**ImageCraft项目开发说明文档**

本文档用于说明关于ImageCraft项目的开发，以便了解项目需求和实施细节。

1. **项目概述**

ImageCraft是一个类似于kolourPaint的图片编辑软件，旨在提供一个易于使用的界面，让用户在使用图像处理、编辑、绘制等功能时获得更好的体验，使用户能够获得预期的效果。

**二、技术要求**

1.开发语言

采用QML和C++交互进行开发

2.开发平台

QtCreater

3.开发需要

OpenCV 集成：部分功能利用 OpenCV 提供的高级图像处理算法和计算机视觉功能，如图像修复和复杂滤镜处理。

**三、开发过程**

1.整体划分

首先划分软件的ui界面，分为菜单栏、工具栏、图片展示模块、底部状态栏四类模块（使用qml和js实现）。

然后根据ui界面来实现每个模块的功能（主要使用c++实现）。

最后把每个模块兼容到一起，组合成完整的ImageCraft软件（使用c++与qml交互使用）。

2.个人分工

曾艳：ui界面实现，“色板”及状态栏实现，“图像、视图、帮助、设置”菜单，“文字、吸管、缩放”工具

任天翔：ui界面实现，“文件、编辑”菜单，“选择”工具，代码逻辑优化和整合，Bug修复，图片展示模块

战学才：ui界面实现，“文件、滤镜”菜单，“移动、矩阵、线条、画笔、橡皮擦”工具，图标设计

3.开发日志

2024-6-18

菜单栏UI的设计以及整体布局。

2024-6-19

工具栏、状态栏、图片展示模块的UI设计。

2024-6-20上午

添加上述四类模块的UI的qml组件；实现了“文件”菜单保存、另存为、打开、新建功能；构建文件改为cmake。

2024-6-20下午

实现了“文件”菜单的导出、图片格式转换、关闭图片、关闭全部、刷新、截取当前屏幕功能；实现了工具栏的“吸管”、“移动”功能；修改了保存、保存为的逻辑结构。

2024-6-21

实现了工具栏的“画笔”工具，“放缩”工具，优化了图片展示和关闭的处理逻辑。

2024-6-22

实现了新的退出和保存交互框，工具栏的“矩阵”工具，底部画板模块UI，优化了放缩工具 。

2024-6-23

实现了“编辑”菜单的撤销和重做功能，工具栏的“椭圆”工具。

2024-6-24

在c++端实现了“状态栏”的画板颜色选择、“图像”菜单的垂直翻转、水平翻转功能、“编辑”菜单关于移动的撤销和重做；优化了画笔和矩阵绘制的定位问题，重写图层组件。

2024-6-25

实现了“编辑”菜单关于缩放和图层可见性的撤销与重做、图像”菜单的旋转功能和菜单下的缩放功能；优化了画笔和矩阵的绘制逻辑，以创建画布图层代替直接在原图上的绘制。

2024-6-26

实现了“图片展示模块”的图层模块和右侧展示模块的信号连接，“图像”菜单的用户自定义的旋转功能，“画笔”工具的画笔大小更改、钢笔和喷漆工具；优化了绘图的实现逻辑，修复了缩放撤销bug。

2024-7-6

实现了工具栏的“线条”工具、“文字”工具、“编辑”菜单关于鼠标右键的撤销与重做；修复了绘图的定位bug。

2024-7-7

优化了从画板中提取画笔、文字颜色、矩阵的描边与填充、多边形功能；修复撤销与重做的诸多bug

2024-7-8

实现了工具栏的“橡皮擦”、“选择”工具；“图像”、“设置”、“滤镜”菜单；优化了“吸管”工具；

2024-7-9

实现了“帮助”菜单；优化了选择的自由缩放、复制粘贴功能，更多的滤镜效果。

1. **集成实现**
2. CMake构建

使用CMake代替qmake，一方面CMake提供了更多的配置选择和扩展功能，相比于qmake更适用于大型的软件开发和跨平台开发，另一方面CMake更容易集成和使用其他方法库，便利了OpenCV的使用

