacionada con la coordenada Z del punto en el espacio de color, debido a que esta propiedad se refleja en la totalidad de un eje. Igual que las otras dos propiedades, se busca como resultado un valor comprendido entre 0 y 1, por lo tanto, se divide entre 100, ya que el valor máximo de cada eje es éste mismo.

$$Lightness = \frac{z}{100}$$

Figura 5.32 *- Fórmula de cálculo de la iluminación*

```
getLColorHSL(x, y, z){
  return z / 100;
}
```

Figura 5.33 *- Implementación del cálculo de la saturación*

5.5.3. Navegación entre páginas

Las pantallas que se van implementando como páginas deben unirse con la navegación deseada entre ellas. Concretamente en esta aplicación, la navegación es lineal entre las pantallas, ya que se elige una figura al principio del juego y después se navega por las mismas pantallas secuencialmente.

Existen varios métodos de navegación en Ionic, por ejemplo, usando Angular. Se ha optado por la opción por defecto que ofrece Ionic, ya que cuando genera una página incluye en su constructor NavController y NavParams, encargados de la navegación entre páginas y el paso de parámetros, como se muestra en la Figura 5.34.

```
export class GeometriesPage {

  constructor(public navCtrl: NavController, public navParams: NavParams) {

  }
```

Figura 5.34 *- Uso de NavController y NavParams*

Se va a ejemplificar un evento de navegación entre páginas, concretamente cuando el usuario elige una de las figuras listadas como opciones de planos para el puzle. Primero, asociamos el botón de la figura en su archivo HTML con un método concreto cuya función será redirigir a la siguiente pantalla con la elección del icosaedro.

```
<ion-content padding>
    <button ion-button
    block
    color="primary"
    (click)="goIcosahedron()">
    Icosaedro
    </button>
```

Figura 5.35 *- Implementación navegación en HTML*

En el método, se hará uso del NavController indicando la siguiente página a mostrar junto con los parámetros que detallan la figura seleccionada, indicando su identificador único como valor.

```
goIcosahedron(){
  this.navCtrl.push(SettingsPage, {'geometry': ICOSAEDRO })
}
```

Figura 5.36 *- Implementación navegación con NavController*

Después, en la página redirigida 'SettingsPage', se obtiene el parámetro pasado a través de NavParams, indicando la clave que se ha indicado como nombre. Así obtiene el tipo de figura que se ha elegido.

```
this.typeGeometry = navParams.get('geometry')
```

Figura 5.37 *- Implementación navegación con NavParams*

Esta ha sido una muestra de cómo se han implementado todas las navegaciones entre páginas de la aplicación, cada una de ellas con los parámetros necesarios.

5.5.4. Creación de la base de datos

En el apartado de diseño se ha descrito la capa de persistencia de esta aplicación, la cual necesita almacenar las geometrías a dibujar en el espacio tridimensional, y sus atributos. Todo ello se debe gestionar en una capa independiente de la lógica y de la interfaz gráfica, de modo que permita su modificación en un futuro sin la necesidad de cambios en toda la aplicación. Esta capa va a estar compuesta por un una base de datos para almacenar la información de las figuras geométricas de forma segura y rápidamente accesible, además, constará de una interfaz para representar los objetos que vamos a utilizar con sus atributos, descritos en el apartado de diseño, más una parte lógica para realizar las operaciones con relación a la gestión de la base de datos.

Entre las opciones que proporciona Ionic para el almacenamiento de datos de la aplicación, la más común es Storage[61]. Esta opción almacena pares de clave-valor y objectos JSON. La herramienta elige la mejor de sus implementaciones de almacenamiento dependiendo de la plataforma destino. En el entorno de aplicaciones nativas prioriza la base de datos SQLite frente a otras de sus implementaciones como pueden ser IndexedDB, WebSQL o la memoria interna de la aplicación. Estas últimas las elige cuando se ejecuta en web. Lo primero de todo, es instalar el plugin de SQLite con Cordova e instalar el paquete que proporciona Ionic de Storage con npm con los siguientes comandos.

[61] https://ionicframework.com/docs/angular/storage

$ ionic cordova plugin add cordova-sqlite-storage

$npm install --save @ionic/storage

Se procede a incluir este paquete en la aplicación para autorizar su uso, accediendo al fichero de 'app.module.ts' e importando el paquete junto a los demás con la línea de la Figura 5.38.

```
import { IonicStorageModule } from '@ionic/storage';
```

Figura 5.38 *- Importación de Storage*

Ahora ya es posible almacenar información en la base de datos. En la actual aplicación se van a recopilar los datos en un fichero de formato JSON, con los campos definidos en el apartado de diseño, creando un objeto por cada geometría definida. Debido a la cantidad de vértices que se describen para cada figura, esta opción de representación se considera adecuada.

Una vez se tienen los datos y la base de datos está lista, se procede a desarrollar la lógica que acceda a toda esta información. Con este fin se va a hacer uso de los Providers[62] que proporciona Angular. Este elemento se trata de un servicio que provee de su funcionalidad propia, y con la ayuda de la inyección de dependencias de Angular, puede inyectarse dinámicamente en otras partes de la aplicación. Así se abstraen tareas como preguntar datos a un servidor, manipular datos, realizar operaciones complicadas y consultar una base de datos, como en este caso. Se facilita su mantenimiento y se puede reutilizar en varias partes de la aplicación. Para crearlo fácilmente se va a usar Ionic CLI y se escribe el siguiente comando.

$ionic g provider geometry

Con este comando se genera el fichero preparado para implementar el Provider y automáticamente lo importa en 'app.module.ts' para poder utilizarlo en cualquier parte de la aplicación. A continuación se procede a su implementación, tal como se muestra en el Figura 5.39.

[62] https://angular.io/guide/providers

```
initDataDB(){
  this.getDataFromJson()
    .subscribe((res) => {

      this.geometryList = res;
      if (this.geometryList != null){
        res.forEach( g => {
          this.storage.set( String(g.id), g);
        })
      }
    })
}

getGeometry( id: string ){
  return this.storage.get(id)
}

getDataFromJson(): Observable<IGeometry[]> {
  return this.http.get<IGeometry[]>(this.dataUrl)
}
```

Figura 5.39 *- Implementación del Provider*

Una vez implementado este Provider, consta de tres métodos para gestionar la capa de persistencia y permitir a la aplicación hacer uso de los datos requeridos. Los métodos son los siguientes:

- *initDataDB()*: Inicializa la base de datos haciendo uso de Storage. Carga la información del archivo 'data.json' y establece como clave para su identificación el campo 'id' que tiene cada figura.
- *getGeometry(id)*: Obtiene la geometría cuyo campo 'id' coincida con el parámetro pasado. Devuelve el objeto JSON que describe a la figura pedida, con todos los datos que la describen.
- *getDataFromJson()*: Obtiene la información contenida en el JSON a través de la dirección que indica dónde se encuentra este archivo dentro del proyecto.

Para abstraer esta capa lo máximo posible se ha implementado la interfaz IGeometry con los atributos base para la descripción de una figura geométrica. Esta interfaz permite acceder homogéneamente a la información de cada figura. Aunque cambie el modo en que se obtienen los datos, no hace falta cambiar la parte implementada en la capa lógica de la aplicación, obteniendo la información gracias a la IGeometry, mostrada en la Figura 5.40.

```
export interface IGeometry {
    id: number;
    name: String;
    vertices: Array<number>;
    faces: Array<number>;
    minRange: number;
    maxRange: number;

}
```

Figura 5.40 *- Implementación interfaz IGeometry*

Los datos que se representan en esta interfaz son los especificados en el apartado de diseño para la capa de persistencia, necesarios para la representación de las figuras geométricas en el espacio 3D.

5.5.5. Figuras geométricas

Las figuras geométricas que se presentan en la aplicación como opciones para resolver el puzle son descritas a continuación. Cada una de ellas ha sido implementada en base a los datos de sus vértices, calculados en su máximo tamaño dentro del espacio de color HSL representado, y el cálculo posterior de sus caras. Toda su información se ha añadido al archivo JSON comentado anteriormente.

