Requisitos técnicos y funcionales. Características implementadas.

Amadeo Delgado Casado g4.3 Vdju

Requisitos funcionales

La herramienta debe:

- Permitir seleccionar desde unity un color.
- Iniciarse con el último color sobre el que se trabajo, si existe.
- Las operaciones se almacenan en un .txt que el usuario puede leer.
- Devolver el color opuesto al seleccionado.
- Devolver colores análogos al seleccionado.
- Almacenar la funcionalidad en una dll.
- Utilizar librerías estáticas.

Requisitos técnicos

- Uso de la API de unity.
- Uso de fstream para general los datos e inicializar la herramienta.
- Uso de integers para pasar los datos de unity color32 a c++ integer.
- Dos proyectos de visual en el que la dll depende del lib, en la dll solo se hacen llamadas a funciones definidas en la librería estática.

Características implementadas

Modelo de datos

Al trabajar solo con colores en mi herramienta, los datos están todos contenidos en 4 bytes. Esto me permite pasar fácilmente de color32 en unity a integer en c++ ya que ocupan el mismo espacio, además cree un struct ColorRGBA en c++ para manejo de los colores más cómodamente.

Todo comienza en unity donde al pulsar un botón de la herramienta se llama a Interface.cs, esta interfaz pasa los colores a la dll en integer mediante operaciones de bits. Se divide en:

R:0000 0000 - B: 0000 0000- G: 0000 0000- A: 0000 0000

Después ya en c++ la dll llama a las funciones definidas en la librería estática donde se hace un/pack del color, as operaciones y la serialización a archivos.

Backend (Librerías estática/ dinámica e interfaces)

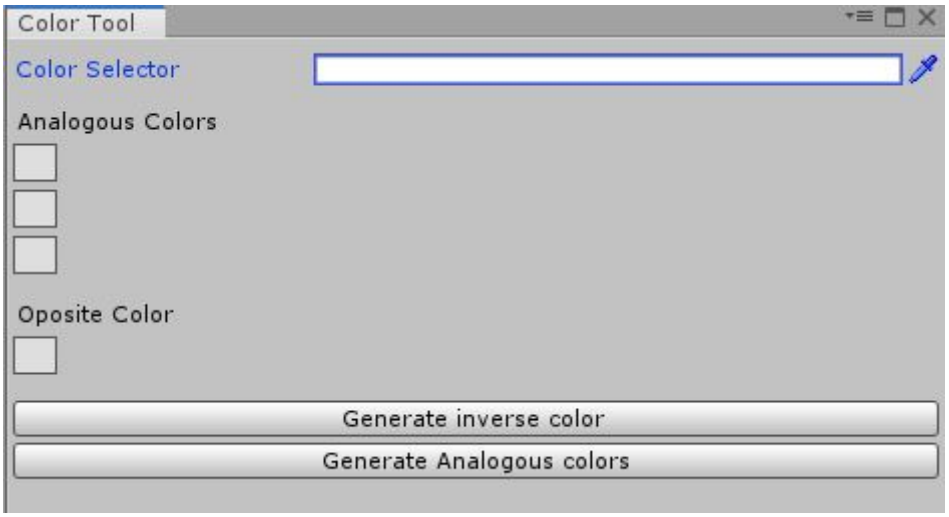
El back end comienza con la clase "*Interface.cs*", esta clase se encarga de transformar los valores de color32 a int y contiene wraps para todas las funciones de la dll. Estas funciones estan definidas en "*ColorToolDLL.dll*", que actúa como otra interfaz entre la librería estática donde se llevan a cabo las operaciones y la interfaz "*Interface.cs*" de unity.

ColorToolDLL.dll incluye la definición de mi api "AMAPI" para exportar la dll y la librería estática, incluyendo ApiDefine.hpp y la cabecera ColorTool.hpp. Dentro de esta librería estática se encuentran las funciones de serialización que utilizan la <fstream> e <iostream> para la serialización.

A la hora de editar las funciones, lo primero que hago es un unpack de los colores para poder trabajar con ellos por separado ya que deben representar de 255-0. Esto lo hago mediante una operación de bits y después guardo el color original para darle persistencia al último color con el que se trabajó en caso de cerrar unity. Después operó con los colores ya sea para sacar el inverso o el análogo y hago pack con los colores guardando los datos , finalmente devuelvo el color.

