Introducción al Proyecto

- **Objetivo**: Describir brevemente el objetivo del proyecto y su relevancia. ¿Qué resuelve o mejora el proyecto en términos de problemas del mundo real?

- El proyecto **"Pintura Mágica"** tiene como objetivo principal proporcionar a los niños una herramienta interactiva que fomente el desarrollo de su creatividad y habilidades motoras mediante el dibujo digital. La aplicación busca ofrecer una experiencia intuitiva y amigable, que les permita experimentar con colores, formas y texturas de manera libre, al tiempo que mejora su destreza manual y capacidad de expresión artística.
- **Relevancia:**
  En un mundo cada vez más digitalizado, es esencial que los niños desarrollen competencias tecnológicas desde temprana edad. Este proyecto combina la tecnología con la creatividad, ofreciendo una alternativa a las herramientas tradicionales de dibujo y potenciando habilidades como la coordinación ojo-mano, la percepción del color y la imaginación. Además, fomenta el aprendizaje lúdico, lo que lo convierte en una herramienta educativa y de entretenimiento accesible para todos.
- El uso de **Pintura Mágica** puede ser especialmente valioso en contextos educativos y terapéuticos, ayudando a niños a mejorar su concentración y habilidades motoras mientras se divierten explorando el mundo del arte digital.

- **Descripción del Sistema**: Proporciona una descripción detallada de cómo funcionará la aplicación. Incluye la funcionalidad de la vista (interfaz gráfica) y la lógica del controlador.

  La aplicación **"Pintura Mágica"** es un software diseñado para que los niños puedan dibujar y pintar de manera interactiva utilizando herramientas digitales. Está desarrollada con Python y utiliza las bibliotecas **Tkinter** y **CustomTkinter** para la creación de la interfaz gráfica, además de **Pillow** para el manejo de imágenes.

Funcionamiento General

El sistema está basado en el patrón **Modelo-Vista-Controlador (MVC)**. Su funcionalidad principal se centra en permitir al usuario dibujar, seleccionar colores, cambiar el tamaño del pincel, borrar, cargar imágenes y guardar los dibujos realizados.

Vista (Interfaz Gráfica)

La interfaz gráfica ha sido diseñada para ser intuitiva y amigable, adecuada para niños.

Características principales de la interfaz:

1. **Lienzo de dibujo:**
   - Espacio donde el usuario puede dibujar líneas o formas utilizando el mouse.
   - Soporte para cambio de fondo y borrador.
2. **Barra de herramientas:**
   - Botones y controles para realizar las siguientes acciones:
     - **Seleccionar color:** Permite elegir un color personalizado mediante un selector.
     - **Cambiar tamaño del pincel:** Slider que ajusta dinámicamente el grosor del pincel.
     - **Borrar todo:** Limpia completamente el lienzo.
     - **Modo borrador:** Alterna entre el modo de dibujo y borrador.
     - **Guardar dibujo:** Guarda el contenido del lienzo como una imagen en formato PNG.
     - **Cargar imagen:** Permite cargar una imagen desde el disco al lienzo.
     - **Deshacer y rehacer:** Opciones para gestionar el historial de acciones de dibujo.

Controlador (Lógica del Sistema)

El controlador maneja las interacciones del usuario con la vista y coordina los cambios en el modelo de datos.

Principales funciones del controlador:

1. **Dibujar en el lienzo:**
   - Controla el evento del mouse para dibujar líneas o formas según el color y tamaño del pincel.
   - Si está activo el modo borrador, borra en lugar de dibujar.
2. **Gestión del historial de acciones:**
   - Implementa funcionalidades de deshacer y rehacer, permitiendo a los usuarios corregir errores.
   - Registra cada acción de dibujo en una lista para manejar la lógica del historial.
3. **Guardar y cargar imágenes:**
   - Convierte el contenido del lienzo en una imagen utilizando la biblioteca Pillow.
   - Permite al usuario guardar su dibujo en el disco o cargar imágenes existentes.

4. **Cambio de fondo:**
    - Permite al usuario seleccionar un color de fondo para el lienzo.