Tabla 5.1 *- Información Icosaedro*

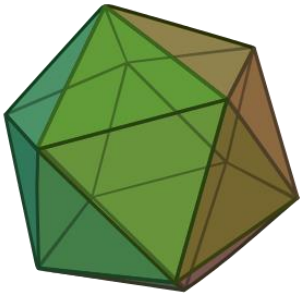
	**Nombre**	Icosaedro
	Familia	Sólidos platónicos
	Caras	20 triángulos equiláteros
	Vértices	12
	Aristas	30

Tabla 5.2 *- Información Triaquistetraedro*

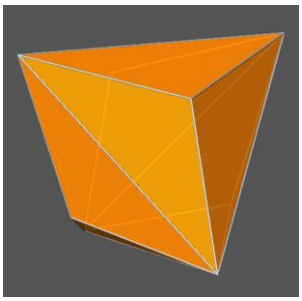
	**Nombre**	Triaquistetraedro
	Familia	Sólidos de catalán
	Caras	12 triángulos isósceles
	Vértices	8
	Aristas	18

Tabla 5.3 *- Información Tetraquishexaedro*

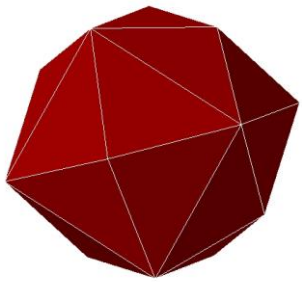
	**Nombre**	Tetraquishexaedro
	Familia	Sólidos de catalán
	Caras	24 triángulos isósceles
	Vértices	14
	Aristas	36

Tabla 5.4 *- Información Hexaquisoctaedro*

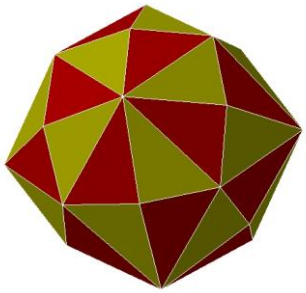
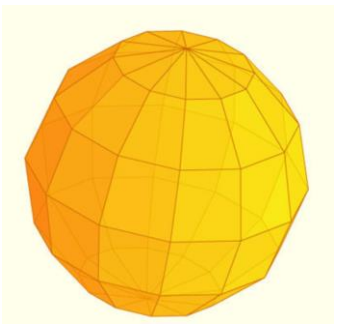
	**Nombre**	Hexaquisoctaedro
	Familia	Sólidos de catalán
	Caras	48 triángulos escalenos
	Vértices	26
	Aristas	72

Tabla 5.5 *- Información Septuaginta*

	**Nombre**	Septuaginta
	Familia	Poliedros circulares
	Caras	72

5.5.6. Configuración de la orientación

En el inicio de la implementación, se comenzó a probar la aplicación ejecutándola en el navegador web del ordenador, haciendo uso de Google Chrome y sus herramientas para desarrolladores que facilitaban la resolución de errores. No se tenía experiencia con ninguna herramienta de las aplicadas y era necesaria una primera fase de aprendizaje y pruebas para conocer si los requisitos se podían cumplir.

Seguidamente, cuando la aplicación era estable y funcionaba correctamente, se comprobó su ejecución en el móvil. Cómo se había diseñado a partir del navegador web del ordenador, los gráficos no se acoplaban correctamente a la pantalla del móvil. Por ello, se decidió establecer una orientación específica para cada pantalla de manera que se visualicen todos los elementos necesarios y sea fácil su uso para los alumnos.

La configuración de la orientación en cada página se modificará con el plugin de Cordova Screen Orientation[63] e Ionic Native. Su instalación se realiza a través de los siguientes comandos.

$ ionic cordova plugin add cordova-plugin-screen-orientation

$ npm install --save @ionic-native/screen-orientation

Después, se importa en la página en la que se quiera hacer uso de este *plugin* tal y como se muestra en la Figura 5.41.

```
import { ScreenOrientation } from '@ionic-native/screen-orientation';
```

Figura 5.41 *- Importación ScreenOrientation*

La orientación de la pantalla se modifica a través de dos métodos de los que dispone este *plugin*. El método 'lock()', al cual se le pasa un parámetro indicando la orientación que se quiere establecer para la página y el método 'unlock()', que permite al usuario volver a rotar la pantalla si lo desea quitando la orientación que se haya establecido previamente.

5.6. Despliegue de la aplicación

Completado el desarrollo de la aplicación, su despliegue consiste en preparar el sistema software para su compilación y ejecución destinada al usuario final. En este trabajo se ha desarrollado una aplicación ejecutable en tres plataformas. A continuación, se explica cómo se producen las distintas versiones finales y el procedimiento para su ejecución.

Las distintas versiones se han subido al repositorio GitHub para hacerlas accesibles tanto a los alumnos como a los lectores de este trabajo que quieran usar la aplicación desarrollada. El enlace para su descarga es el siguiente.

https://github.com/iirenece/ColorDoku3D

[63] https://ionicframework.com/docs/native/screen-orientation/

5.6.1. Despliegue en Android

El *framework* Cordova es fundamental para el despliegue de la aplicación en plataformas móviles. En este caso se va a usar para generar la versión final de la aplicación ejecutable en la plataforma Android.

Lo primero que se debe hacer es preparar el entorno con las herramientas necesarias. Se deben instalar el JDK de Java[64] y el SDK de Android. También existe la opción de instalar el entorno de desarrollo de Android Studio[65], el cual facilita la instalación de estas herramientas, tal y como se ha hecho en este trabajo. Una vez instalado, se ha de configurar las variables de entorno para añadir la localización del SDK en nuestro equipo. Después hay que añadir la plataforma Android a Cordova para preparar el proyecto en la plataforma nativa.

$ ionic cordova platform add android

Finalmente, se ejecuta el comando *build* que construye el ejecutable final con extensión '.apk' para instalar la aplicación en un dispositivo con el sistema operativo de Android. Se puede utilizar la opción *debug* para construir un ejecutable para probar la aplicación en el móvil. Tal y como se ha hecho en este trabajo para las pruebas en el dispositivo Android.

$ ionic cordova build android --debug

Si se quiere construir la versión final para subir a la tienda de Play Store de Android, se deberá firmar primero con la herramienta que proporciona el SDK de Android y después ejecutar el comando con la opción *release.*

$ ionic cordova build android --release

Este comando dará como resultado un archivo con formato '.apk' firmado y listo para subir a la tienda de aplicaciones Play Store.

En este caso se ha construido el archivo sin firma para poder instalarlo independientemente. Para ejecutar esta versión en su dispositivo deberá descargar la opción de Aplicación Android en el repositorio mencionado, configurar su dispositivo a través de ajustes para aceptar la instalación de aplicaciones de fuentes externas, y ejecutar el archivo '.apk', aceptando la instalación de ColorDoku3D en el dispositivo.

5.6.2. Despliegue en web

Para ejecutar la aplicación en el navegador web del ordenador, es necesario un solo comando que genera la construcción de la página web a ejecutar en la carpeta 'www', contenida en el proyecto.

$ ionic build --prod

[64] https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
[65] https://developer.android.com/studio?hl=es

Si se quiere ejecutar la aplicación en un sitio web se debe cargar la carpeta ‘www’ en un servidor para que lo ejecute y sea accesible la aplicación web.

En este caso, se ha cargado la carpeta en cuestión en un repositorio para poder descargarla y ejecutar la aplicación en local. Si quiere ejecutar el juego en su navegador web, descargue la versión para web del repositorio. Extraiga el contenido del archivo comprimido y dentro de la carpeta ‘www’ busque el archivo ‘index.html’ y ejecútelo.

6. Resultados

Finalizado el desarrollo de la aplicación, se procede a exponer los resultados conseguidos. Se mostrarán las capturas de pantalla de la aplicación ejemplificando su uso por un alumno ficticio, el cual pretende practicar su percepción del color resolviendo un puzle. Por defecto, este alumno tiene conocimientos sobre el espacio de color HSL y conoce qué tipo de relaciones establece. Las capturas están realizadas desde un dispositivo Huawei P20, con sistema operativo Android 10.

Lo primero que encuentra el alumno al iniciar la aplicación es una pantalla con el listado de figuras geométricas disponibles para el juego, identificadas por su nombre.

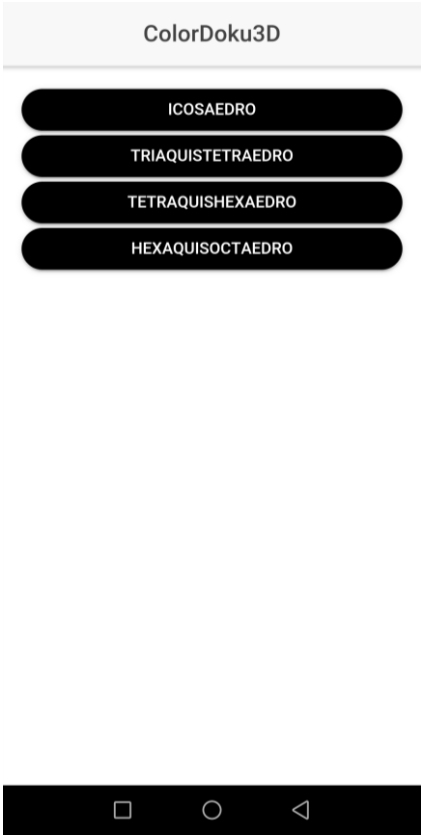


Figura 6.1 *- Captura selección figura*

El alumno decide probar con la primera opción, presionando el botón de ‘Icosaedro’. Seguidamente la aplicación navega a la pantalla de configuración del juego. La figura geométrica se muestra en su máximo tamaño dentro del espacio de color y el alumno la mueve arrastrando el dedo por la pantalla para visualizarla.