GUI

Para la interfaz de usuario he utilizado la clase EditorWindow de unity que incluye todas las funciones que necesitaba para botones y selección de colores. Está definido dentro del archivo "*GUI.cs*" y la interfaz es la siguiente:



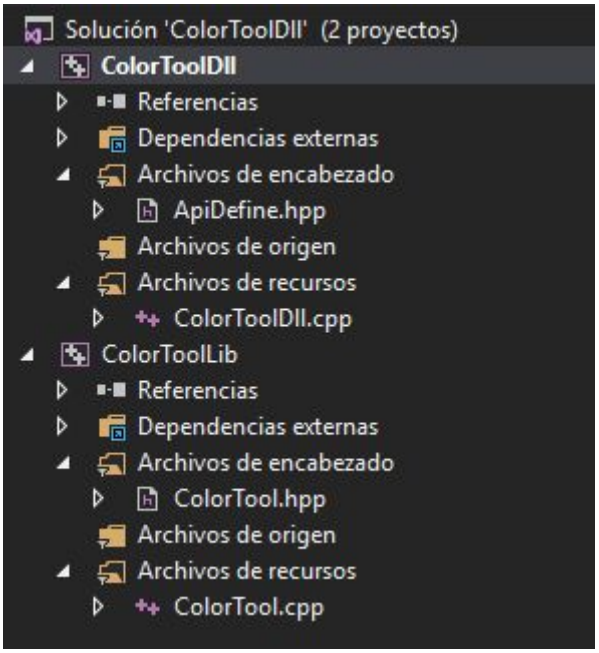
Consta de un selector de color, unos cuadrados que permiten ver lo colores generados, y dos botones: uno para colores opuestos y otro para colores análogos.

Serialización

Utilizo ifstream para leer los datos y ofstream para escribir. Al iniciar la herramiento busco si existe el archivo "*config.txt*", en caso de no existir lo creo y devuelvo color blanco(0xFFFFFFFF). Si existe, utilizo ese archivo para extraer el color y utilizarlo para calcular, esto le da persistencia ya que se guarda y utiliza el último color con el que se trabajó. Además al realizar una operación se almacenan en "*data.txt*" los resultados, dentro de la carpeta de unity pudiéndose ver todas las operaciones con colores realizadas. Se guarda el tipo de operación, el color original y el color modificado respectivamente.

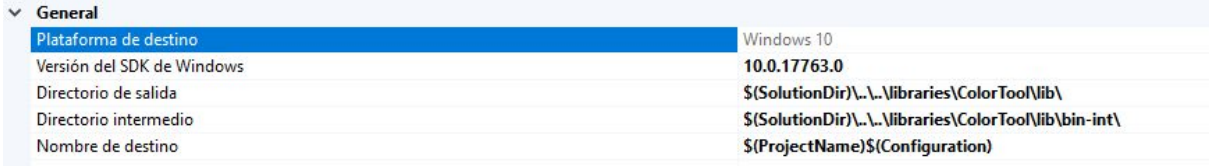
Estructura del proyecto

Estoy muy orgulloso de la estructura del proyecto que he conseguido tener en visual studio.

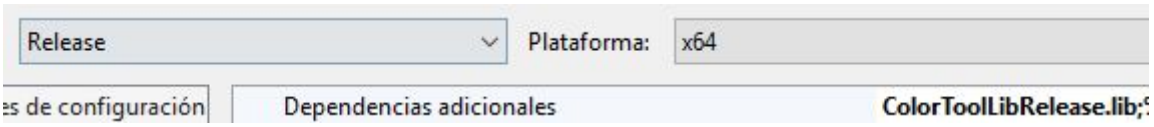


La solución está formada por dos proyectos, uno para la dll y otro para la .lib. La librería estática tiene los header dentro de la carpeta libraries/ColorTool/include, junto a esta carpeta se encuentra otra llamada lib donde se guardan los binarios con el nombre "ColorToolLib$(Configuration).lib". Hize que el el nombre cambiara segun la compilacion para poder hacer que dandole a compilar al la dll en cualquier configuración se compila primero la librería estática y esta genera el archivo que la dll necesita.

Propiedades de el proyecto .lib



Dependencias adicionales de la .dll



Ademas le agregue una referencia a la al proyecto de la dll del proyecto de la librería estática para poder compilar los dos juntos dando solamente al botón de compilar en la ColorToolDll.vs