Modelo (Datos)

El modelo se encarga de almacenar y gestionar los datos relacionados con las acciones de dibujo y configuración de la herramienta:

Componentes del modelo:

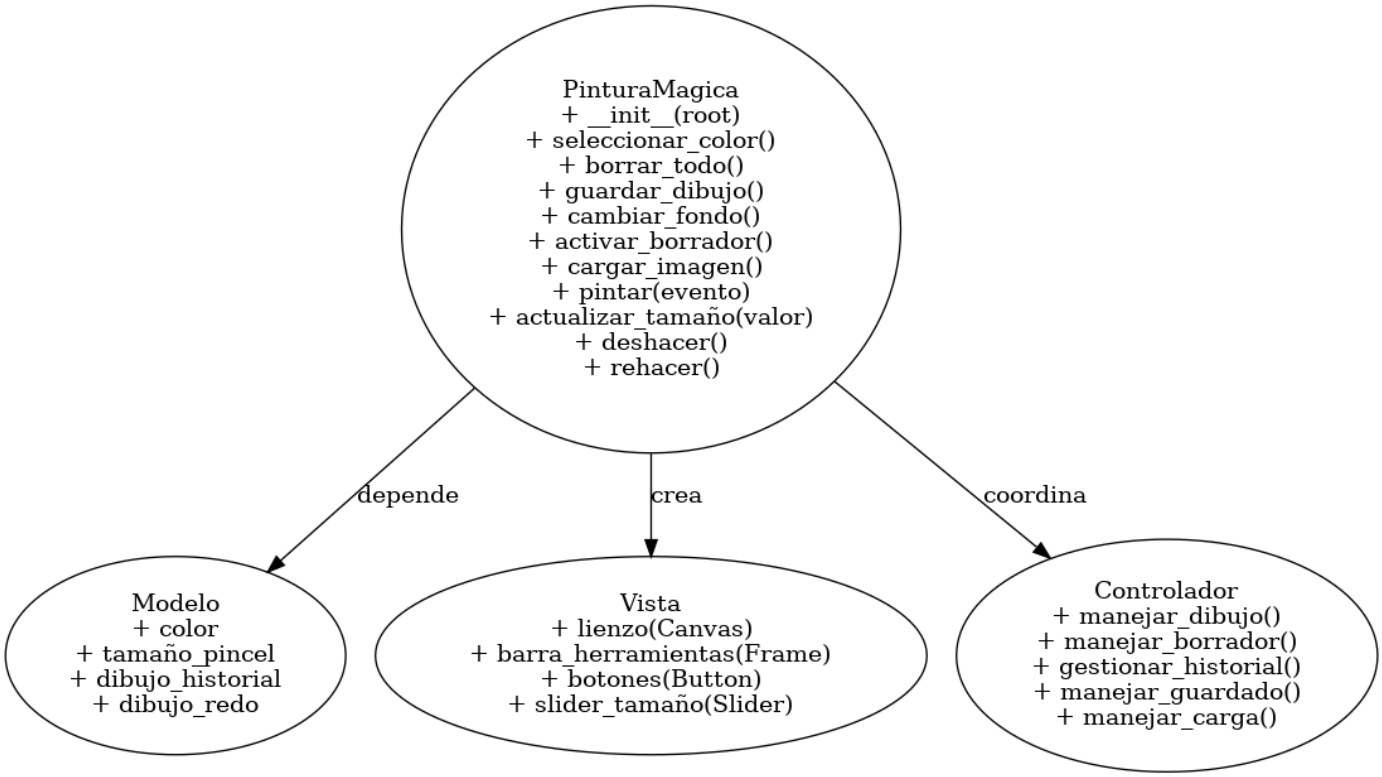
- **Color:** Almacena el color actual del pincel.
- **Tamaño del pincel:** Representa el grosor de las líneas dibujadas.
- **Historial de dibujo:** Listas para almacenar las acciones de dibujo (para deshacer/rehacer).
- **Modo borrador:** Variable booleana que indica si el borrador está activado.

Flujo de Operaciones

1. **Inicio de la aplicación:**
   Al abrir la aplicación, se presenta el lienzo vacío y los controles en la barra de herramientas.
2. **Dibujar:**
   El usuario mueve el mouse sobre el lienzo para dibujar. Puede ajustar el color y tamaño del pincel según sus necesidades.
3. **Editar:**
   Puede usar las herramientas para limpiar el lienzo, activar el borrador, o deshacer y rehacer acciones.
4. **Guardar o cargar:**
   Al finalizar, el usuario puede guardar el dibujo como imagen o cargar una imagen para personalizarla.

Diagrama de Clases

- **Creación del Diagrama de Clases**: Usa una herramienta de diagrama de clases (como Draw.io, Lucidchart o una herramienta de diagramación UML) para crear un diagrama de clases que muestre las principales clases y sus relaciones (herencia, asociaciones, agregación, composición).



- **Explicación del Diagrama**: Incluye un apartado donde se explique el diagrama de clases:

  El diagrama de clases para el proyecto **"Pintura Mágica"** está diseñado según el patrón **Modelo-Vista-Controlador (MVC)**. Este enfoque separa las responsabilidades en tres capas para facilitar la gestión del código y la expansión futura.

  - **Clases principales**: Enumera las clases clave del proyecto, su propósito y cómo interactúan entre sí.