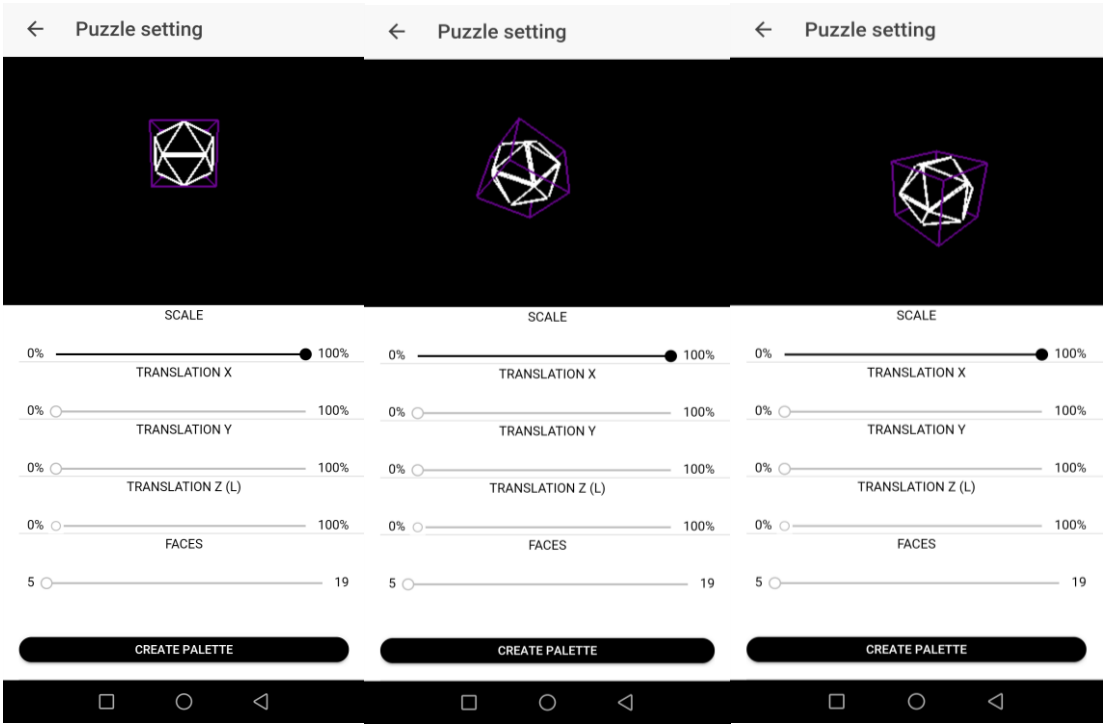


Figura 6.2 *- Capturas movimiento espacio de color*

Lo próximo que quiere realizar el alumno es configurar el puzle a resolver, modificando la geometría dentro del espacio de color, y comienza a probar con los controles mostrados en la parte inferior de la pantalla. El alumno quiere un puzle de nivel medio, donde los colores sean medianamente parecidos, por lo que decide escalar la figura a un tamaño medio, ya que cuando mayor sea la figura los colores que adoptarán sus caras serán más distintos en su tonalidad. Del mismo modo, cuando más pequeña sea la geometría dentro del espacio de color, existirá más relación entre los colores resultantes. El alumno arrastra el *slider* de escalado hacia el valor 57%, como se aprecia en la Figura 6.3.

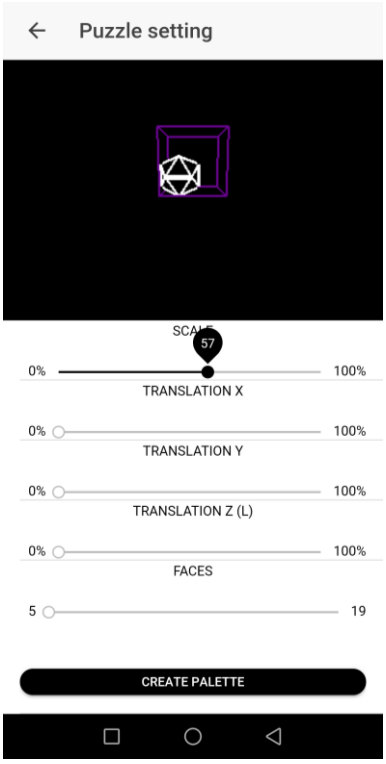


Figura 6.3 *- Captura escalado figura*

Después, decide cambiar la posición de la figura geométrica cambiando sus posiciones sobre el eje X y el eje Y. El espacio de color no muestra cómo los tonos se reparten por su espacio, pero como el alumno conoce el espacio de color HSL, sabe que hacia los extremos los colores resultantes seguirán la tónica de un tono en concreto, mientras si se posiciona por el centro del espacio de color, las caras estarán asociadas a tonos con mayor distinción. El alumno arrastra el control del eje X hasta el valor 52%, después mueve el espacio de color y arrastra el control del eje Y hasta el valor 39%, y vuelve a comprobar que la figura está en medio del espacio de color.

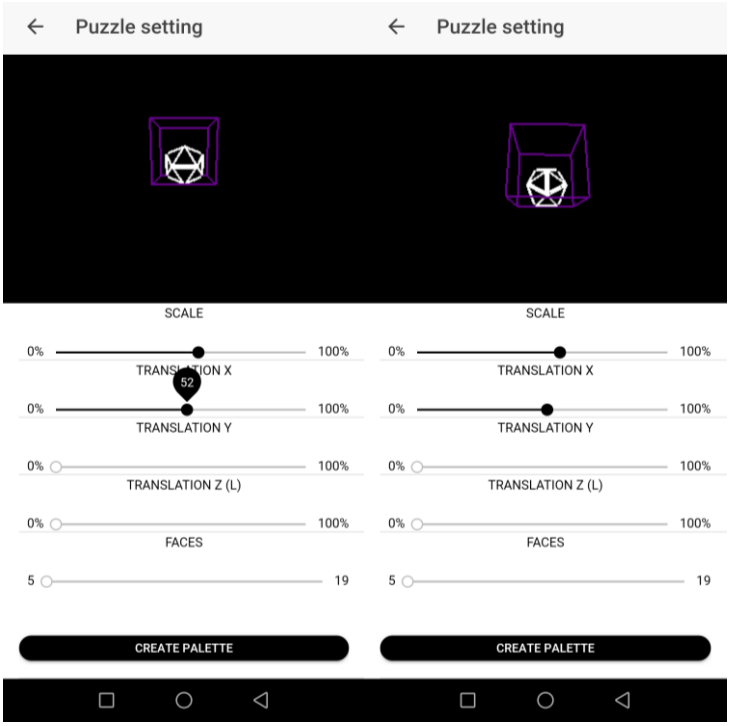


Figura 6.4 *- Capturas desplazamiento de figura en el eje X*

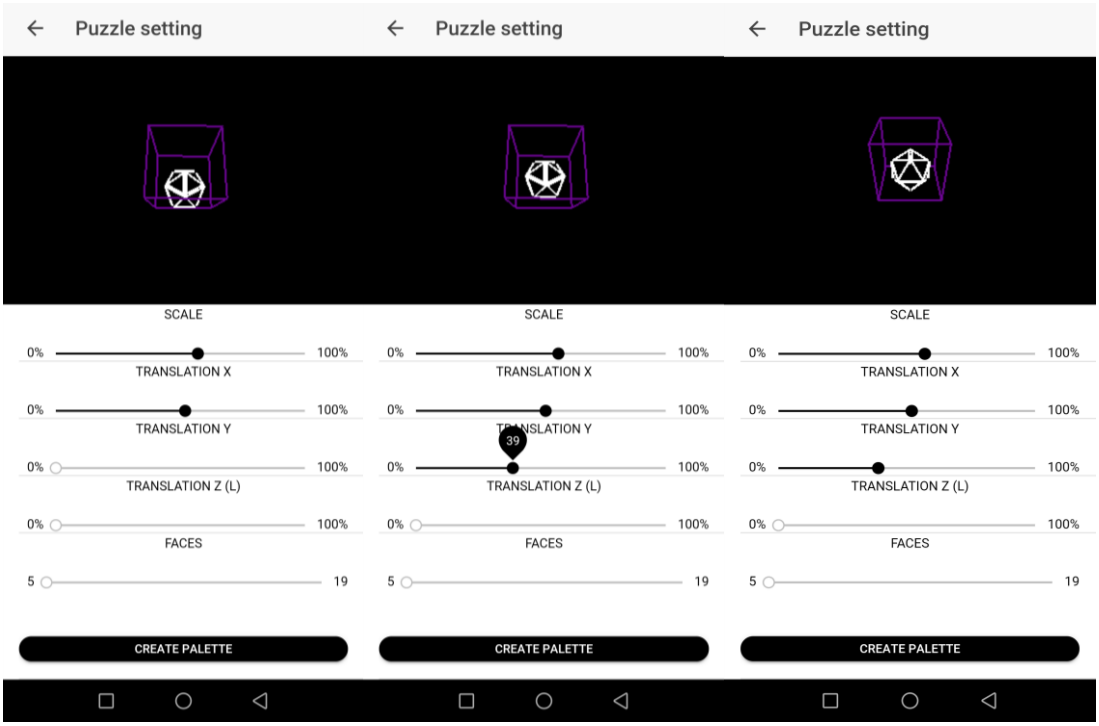


Figura 6.5 *- Capturas desplazamiento de figura en el eje Y*

Seguidamente, modifica el valor del control del eje Z, el cual está relacionado directamente con la característica L del espacio de color HSL (*Lightness* o Luminosidad). El alumno prefiere jugar con colores más claros por lo que arrastra el control del eje Z hasta el valor 56%.