1. C++与QML的交互实现

通过使用QML\_SINGLETON宏来实现把C++类自动注册为QML组件、Q\_INVOKABLE宏来实现把C++类中的方法开放给QML端使用，实现了C++向QML的交互工作。

通过使用QMetaObject::invokeMethod()方法来实现在C++类中调用QML中的Js方法、定义QVariant和QObject类型来接收并传参，通过在C++端使用connect连接来解决QML的异步问题，实现了QML向C++的交互工作

通过在C++端发出信号、QML端接收信号并调用C++端的方法来实现QML与C++的双向交互

1. OpenCV4的集成实现

通过使用OpenCV计算机视觉库来完成对复杂功能的实现，主要通过把QImage类型转换成cv::Mat类型并进行对应的opencv处理，处理后把cv::Mat类型转换回QImage类型用于展示和保存。

下面是在cmake中使用opencv4的代码：  
target\_compile\_features(appImageCraft PRIVATE cxx\_std\_20)

set\_target\_properties(

appImageCraft

PROPERTIES AUTORCC TRUE

MACOSX\_BUNDLE\_GUI\_IDENTIFIER my.example.com

MACOSX\_BUNDLE\_BUNDLE\_VERSION ${PROJECT\_VERSION}

MACOSX\_BUNDLE\_SHORT\_VERSION\_STRING

${PROJECT\_VERSION\_MAJOR}.${PROJECT\_VERSION\_MINOR}

MACOSX\_BUNDLE TRUE

WIN32\_EXECUTABLE TRUE)

target\_link\_libraries(appImageCraft PRIVATE Qt6::Quick)

target\_link\_libraries(

appImageCraft

PRIVATE Qt6::Quick

# /usr/lib \ # OpenCV的lib路径

opencv\_core

opencv\_imgproc

opencv\_highgui

opencv\_imgcodecs

opencv\_xphoto)

# 其他需要的OpenCV模块库)

include(GNUInstallDirs)

install(

TARGETS appImageCraft

BUNDLE DESTINATION .

LIBRARY DESTINATION ${CMAKE\_INSTALL\_LIBDIR}

RUNTIME DESTINATION ${CMAKE\_INSTALL\_BINDIR})

include\_directories(/usr/include/opencv4)

1. **开发心得体会**
2. **信号与槽的连接**：在C++和QML之间进行通信的一个重要机制是使用信号（Signals）和槽（Slots）。通过在C++中声明信号，并在QML中连接这些信号到JavaScript函数或者其他QML对象的槽，可以实现跨语言层面的通信。这种机制非常有力，因为它允许从C++代码触发QML界面的更新或者响应用户交互。
3. **属性绑定**：QML允许在其语言内部进行属性绑定，这意味着你可以直接将QML对象的属性绑定到C++对象的属性或者函数的返回值上。通过将C++对象的属性注册到QML类型系统中，或者使用Q\_PROPERTY宏来声明属性，可以使得QML能够直接访问这些属性。这种方式非常便捷，特别是在处理界面元素状态同步时。**3. 模型/视图的集成**：在使用QML时，经常会遇到需要展示动态数据的情况，这时可以通过将C++中的数据模型（如QAbstractListModel或自定义的数据模型类）暴露给QML，从而在QML中使用模型/视图架构。这种方式不仅提供了高效的数据展示能力，还可以利用QML的强大视图控件来展示和编辑数据。

**4.性能考量**：

尽管QML提供了很高的界面表现能力，但在处理大量数据或者复杂逻辑时，需要谨慎考虑性能问题。尤其是在C++与QML交互时，频繁的信号传递和属性更新可能会导致性能下降。

**5.实验不足：**

**项目时间有点紧迫，项目管理经验不足，项目周期管理不合理，迭代周期过长，对项目预期过多，功能兼容性没有达到预期，内存优化不足**