- **`PinturaMagica` (Clase Principal):**
  - Propósito: Es la clase que inicializa la aplicación y coordina la interacción entre el modelo, la vista y el controlador.
  - Función: Administra la lógica general, crea la interfaz gráfica y responde a las acciones del usuario.

- **`Modelo`:**

- Propósito: Almacena y administra los datos principales del sistema, como el color del pincel, el tamaño, y el historial de acciones.
- Función: Actúa como la fuente única de verdad para los datos y asegura su consistencia.

- **`Vista:`**

  - Propósito: Define los elementos de la interfaz gráfica (lienzo, botones, sliders) y los presenta al usuario.
  - Función: Sirve como el medio a través del cual los usuarios interactúan con la aplicación.

- **`Controlador:`**

  - Propósito: Maneja las interacciones del usuario, como dibujar, borrar, o cambiar configuraciones.
  - Función: Gestiona la comunicación entre el modelo y la vista, y traduce las acciones del usuario en actualizaciones de los datos o cambios visuales.

    - **Relaciones**: Describe cómo se relacionan las clases entre sí, por ejemplo, herencia, asociaciones y composición.

- **`PinturaMagica → Modelo:`**

  - Relación: Dependencia.
  - Descripción: La clase principal accede al modelo para recuperar y actualizar los datos relacionados con el dibujo.

- **`PinturaMagica → Vista:`**

  - Relación: Asociación fuerte.
  - Descripción: Crea instancias de los elementos de la interfaz gráfica y los gestiona directamente.

- **`PinturaMagica → Controlador:`**

  - Relación: Coordinación.
  - Descripción: Delegación de tareas al controlador para manejar eventos y acciones específicas del usuario.

- **Atributos y Métodos**: Detalla los atributos de cada clase y los métodos públicos y privados, explicando su propósito y cómo contribuyen a la funcionalidad del sistema.

- **`PinturaMagica`**
  - **Atributos:**
    - `color`: Almacena el color actual del pincel.
    - `tamaño_pincel`: Define el tamaño del pincel.
    - `modo_borrador`: Indica si el borrador está activo o no.
  - **Métodos Públicos:**
    - `__init__`: Inicializa los componentes y establece la configuración inicial.
    - `seleccionar_color`: Abre un cuadro de diálogo para seleccionar un color.
    - `guardar_dibujo`: Guarda el dibujo en un archivo de imagen.
- **`Modelo`**
  - **Atributos:**
    - `dibujo_historial`: Lista que almacena las acciones realizadas en el lienzo (para deshacer/rehacer).
    - `dibujo_redo`: Lista que almacena las acciones deshechas (para rehacer).
  - **Métodos:**
    - Se gestionan internamente por `Controlador` o `PinturaMagica`.
- **`Vista`**
  - **Atributos:**
    - `lienzo`: Representa el área de dibujo principal.
    - `botones`: Elementos de la barra de herramientas para realizar acciones específicas.
    - `slider_tamaño`: Control para ajustar dinámicamente el grosor del pincel.
  - **Métodos Públicos:**
    - `configurar_elementos`: Configura y organiza los widgets de la interfaz gráfica.
- **`Controlador`**
  - **Métodos Públicos:**
    - `manejar_dibujo`: Permite dibujar o borrar según la configuración actual.
    - `gestionar_historial`: Administra el historial de acciones (deshacer/rehacer).
    - `manejar_guardado`: Convierte el lienzo en una imagen para guardarla.

Estructura del Proyecto

- **Modelo Vista-Controlador (MVC)**:
    - **Modelo**: Define las clases que modelan los datos y las reglas de negocio. Deben estar encapsuladas y ser independientes de la interfaz gráfica.

El **Modelo** se encarga de representar los datos del sistema y las reglas de negocio. En este caso, el modelo se ocupa de la lógica relacionada con el dibujo, como el color del pincel, el tamaño, el historial de acciones y el almacenamiento de los dibujos.

- **Responsabilidades del Modelo:**
    - Gestionar los datos de dibujo, como las coordenadas, el color y el tamaño del pincel.
    - Mantener el historial de las acciones (para deshacer y rehacer).
    - Manejar la lógica para guardar y cargar los dibujos.
    - Definir las reglas de negocio relacionadas con el dibujo (ejemplo: el tamaño del pincel no puede ser negativo).

Beneficios del Modelo:

- Está completamente aislado de la interfaz gráfica, lo que permite cambiar la vista sin afectar la lógica del modelo.
- Se encarga de almacenar y procesar los datos, permitiendo la manipulación y persistencia de estos.

    - **Vista**: Las interfaces de usuario (IU) que interactúan con el usuario final, usando Tkinter o CustomTkinter. Muestra ejemplos de cómo se crean las ventanas, botones, cuadros de texto, etc.