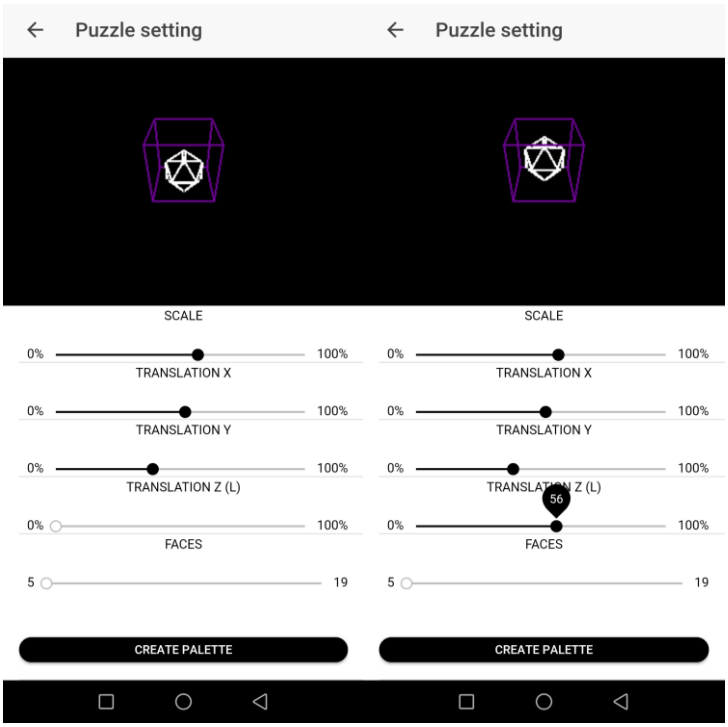


Figura 6.6 *- Capturas desplazamiento de figura en el eje Y*

Por último, el alumno decide elegir una dificultad baja en la resolución del puzle y selecciona 14 caras en el último *slider* que se muestra en pantalla, quedando 6 caras por completar. Hace las últimas comprobaciones finales arrastrando el espacio de color y comprobando que la figura está ocupando el centro cómo él quería.

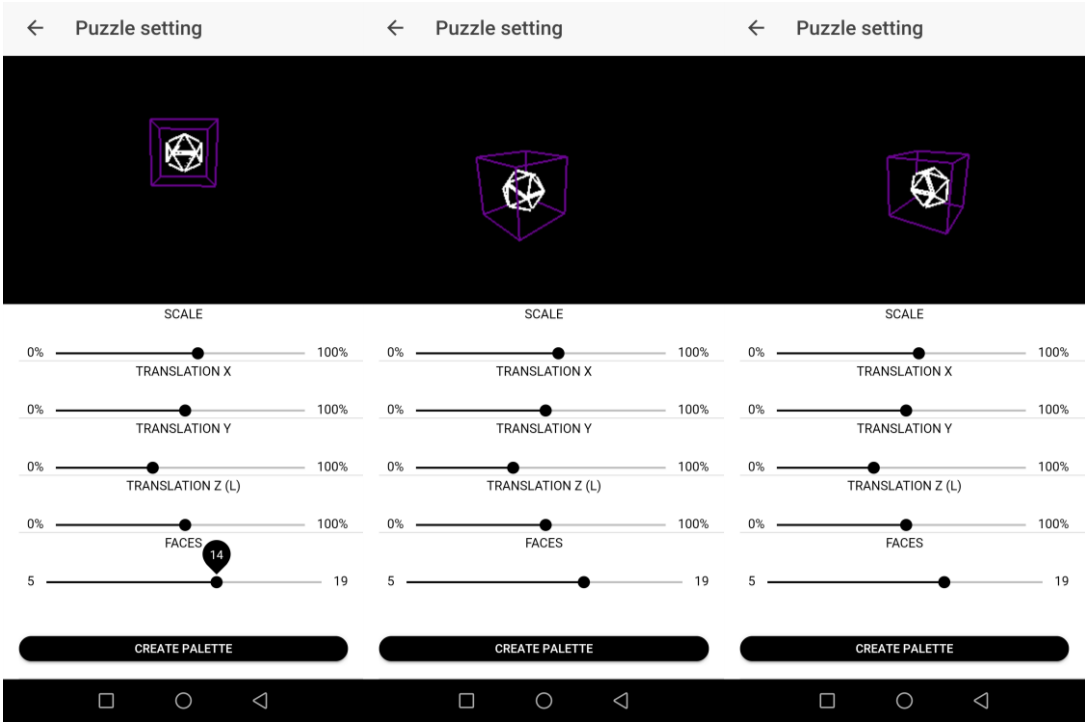


Figura 6.7 *- Capturas selección caras predefinidas*

Seguidamente, presiona el botón de ‘Create palette’. La aplicación navega a la pantalla donde se muestra la paleta de colores que se ha configurado como puzle. El alumno ve 20 colores que se corresponden con las 20 caras del icosaedro. Comprueba que los colores resultantes tienen distintos tonos entre sí y se pueden distinguir con facilidad en su mayoría y le parecen bien para jugar con ellos. Seguidamente presiona el botón de ‘Play’.

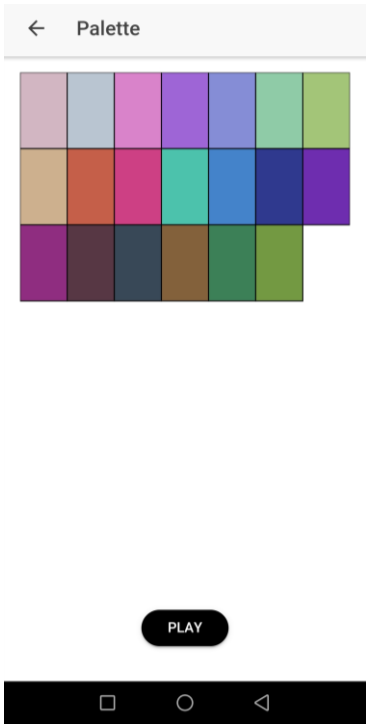


Figura 6.8 *- Captura pantalla para visualizar la paleta de colores*

Automáticamente, la aplicación cambia su orientación para mostrar la pantalla de juego en horizontal. De esta manera se pueden acceder a todos los elementos que participan en la resolución del puzle en una sola pantalla. Además, resulta muy intuitivo controlar la paleta de colores con el pulgar izquierdo de la mano, mientras se manipula la figura con el pulgar derecho, rotándola y seleccionando sus caras.

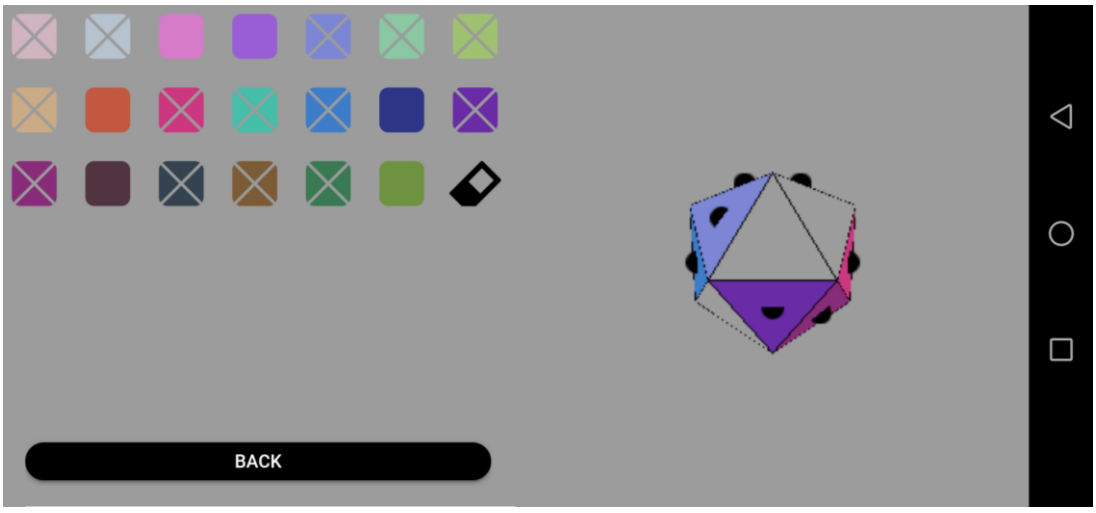


Figura 6.9 *- Captura pantalla juego*

En la Figura 6.9 se puede observar cómo hay catorce colores de la paleta no disponibles para el puzle, ya que se ha configurado anteriormente en la pantalla de configuración catorce caras predefinidas para iniciar el juego. Estos colores están pintados ya sobre la figura, y sus caras están marcadas por un pequeño círculo, indicando que esas caras son las predefinidas del puzle.

El alumno comienza a manipular la figura, moviéndola y haciendo *zoom* en ella para redimensionarla. Después presiona un color, el cual se muestra bordeado de blanco indicando que es el color activo, y selecciona la cara de la figura en la que cree que ese color es el correcto. El color activo se tacha, indicando que ya no está disponible para colorear otra cara, como se en la Figura 6.11.

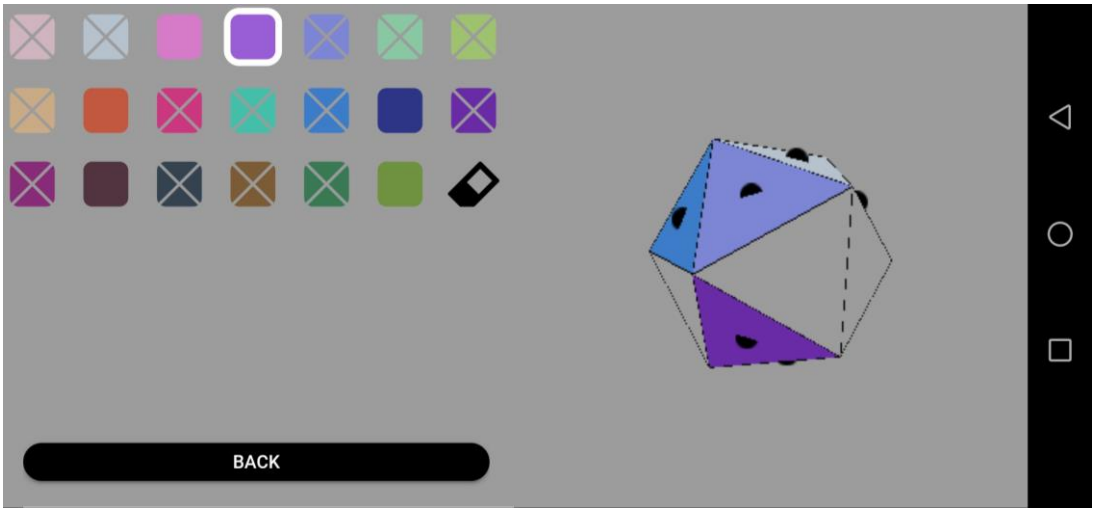


Figura 6.10 *- Captura pantalla juego color activo*

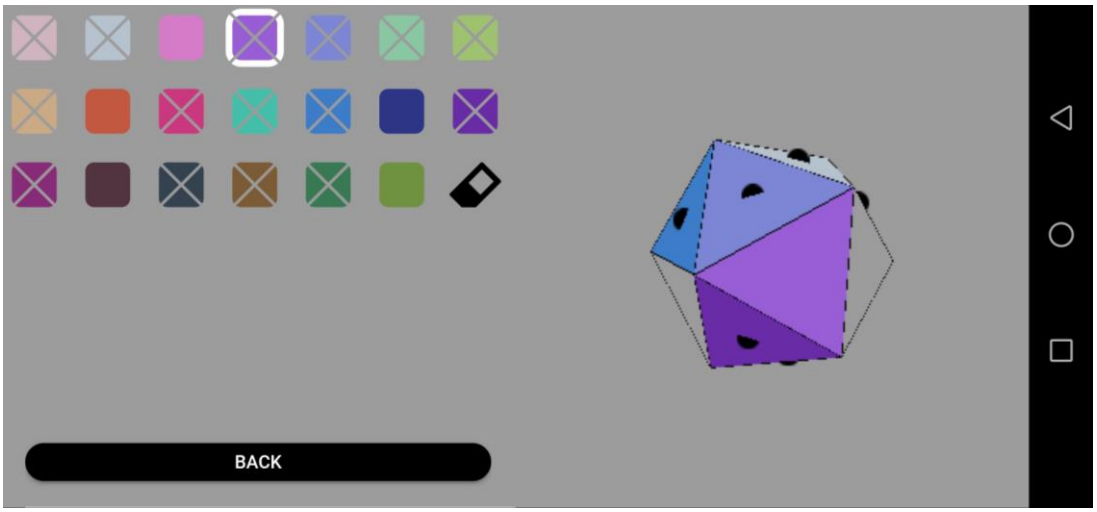


Figura 6.11 *- Captura pantalla juego color pintado*

El alumno va repitiendo este proceso con cada color, pintándolo en la cara que cree que corresponde, hasta que termina el puzle. Para ello, todas las caras deberán estar pintadas del color correcto, y el juego avisará de que se ha completado el puzle.

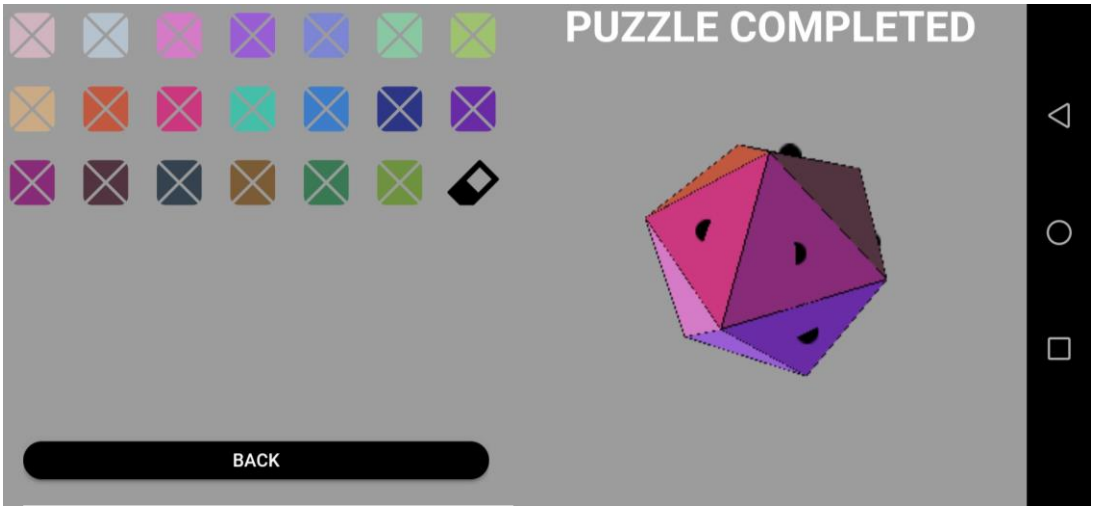


Figura 6.12 *- Captura pantalla juego completado*

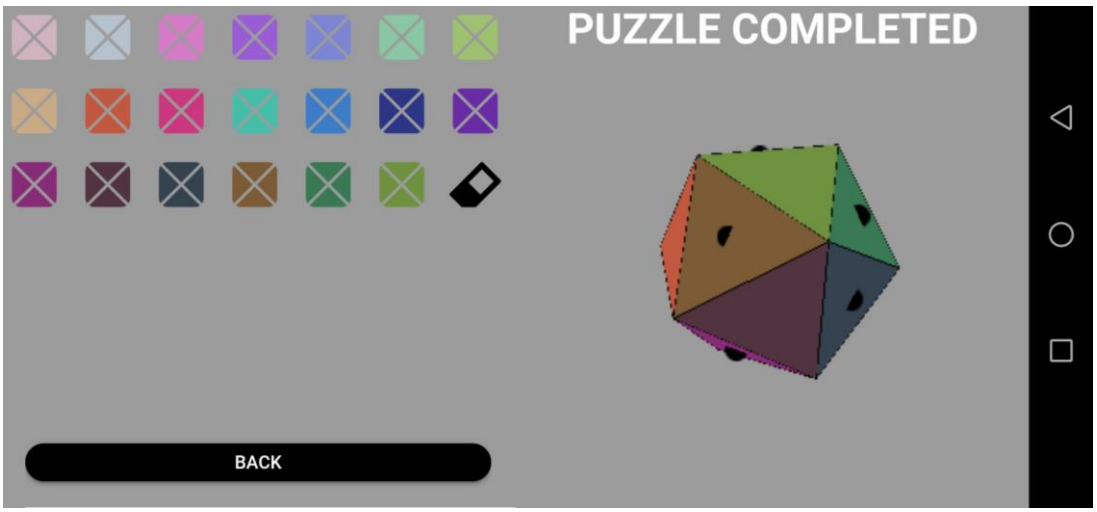


Figura 6.13 *- Captura pantalla juego completado*

Se debe seguir el mismo procedimiento sea cual sea la figura seleccionada. En el caso del Triaquistetraedro, es el puzles con menos caras y colores a resolver. Aún así, presenta un buen nivel de desafío al alumno, debido a la inclinación que tienen sus caras respecto a sus colindates, lo cual afecta al color asociado en el espacio de color HSL. A continuación, se muestran en la Figura 6.14 y 6.15 la configuración de un puzle con tonalidades más parecidas debido al tamaño más pequeño de la figura y su posición en un extremo del espacio de color y no en su centro. Además se puede comprobar que aparecen colores más oscuros debido al pequeño desplazamiento en el eje Z.