La **Vista** se encarga de la interfaz de usuario (UI), lo que significa que se ocupa de mostrar los datos del modelo de manera visual y recibir las entradas del usuario. En este caso, la vista se crea utilizando **Tkinter** o **CustomTkinter**, que permite crear la interfaz gráfica con botones, sliders y otros componentes.

- **Responsabilidades de la Vista:**
    - Crear y organizar los componentes visuales (botones, sliders, lienzo de dibujo).
    - Mostrar los cambios de datos del modelo (por ejemplo, cambios en el color del pincel, el tamaño, etc.).
    - Recoger las interacciones del usuario (por ejemplo, clics en botones, arrastrar el mouse para dibujar).
    - Invocar los métodos del controlador cuando el usuario interactúa con la UI.

Beneficios de la Vista:

- La interfaz gráfica está completamente separada de la lógica de negocio, lo que permite cambiar la apariencia sin afectar las funcionalidades del sistema.
- Responde a los eventos de usuario y actualiza la visualización cuando el modelo cambia.

    - **Controlador**: Las clases que manejan las interacciones del usuario y las actualizaciones de la vista y el modelo. Incluye lógica de validación y comunicación entre modelo y vista.

El **Controlador** es el intermediario entre la Vista y el Modelo. Es responsable de manejar la lógica de interacción del usuario, procesar las entradas y coordinar la actualización de la Vista y el Modelo.

- **Responsabilidades del Controlador:**
    - Gestionar los eventos del usuario (clics, cambios de configuración, etc.).
    - Actualizar el Modelo según las interacciones del usuario (por ejemplo, cambiar el color del pincel, agregar un nuevo dibujo).
    - Actualizar la Vista para reflejar los cambios del Modelo.

Beneficios del Controlador:

- Asegura que la lógica de negocio no se mezcle con la interfaz de usuario.
- Permite gestionar los eventos del usuario y coordinar entre el modelo y la vista.

Comentarios en el Código

- **Comentarios In-Line**: Añade comentarios en línea para explicar cada clase, método y segmento de código importante. Esto incluye la lógica detrás de los métodos, decisiones importantes y puntos críticos del código.

Requerimientos y Consideraciones

- **Bibliotecas Necesarias**: Indica todas las bibliotecas que se deben instalar (como Tkinter o CustomTkinter), cómo instalar y configurar el entorno para ejecutarlo correctamente.

Para ejecutar este proyecto, se deben instalar algunas bibliotecas externas. Las más importantes son **Tkinter** o **CustomTkinter** para la interfaz gráfica, y otras bibliotecas si es necesario. Aquí están las bibliotecas requeridas:

- **Tkinter:** Es una biblioteca estándar en Python para crear interfaces gráficas. En la mayoría de las instalaciones de Python, Tkinter ya está incluido. Si no está disponible, puede instalarse con:

pip install tk

CustomTkinter: Esta es una biblioteca adicional que proporciona un conjunto de widgets modernos y atractivos para Tkinter. Para instalarla, puedes usar:

pip install customtkinter

- **Instrucciones de Instalación**: Proporciona pasos detallados sobre cómo instalar y ejecutar la aplicación, incluyendo cualquier configuración adicional o pasos necesarios.

- **Instalar Python:**

- Asegúrate de tener **Python 3.7 o superior** instalado. Puedes descargarlo desde python.org.

- **Instalar las bibliotecas necesarias:**

- Abre la terminal o línea de comandos y ejecuta los siguientes comandos para instalar las bibliotecas necesarias:

pip install tk

pip install customtkinter

Descargar o clonar el código del proyecto:

- Si tienes el código en un archivo comprimido o en un repositorio, descomprímelo o clónalo.
- Si es un repositorio de GitHub, puedes clonar el proyecto con:

git clone <URL-del-repositorio>

Ejecutar la aplicación:

- Navega a la carpeta del proyecto desde la terminal o línea de comandos.
- Ejecuta el script principal del proyecto. Si el archivo principal es `main.py`, ejecuta:

python main.py

Configuración adicional:

- No debería haber configuraciones adicionales, pero si utilizas un entorno virtual, asegúrate de activar el entorno antes de instalar las dependencias.

Repositorio en GitHub

- **Crear el Repositorio**: Crea un repositorio en GitHub para alojar el código fuente del proyecto.
- **Subir el Código**: Asegúrate de que el código esté bien documentado y comentado antes de subirlo.
- **Documentación Adicional**: Incluye un archivo README.md con toda la documentación del proyecto (descrita anteriormente) y un flujo de instalación y ejecución claro.
- **Versionado**: Utiliza branches para organizar diferentes etapas del desarrollo y mantener el historial de cambios.

Que entregar:

- Un link con el repositorio en github
- Dentro del repositorio reposaran los códigos del software
- Y el documento donde pondrán lo referente al diagrama de clases, objetivos, requerimientos y consideraciones.