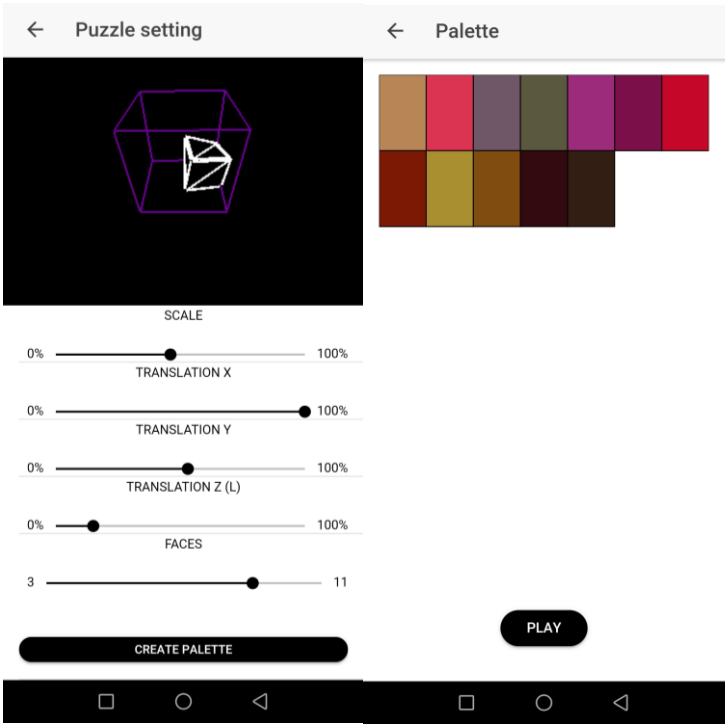


Figura 6.14 *- Capturas Triaquistetraedro*

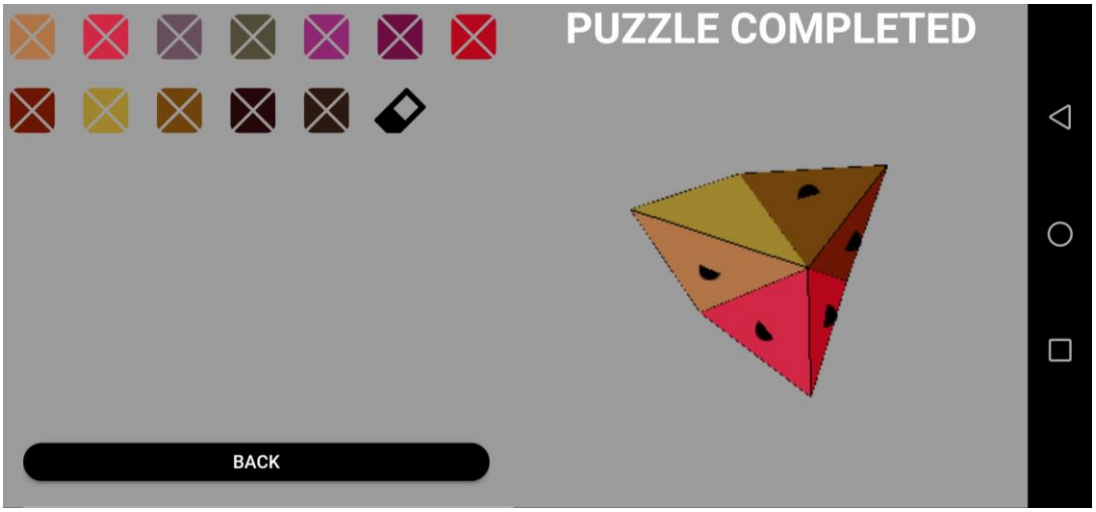


Figura 6.15 *- Captura juego Triaquistetraedro*

El alumno realiza otra configuración con el tetraquishexaedro, el cual no tiene las caras inclinadas como el triaquistetraedro y eso facilita su resolución, pese a su mayor número de caras y por lo tanto de colores a resolver. En este caso también se sitúa la figura en un extremo dentro del espacio de color, por el desplazamiento del eje Y, por lo que predomina la tonalidad verde en los colores.

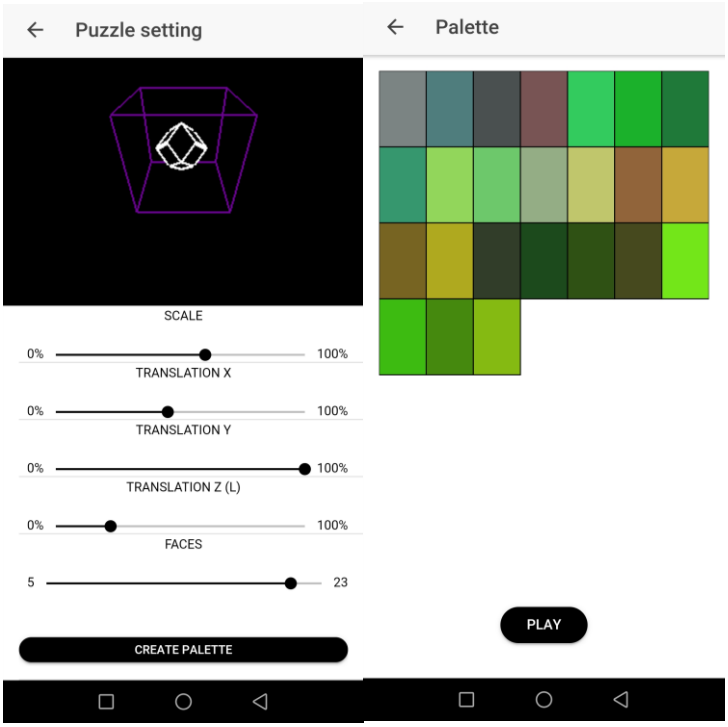


Figura 6.16 *- Captura Tetraquishexaedro*

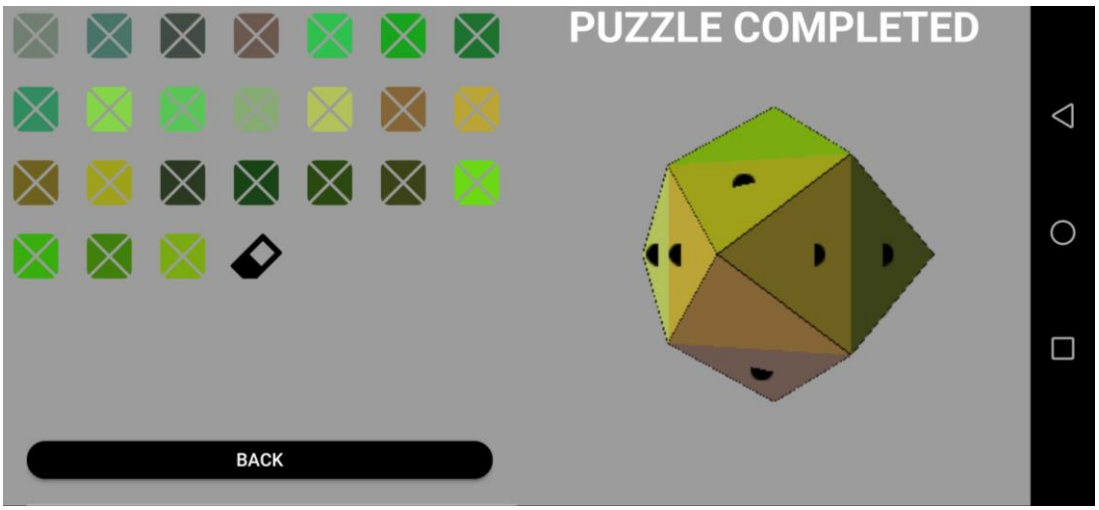


Figura 6.17 *- Captura juego Tetraquishexaedro*

Finalmente, el alumno decide completar un puzle en la figura más compleja de este juego, el hexaquisoctaedro. Con un total de 48 caras, su resolución puede parecer también más compleja pero realmente es más intuitiva, debido a la alta relación entre sus caras y, por lo tanto, la alta relación entre los colores asociados en el espacio de color. Su complejidad reside en el número de incóginitas a resolver, esto quiere decir que cuanto menores sean las caras preestablecidas desde el inicio, más complicada resultará su resolución debido a los parecidos colores de la paleta

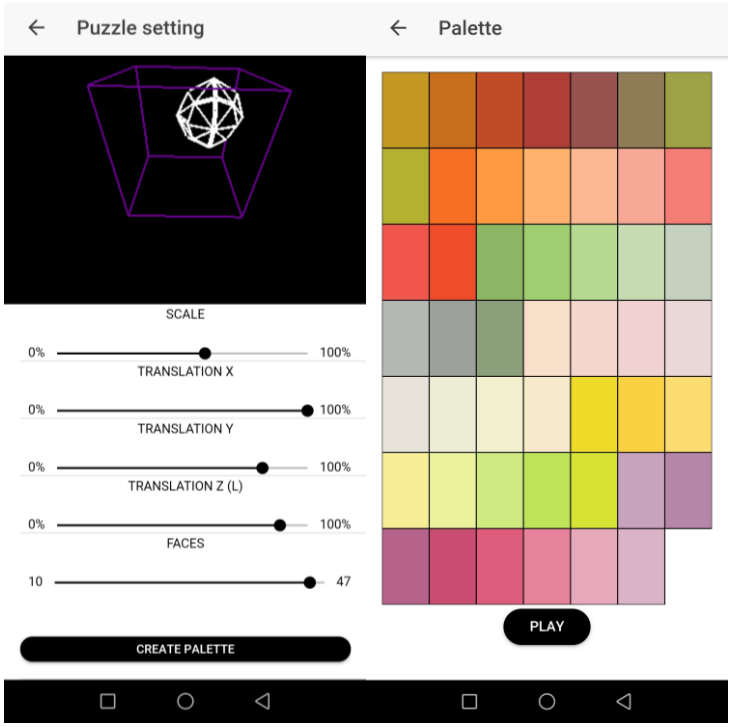


Figura 6.18 *- Captura Hexaquisoctaedro*

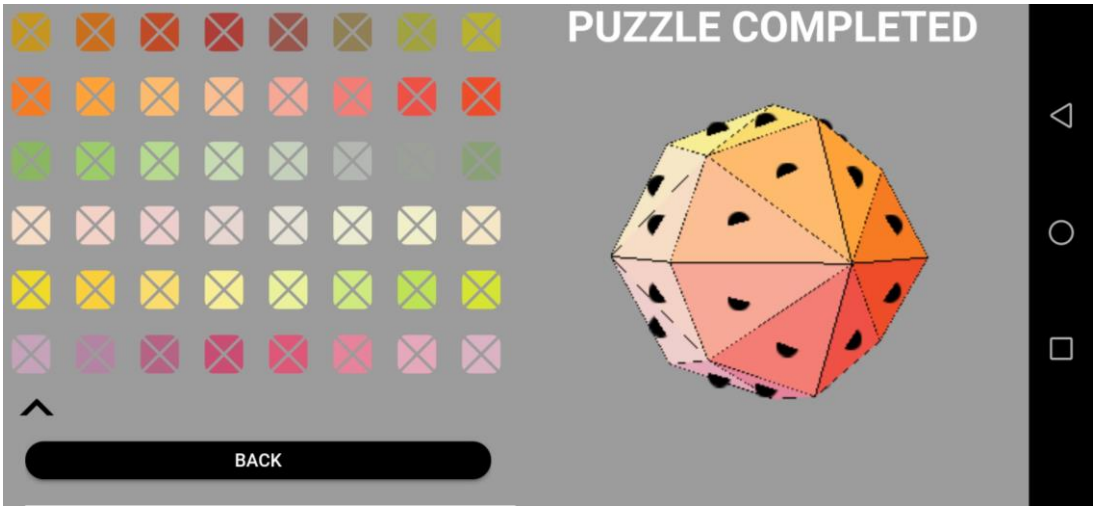


Figura 6.19 *- Captura juego Hexaquisoctaedro*

7. Conclusiones

La realización de este Trabajo Fin de Grado ha resultado muy útil personalmente, debido a las distintas tareas que se han desarrollado durante su ejecución. El conjunto de todas ellas me ha ayudado a adquirir un mayor conocimiento sobre las aplicaciones móviles, ámbito en el que estoy muy interesada desde el inicio de mi experiencia laboral. Particularmente, se ha abordado el campo de las aplicaciones móviles híbridas sobre el cual no tenía conocimientos previos y ha sido productivo poder ampliarlos a través de este proyecto.

Este trabajo comenzó con el desafío inicial de comprender los conceptos relacionados a la idea principal de realizar un puzle de color. Fue primordial estudiar qué era un espacio de color, cómo se pretendía aplicar a este proyecto, cuál es la dinámica habitual en la resolución de puzles de color. En resumen, era necesario entender los elementos que formarían parte del desarrollo del proyecto, ya que para mí no eran evidentes.

Cuando la idea principal estuvo clara, se comenzó a estudiar los tipos de aplicación móvil, defendiéndose la idea de desarrollar una aplicación válida para las plataformas de Android y iOS para que cualquier alumno a los que va dirigida, pudiese ejecutarla en su móvil. La decisión de desarrollar una aplicación móvil híbrida también aportaba la reducción de costes en el desarrollo debido a que no se debían desarrollar dos aplicaciones diferentes para cada plataforma. Junto con esta elección fue respaldada la idea de desarrollar la aplicación con el *framework* de Ionic para emplear las tecnologías web y hacer uso de la librería Three.js, imprescindible para el desarrollo de espacios 3D.

Al desplegar la ida principal junto con los tutores de este trabajo, se comenzó a establecer grupos de requisitos deseados para la aplicación. Progresivamente, se intentaban cumplir estos requerimientos y se detallaban otro conjunto de ellos. La razón de esta metodología ágil de trabajo era principalmente la carencia de experiencia con las herramientas y tecnologías elegidas, de manera que todo el desarrollo ha sido un desafío de aprendizaje de estos recursos y adaptación de los requisitos deseados a las posibilidades que ofrecían.

Se comenzó a especificar los requisitos que debía cumplir la aplicación empezando por la pantalla de juego, ya que es la finalidad de toda la aplicación. Cómo no se tenía experiencia con las herramientas elegidas para el desarrollo, se realizó una investigación previa al desarrollo para obtener un conocimiento básico. Después, se comenzó a desarrollar la pantalla de juego teniendo en cuenta los requisitos detallados y profundizando sobre las capacidades que ofrece la librería Three.js. Los requisitos pensados inicialmente con esta pantalla se han podido cumplir en su totalidad, tanto

los controles para manipular la figura geométrica, como la paleta y su funcionamiento para colorear las caras de la geometría.

Posteriormente, se comenzó a desarrollar el motor del puzle, el cual permite al alumno configurar el tipo de juego que desea llevar a cabo. Los principales requisitos gráficos que se acordaron para esta pantalla estaban referidos al modo de control de la figura y el espacio de color con el uso de *sliders* y fueron abordados exitosamente. Los requisitos funcionales que permitían el uso de los controles de configuración supusieron un gran reto a nivel teórico. La modificación de una geometría respecto otra geometría que la contiene conllevó la elaboración de cálculos estrictamente relacionados con el espacio tridimensional y la geometría como rama de las matemáticas. Conseguir que los controles de escalado y desplazamiento funcionaran como se requería fue la tarea que resultó más compleja y duradera. Una vez conseguida fue muy gratificante comprender los aspectos elementales del espacio tridimensional.

Los siguientes cálculos matemáticos fueron las mediciones de los colores resultantes de la figura geométrica respecto al espacio de color, una vez modificada dentro de éste. Implicaron otro desafío debido a que el espacio de color que se utiliza está adaptado a las necesidades de la aplicación, y en un principio los colores no se calculaban de forma correcta. A través de pruebas, más la ayuda de los tutores de este trabajo, se consiguió calcular de forma correcta los colores de la figura según el espacio de color HSL representado.

Una vez se habían implementado las pantallas más importantes de la aplicación, se comenzó a desarrollar todas las demás. Paralelamente, se introdujeron más figuras geométricas para aumentar las posibilidades que proporciona el juego. Una de estas geometrías, llamada Septuaginta y descrita en la Tabla 5.5, era más compleja que las demás y se intentó introducir dentro de la aplicación mediante un método distinto de carga de objetos de la librería Three.js, pero resultó un proceso muy complicado y finalmente, pese al tiempo empleado para su implementación, se abandonó su incorporación al juego.

Por último, el despliegue de la aplicación se ha conseguido para la plataforma Android y la ejecución en el navegador web, pero no para iOS. Hubiese sido conveniente conseguir desplegar la aplicación a esta plataforma y subir todas las versiones a los correspondientes mercados de aplicaciones para facilitar su acceso a los alumnos.

7.1. Cumplimiento de objetivos

Se van a revisar los objetivos propuestos al inicio de este trabajo para valorar si el resultado cumple la idea propuesta.

En el estado del arte se han descrito los distintos tipos de aplicaciones móviles y sus características, lo cual ha implicado una investigación sobre ellos y el consecuente

conocimiento sobre su desarrollo. Ha ocurrido lo mismo con las herramientas más popularizadas para el desarrollo de aplicaciones móviles híbridas, comprendiendo el estado actual de estas tecnologías y las características que ofrecen cada una de ellas.

Previamente a la implementación, se ha desarrollado un diseño interno de la aplicación, estructurándola con una arquitectura de tres capas fácilmente conseguida gracias a Ionic. También, se ha especificado el comportamiento externo de la aplicación el cual ha servido de guía durante el desarrollo.

Antes de iniciar la implementación, se han consultado las documentaciones y guías básicas para conocer lo más básico de las tecnologías que se iban a aplicar. Además, se ha estudiado el espacio de color HSL y las transformaciones de geometría en el espacio tridimensional, necesarias para los cálculos dentro del espacio de color.

Finalmente se ha conseguido una aplicación la cual es ejecutable en cualquier dispositivo móvil que consiste en un juego para componer puzles de color. La configuración del puzle ayuda al alumno a conocer qué relaciones especifican el espacio de color HSL, y la posterior resolución del puzle desarrolla su percepción del color porque debe identificar qué relaciones existen entre las caras de la figura representada y con ello, discriminar colores.

7.2. Relación del trabajo con los estudios cursados

El conjunto de asignaturas cursadas a lo largo del Grado de Ingeniería Informática ha servido como base para el desempeño de un trabajo de esta índole. En especial, la rama de Ingeniería de Software estudiada comparte una gran relación con la planificación de un proyecto como el que se presenta en este trabajo.

En primer lugar, las asignaturas dedicadas a la programación cursadas en el primer y segundo curso me ayudaron a comprender en qué consistía el desempeño de esta función dentro de la informática. A pesar de que se basaban en un paradigma orientado a objetos con ayuda del lenguaje Java, el cual no tiene relación con el paradigma aplicado en la programación de esta aplicación, su aprendizaje ayudó a comprender los elementos principales de la programación para ser capaz de aprender cualquier otro lenguaje, como en este caso TypeScript.

La asignatura del tercer curso de Ingeniería del Software ha ayudado a la estructuración de la aplicación con una arquitectura basada en capas.

Por último, la rama de Ingeniería de Software ha aportado en su totalidad un conjunto de buenas prácticas que han facilitado el desarrollo de la aplicación. Asignaturas como Análisis y especificación de requisitos o Proceso de software han sido piezas claves para abordar este trabajo.

7.3. Trabajos futuros

La primera versión de ColorDoku3D producida en este trabajo ha conseguido los objetivos básicos de implementación de este juego. Se ha creado el motor del juego relacionado con el espacio de color HSL, la configuración de una figura geométrica dentro y el posterior cálculo de los colores respectivos a sus caras. Además, está disponible la pantalla de juego con todos los controles necesarios, junto con el sistema de comprobación del puzle.

Aunque ahora la configuración del juego sea manual, sería interesante aprovechar el motor para la creación de puzles preestablecidos con unos valores de configuración concretos. Estos puzles se podrían poner a disposición del usuario ordenados por niveles, para crear un recorrido desde los puzles más básicos hasta los más avanzados y así guiarlos en su aprendizaje. Además, podrían conseguir puntos y añadir un apartado de clasificaciones. En definitiva, se pueden desarrollar estrategias de gamificación para hacer más atractivo el juego.

Por otro lado, este juego estará más enriquecido cuanta más variedad de figuras geométricas tenga a disposición del usuario. En este trabajo se han abordado las figuras con caras triangulares, debido a la sencillez de su implementación. Incorporar figuras como el dodecaedro, cuyas caras son pentágonos, implica crear un sistema de reconocimiento especial del tipo de cara en cuestión, ya que se deben tratar como conjuntos de caras triangulares. Esto afecta tanto en el cálculo del color como en la pantalla del juego a la hora de interpretar la selección de las caras por individual, y cuando se pinta o borra un color sobre ellas.

La implementación de la Septuaginta estaría ligada a este último propósito. La Septuaginta es un poliedro semejante a una esfera. Se pretendía implementar para comprobar cómo los colores de cada cara resultarían muy parecidos entre ellos, y así comprender mejor el espacio de color y su clasificación específica de colores.

Considerar esta figura geométrica en el desarrollo presentó un gran desafío debido al alto número de caras, ya que se tienen que calcular manualmente como se ha mencionado en el desarrollo de la pantalla de juego. Por lo tanto, se probó la opción que aporta Three.js para cargar objetos tridimensionales llamada OBJLoader, la cual acepta archivos con formato ‘.obj’. Fue posible obtener los vértices de la Septuaginta, pero resultó muy complicado obtener sus caras e identificarlas. Era necesario conocer e identificar todas las caras para distinguirlas entre triangulares o no, y así poder pintarlas adecuadamente en el puzle. Además, las caras que no son triangulares y están compuestas por 4 lados, tendrían que ser tratadas como dos caras, es decir, dos triángulos representados cada uno de ellos con la clase Face3 de Three.js, ya que es la única manera de representarlas. Finalmente, no se encontraron alternativas a estos problemas y se optó por no incluir esta figura geométrica en la aplicación. Por lo tanto, sería un objetivo claro para desarrollar en el futuro.

Queda pendiente desplegar la aplicación para la plataforma iOS, ya que está preparada para su despliegue sin mayores dificultades usando Cordova, pero al

carecer del equipo necesario (Mac, XCode, iPhone o iPad) para realizarlo y testearlo, no se ha concluido.

Por último, sería conveniente subir la aplicación a los mercados de aplicaciones correspondientes de cada plataforma móvil. Incluso se podría crear una página web para su uso desde el ordenador. Este fin ayudaría al acceso de los alumnos al juego, y además lo pondría a disposición de otras personas interesadas en él.

8. Bibliografía

[1] Inspire Lab, «Gamification - 1. Introduction,» 18 Septiembre 2013. [En línea]. Available: https://www.youtube.com/watch?v=-N-jMSymzBM. [Último acceso: 25 Septiembre 2020].

[2] Game Learn, «Todo lo que necesitas saber sobre los serious games y el game-based learning, explicado con ejemplos,» [En línea]. Available: https://www.game-learn.com/lo-que-necesitas-saber-serious-games-game-based-learning-ejemplos/. [Último acceso: 16 Octubre 2020].

[3] V. Perez Castillo, «Más propiedades del color: tono, brillo y saturación,» Envero, 6 Abril 2020. [En línea]. Available: https://www.enverodeco.es/blog/propiedades-del-color-tono-brillo-y-saturacion. [Último acceso: 2020 Octubre 16].

[4] J. L. H. Calvet, La edición de los Elementos de Euclides de Oliver Byrne. Londres 1847, p. 424.

[5] ACM, «Software Engineering 2014,» 23 Febrero 2015. [En línea]. Available: https://www.acm.org/binaries/content/assets/education/se2014.pdf. [Último acceso: 19 Octubre 2020].

[6] Antevenio, «Antevenio,» 24 Febrero 2020. [En línea]. Available: https://www.antevenio.com/blog/2020/02/que-es-la-metodologia-scrum/. [Último acceso: 14 Agosto 2020].

[7] H. K. Flora, X. Wang and S. V. Chande, "An Investigation on the Characteristics of Mobile Applications: A Survey Study," *International Journal of Information Technology and Computer Science,* pp. 21-27, 2014.

[8] M. Juste, «Las cifras de Internet: En España el 85% de la población está conectada,» 2018.

[9] P. LePage, «Developers Code Labs,» Google, [En línea]. Available: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp?hl=es. [Último acceso: Septiembre 2020].

[10] G. Hartmann, G. Stead y A. DeGani, «Cross-platform mobile development,» Tribal Group, 2011.

[11] A. Beshir, *Cross-platform development with React Native,* 2016.

[12] S. Yusuf, de *Ionic Framework By Example*, Packt, 2016, p. 158.

[13] Mobile Hackers.io, «Medium,» Julio 2018. [En línea]. Available: https://medium.com/@mobilehackersio/top-8-apps-built-with-ionic-framework-downloaded-more-than-10m-times-601cedd75886.

[14] W. Wu, *React Native vs Flutter, cross-platform mobile application,* 2018.

[15] Appstronauts, «Flutter App Examples: 10 Successful Apps Built Using,» 30 Marzo 2020. [En línea]. Available: https://appstronauts.co/blog/flutter-app-examples-10-successful-apps-biult-using-flutter/. [Último acceso: 13 Septiembre 2020].

[16] RJ Code Advance, «Arquitectura en capas – Análisis completo + Tradicional vs Modernas, DDD, DIP (Cap 5),» 25 Junio 2019. [En línea]. Available: https://rjcodeadvance.com/patrones-de-software-arquitectura-en-capas-analisis-completo-ejemplo-ddd-parte-5/. [Último acceso: 1 Octubre 2020].

[17] Khan Academy, «HSL color model,» 25 Abril 2019. [En línea]. Available: https://www.youtube.com/watch?v=Ceur-ARJ4Wc&feature=youtu.be. [Último acceso: 24 Octubre 2020].

[18] J. Fayzullaev, *Native-like Cross-Platform Mobile Development,* 2018.

[19] V. a. Don't Panic MOBILE DEVELOPER'S GUIDE TO THE GALAXY, Open-Xchange, 2019, p. 337.